




# Reference

---



VERSION 4.5

**VisiBroker® for Java™**

Inprise Corporation, 100 Enterprise Way  
Scotts Valley, CA 95066-3249

Inprise may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1996, 2000 Inprise Corporation. All rights reserved. All Inprise and Borland brands and product names are trademarks or registered trademarks of Inprise Corporation. Java and all Java-based marks are trademarks or registered trademarks in the United States and other countries. Other brands and product names are trademarks or registered trademarks of their respective owners.

Printed in the U.S.A.      PDF VJD0045WW21001

# Contents

## Chapter 1

### Preface

1-1

What's new in 4.0 . . . . .	1-1
What's new in 4.1 . . . . .	1-3
What's new in 4.5 . . . . .	1-3
Manual conventions . . . . .	1-4
Typographic conventions . . . . .	1-4
Platform conventions . . . . .	1-4
Where to find additional information . . . . .	1-5
Contacting developer support . . . . .	1-5

## Chapter 2

### Programmer tools

2-1

Options . . . . .	2-1
General options . . . . .	2-1
idl2ir . . . . .	2-1
ir2idl . . . . .	2-2
idl2java . . . . .	2-3
java2idl . . . . .	2-5
java2iio . . . . .	2-6
vbj . . . . .	2-9
vbjc . . . . .	2-10
Specifying the classpath . . . . .	2-10
Specifying the JVM . . . . .	2-11
Other tools . . . . .	2-12

## Chapter 3

### IDL to Java mapping

3-1

Names . . . . .	3-1
Reserved names . . . . .	3-2
Reserved words . . . . .	3-2
Modules . . . . .	3-3
Basic types . . . . .	3-3
IDL type extensions . . . . .	3-4
Holder classes . . . . .	3-5
Boolean . . . . .	3-8
Char . . . . .	3-8
Octet . . . . .	3-8
String . . . . .	3-8
WString . . . . .	3-9
Integer types . . . . .	3-9
Floating point types . . . . .	3-9
Helper classes . . . . .	3-9
Constants . . . . .	3-10
Constants within an interface . . . . .	3-10
Constants NOT within an interface . . . . .	3-11

Constructed types . . . . .	3-11
Enum . . . . .	3-11
Struct . . . . .	3-13
Union . . . . .	3-14
Sequence . . . . .	3-16
Array . . . . .	3-17
Interfaces . . . . .	3-18
Passing parameters . . . . .	3-20
Server implementation with inheritance . . . . .	3-21
Server implementation with delegation . . . . .	3-22
Interface scope . . . . .	3-23
Mapping for exceptions . . . . .	3-23
User-defined exceptions . . . . .	3-23
System exceptions . . . . .	3-24
Mapping for the Any type . . . . .	3-25
Mapping for certain nested types . . . . .	3-25
Mapping for Typedef . . . . .	3-25
Simple IDL types . . . . .	3-26
Complex IDL types . . . . .	3-26

## Chapter 4

### Generated interfaces and classes

4-1

Overview . . . . .	4-1
Signature and operations classes . . . . .	4-1
Ancillary classes . . . . .	4-2
Portability stub and skeleton interfaces . . . . .	4-2
<interface_name>Operations . . . . .	4-3
<type_name>Helper . . . . .	4-3
Methods for all Helper classes . . . . .	4-3
Methods generated for interfaces . . . . .	4-4
Methods generated for object wrappers . . . . .	4-5
<type_name>Holder . . . . .	4-6
Member data . . . . .	4-7
Methods . . . . .	4-7
_<interface_name>Stub . . . . .	4-7
<interface_name>POA . . . . .	4-7
<interface_name>POATie . . . . .	4-8
Methods . . . . .	4-8

## Chapter 5

### Core interfaces and classes

5-1

BindOptions . . . . .	5-1
IDL definition . . . . .	5-1
BindOptions constructors . . . . .	5-2
BOA . . . . .	5-2
IDL definition . . . . .	5-3

BOA methods . . . . .	5-3
CompletionStatus . . . . .	5-5
IDL definition . . . . .	5-5
CompletionStatus methods . . . . .	5-5
Context . . . . .	5-6
IDL definition . . . . .	5-6
Context methods . . . . .	5-6
InvalidName . . . . .	5-8
Object . . . . .	5-8
org.omg.CORBA Object definition . . . . .	5-8
org.omg.Object methods . . . . .	5-9
VisiBroker extension to Object . . . . .	5-11
VisiBroker extension to Object methods . . . . .	5-12
ORB . . . . .	5-13
JDK's ORB definition . . . . .	5-14
JDK ORB methods . . . . .	5-15
OMG ORB definition . . . . .	5-22
VisiBroker ORB extensions . . . . .	5-22
VisiBroker ORB methods . . . . .	5-23
PortableServer.AdapterActivator . . . . .	5-25
PortableServer.AdapterActivator methods . . . . .	5-25
PortableServer.Current . . . . .	5-26
PortableServer.Current methods . . . . .	5-26
PortableServer.POA . . . . .	5-26
PortableServer.POA methods . . . . .	5-26
PortableServer.POAManager . . . . .	5-36
PortableServer.POAManager methods . . . . .	5-37
PortableServer.ServantActivator . . . . .	5-37
PortableServer.ServantActivator methods . . . . .	5-37
PortableServer.ServantLocator . . . . .	5-38
PortableServer.ServantLocator methods . . . . .	5-39
PortableServer.ServantManager . . . . .	5-40
Principal . . . . .	5-40
IDL definition . . . . .	5-40
Principal methods . . . . .	5-41

## Chapter 6

### **Dynamic interfaces and classes 6-1**

Any . . . . .	6-1
Any methods . . . . .	6-1
Any extraction methods . . . . .	6-2
Any Insertion methods . . . . .	6-3
ARG_IN . . . . .	6-4
Variables . . . . .	6-4
ARG_INOUT . . . . .	6-4
Variables . . . . .	6-4
ARG_OUT . . . . .	6-4
Variables . . . . .	6-5
ContextList . . . . .	6-5

IDL definition . . . . .	6-5
ContextList methods . . . . .	6-5
DynAny . . . . .	6-6
Important usage restrictions . . . . .	6-6
DynAny methods . . . . .	6-7
DynAny Extraction methods . . . . .	6-8
DynAny Insertion methods . . . . .	6-8
DynArray . . . . .	6-9
Important usage restrictions . . . . .	6-9
DynArray methods . . . . .	6-10
DynAnyFactory . . . . .	6-10
Important usage restrictions . . . . .	6-10
DynAnyFactory methods . . . . .	6-10
DynEnum . . . . .	6-11
Important usage restrictions . . . . .	6-11
DynEnum methods . . . . .	6-11
DynFixed . . . . .	6-12
DynFixed methods . . . . .	6-12
DynSequence . . . . .	6-12
Important usage restrictions . . . . .	6-12
DynSequence methods . . . . .	6-13
DynStruct . . . . .	6-13
Important usage restrictions . . . . .	6-14
DynStruct methods . . . . .	6-14
DynUnion . . . . .	6-14
Important usage restrictions . . . . .	6-15
DynUnion methods . . . . .	6-15
DynValue . . . . .	6-16
DynValue methods . . . . .	6-16
DynamicImplementation . . . . .	6-17
Constructors . . . . .	6-17
DynamicImplementation methods . . . . .	6-17
Environment . . . . .	6-18
Environment methods . . . . .	6-18
ExceptionList . . . . .	6-18
IDL definition . . . . .	6-18
ExceptionList methods . . . . .	6-19
InputStream . . . . .	6-19
InputStream methods . . . . .	6-19
Invalid . . . . .	6-20
InvalidSeq . . . . .	6-21
NamedValue . . . . .	6-21
IDL definition . . . . .	6-21
NameValue methods . . . . .	6-21
NameValuePair . . . . .	6-22
NameValuePair variables . . . . .	6-22
NameValuePair constructors . . . . .	6-22
NVList . . . . .	6-22
IDL definition . . . . .	6-22

NVList methods . . . . .	6-23
OutputStream . . . . .	6-24
OutputStream methods . . . . .	6-24
Request . . . . .	6-25
IDL definition. . . . .	6-26
Request methods . . . . .	6-26
ServerRequest . . . . .	6-29
IDL definition. . . . .	6-29
ServerRequest methods . . . . .	6-29
TCKind . . . . .	6-30
IDL definition. . . . .	6-30
TCKind methods . . . . .	6-31
TypeCode . . . . .	6-31
IDL definition. . . . .	6-31
TypeCode methods. . . . .	6-32
UnknownUserException . . . . .	6-34

## Chapter 7

### Interface repository

#### **interfaces and classes** **7-1**

AbstractInterfaceDef. . . . .	7-1
AbstractInterfaceDef methods . . . . .	7-1
AliasDef. . . . .	7-3
AliasDef methods . . . . .	7-4
ArrayDef . . . . .	7-4
ArrayDef methods . . . . .	7-4
AttributeDef . . . . .	7-5
AttributeDef methods . . . . .	7-5
AttributeDescription. . . . .	7-6
AttributeDescription variables . . . . .	7-6
AttributeDescription methods . . . . .	7-6
AttributeMode . . . . .	7-7
AttributeMode enumeration elements. . . . .	7-7
ConstantDef . . . . .	7-8
ConstantDef methods . . . . .	7-8
ConstantDescription. . . . .	7-8
ConstantDescription variables . . . . .	7-9
Constant Description methods . . . . .	7-9
Contained. . . . .	7-10
IDL definition. . . . .	7-10
Contained methods . . . . .	7-10
ContainedPackage.Description. . . . .	7-12
ContainedPackage.Description variables . . . . .	7-12
ContainedPackage.Description methods . . . . .	7-12
Container . . . . .	7-12
IDL definition. . . . .	7-13
Container methods. . . . .	7-14
ContainerPackage.Description . . . . .	7-19
ContainerPackage.Description variables . . . . .	7-19

ContainerPackage.Description methods . . . . .	7-19
DefinitionKind . . . . .	7-20
DefinitionKind methods . . . . .	7-20
DefinitionKind enumeration values . . . . .	7-20
EnumDef. . . . .	7-21
EnumDef methods . . . . .	7-21
ExceptionDef . . . . .	7-21
ExceptionDef methods . . . . .	7-22
ExceptionDescription. . . . .	7-22
ExceptionDescription variables . . . . .	7-22
ExceptionDescription methods . . . . .	7-23
FixedDef . . . . .	7-23
FixedDef methods. . . . .	7-23
FullValueDescription . . . . .	7-24
FullValueDescription variables . . . . .	7-24
FullValueDescription methods. . . . .	7-25
IDLType . . . . .	7-25
IDL definition . . . . .	7-26
IDLType methods . . . . .	7-26
InterfaceDef . . . . .	7-26
IDL definition . . . . .	7-26
InterfaceDef methods . . . . .	7-27
InterfaceDefPackage.FullInterfaceDescription. . . . .	7-28
InterfaceDefPackage.FullInterfaceDescription variables . . . . .	7-28
InterfaceDefPackage.FullInterfaceDescription methods. . . . .	7-29
InterfaceDescription . . . . .	7-29
InterfaceDescription variables . . . . .	7-30
InterfaceDescription methods . . . . .	7-30
IRObjct . . . . .	7-30
IDL definition . . . . .	7-31
IRObjct methods . . . . .	7-31
ModuleDef. . . . .	7-31
ModuleDescription . . . . .	7-31
ModuleDescription variables . . . . .	7-31
ModuleDescription methods. . . . .	7-32
NativeDef . . . . .	7-32
OperationDef . . . . .	7-32
OperationDef methods . . . . .	7-33
OperationDescription . . . . .	7-34
OperationDescription variables . . . . .	7-34
OperationDescription methods . . . . .	7-35
OperationMode . . . . .	7-35
ParameterDescription . . . . .	7-36
ParameterDescription variables . . . . .	7-36
ParameterDescription methods . . . . .	7-36
ParameterMode . . . . .	7-37
PrimitiveDef. . . . .	7-37

PrimitiveDef method . . . . .	7-37
PrimitiveKind . . . . .	7-37
PrimitiveKind methods . . . . .	7-38
PrimitiveKind constants . . . . .	7-38
Repository . . . . .	7-38
Repository methods . . . . .	7-39
SequenceDef . . . . .	7-40
SequenceDef methods . . . . .	7-40
StringDef . . . . .	7-41
StringDef methods . . . . .	7-41
StructDef . . . . .	7-42
StructDef methods . . . . .	7-42
StructMember . . . . .	7-42
StructMember variables . . . . .	7-42
StructMember methods . . . . .	7-43
TypedefDef . . . . .	7-43
TypeDescription . . . . .	7-43
TypeDescription variables . . . . .	7-43
TypeDescription methods . . . . .	7-44
UnionDef . . . . .	7-44
UnionDef methods . . . . .	7-44
UnionMember . . . . .	7-45
UnionMember variables . . . . .	7-45
UnionMember methods . . . . .	7-45
ValueBoxDef . . . . .	7-46
ValueBoxDef methods . . . . .	7-46
ValueDef . . . . .	7-46
ValueDef methods . . . . .	7-47
ValueDescription . . . . .	7-49
ValueDescription variables . . . . .	7-49
ValueDescription methods . . . . .	7-50
ValueMemberDef . . . . .	7-50
ValueMemberDef methods . . . . .	7-50
WstringDef . . . . .	7-51
WStringDef methods . . . . .	7-51

## Chapter 8

### Activation interfaces and classes 8-1

ActivationImplDef . . . . .	8-1
ActivationImplDef methods . . . . .	8-1
Activator . . . . .	8-2
Activator methods . . . . .	8-2
CreationImplDef . . . . .	8-3
IDL definition . . . . .	8-3
Activation policy . . . . .	8-3
Examples . . . . .	8-4
Environment variables . . . . .	8-4
Environment variables that are propagated or passed explicitly . . . . .	8-5
CreationImplDef methods . . . . .	8-5

ImplementationDef . . . . .	8-7
OAD . . . . .	8-7
ImplementationStatus . . . . .	8-8
OAD methods . . . . .	8-9

## Chapter 9

### Naming service interfaces and classes 9-1

NamingContext . . . . .	9-1
IDL definition . . . . .	9-1
NamingContext methods . . . . .	9-2
NamingContextExt . . . . .	9-6
IDL definition . . . . .	9-6
NamingContextExt methods . . . . .	9-7
Binding and BindingList . . . . .	9-8
IDL definition . . . . .	9-9
BindingIterator . . . . .	9-9
IDL definition . . . . .	9-9
BindingIterator methods . . . . .	9-10
NamingContextFactory . . . . .	9-10
IDL definition . . . . .	9-10
NamingContextFactory methods . . . . .	9-11
ExtendedNamingContextFactory . . . . .	9-11
IDL definition . . . . .	9-11
ExtendedNamingContextFactory methods . . . . .	9-12

## Chapter 10

### Event service interfaces and classes 10-1

ConsumerAdmin . . . . .	10-1
IDL definition . . . . .	10-1
Java definition . . . . .	10-1
ConsumerAdmin methods . . . . .	10-2
EventChannel . . . . .	10-2
Java definition . . . . .	10-2
EventChannel methods . . . . .	10-2
EventLibrary (Java) . . . . .	10-3
Java definition . . . . .	10-3
EventLibrary methods . . . . .	10-3
ProxyPullConsumer . . . . .	10-4
IDL definition . . . . .	10-4
Java definition . . . . .	10-5
ProxyPullConsumer method . . . . .	10-5
ProxyPushConsumer . . . . .	10-5
IDL definition . . . . .	10-5
Java definition . . . . .	10-5
ProxyPushConsumer method . . . . .	10-6
ProxyPullSupplier . . . . .	10-6
IDL definition . . . . .	10-6
Java definition . . . . .	10-6

ProxyPullSupplier method . . . . .	10-6
ProxyPushSupplier . . . . .	10-7
IDL definition . . . . .	10-7
Java definition . . . . .	10-7
ProxyPushSupplier method . . . . .	10-7
PullConsumer . . . . .	10-7
IDL definition . . . . .	10-8
Java definition . . . . .	10-8
PullConsumer method . . . . .	10-8
PushConsumer . . . . .	10-8
IDL definition . . . . .	10-8
Java definition . . . . .	10-8
PushConsumer methods . . . . .	10-9
PullSupplier . . . . .	10-9
IDL definition . . . . .	10-9
Java definition . . . . .	10-9
PullSupplier methods . . . . .	10-10
PushSupplier . . . . .	10-10
IDL definition . . . . .	10-10
Java definition . . . . .	10-10
PushSupplier method . . . . .	10-11
SupplierAdmin . . . . .	10-11
IDL definition . . . . .	10-11
Java definition . . . . .	10-11
SupplierAdmin methods . . . . .	10-11

## Chapter 11

### Exceptions classes 11-1

Introduction . . . . .	11-1
SystemException . . . . .	11-1
SystemException attributes . . . . .	11-2
UserException . . . . .	11-3
UserException constructor . . . . .	11-3

## Chapter 12

### Interceptor and object wrapper interfaces and classes 12-1

Introduction . . . . .	12-1
InterceptorManagers . . . . .	12-1
IOR templates . . . . .	12-2
InterceptorManager . . . . .	12-2
InterceptorManagerControl . . . . .	12-2
Import statement . . . . .	12-2
InterceptorManagerControl method . . . . .	12-3
BindInterceptor . . . . .	12-3
Import statement . . . . .	12-3
BindInterceptor methods . . . . .	12-3
BindInterceptorManager . . . . .	12-4
Import statement . . . . .	12-5

BindInterceptorManager method . . . . .	12-5
ClientRequestInterceptor . . . . .	12-5
Import statement . . . . .	12-5
ClientRequestInterceptor methods . . . . .	12-5
ClientRequestInterceptorManager . . . . .	12-7
Import statement . . . . .	12-7
ClientRequestInterceptorManager methods . . . . .	12-7
POALifeCycleInterceptor . . . . .	12-7
Import statement . . . . .	12-7
POALifeCycleInterceptor methods . . . . .	12-7
POALifeCycleInterceptorManager . . . . .	12-8
Import statement . . . . .	12-8
POALifeCycleInterceptorManager method . . . . .	12-8
ActiveObjectLifeCycleInterceptor . . . . .	12-9
Import statement . . . . .	12-9
ActiveObjectLifeCycleInterceptor methods . . . . .	12-9
ActiveObjectLifeCycleInterceptorManager . . . . .	12-9
Import statement . . . . .	12-10
ActiveObjectLifeCycleInterceptorManager method . . . . .	12-10
ForwardRequestException . . . . .	12-10
Variables . . . . .	12-10
ServerRequestInterceptor . . . . .	12-10
Import statement . . . . .	12-10
ServerRequestInterceptor methods . . . . .	12-11
ServerRequestInterceptorManager . . . . .	12-12
Import statement . . . . .	12-12
ServerRequestInterceptorManager method . . . . .	12-12
IORCreationInterceptor . . . . .	12-13
Import statement . . . . .	12-13
IORInterceptor method . . . . .	12-13
IORInterceptorManager . . . . .	12-13
Import statement . . . . .	12-13
IORInterceptorManager method . . . . .	12-14
Location . . . . .	12-14
Closure . . . . .	12-14
ExtendedClosure . . . . .	12-14
ChainUntypedObjectWrapperFactory . . . . .	12-15
Import statement . . . . .	12-15
ChainUntypedObjectWrapperFactory methods . . . . .	12-16
UntypedObjectWrapper . . . . .	12-16
UntypedObjectWrapper methods . . . . .	12-17
UntypedObjectWrapperFactory . . . . .	12-18
Import statement . . . . .	12-18
UntypedObjectWrapperFactory method . . . . .	12-18

## Chapter 13

### Quality of Service interfaces and classes 13-1

PolicyManager . . . . .	13-1
IDL definition . . . . .	13-2
Policy Manager methods . . . . .	13-2
PolicyCurrent . . . . .	13-3
IDL definition . . . . .	13-3
Object . . . . .	13-3
org.omg.CORBA.Object methods . . . . .	13-3
com.inprise.vbroker.CORBA.Object methods . . . . .	13-4
RebindPolicy . . . . .	13-5
IDL definition . . . . .	13-6
Policy Values . . . . .	13-6
RelativeConnectionTimeoutPolicy . . . . .	13-8
IDL definition . . . . .	13-8
DeferBindPolicy . . . . .	13-9
IDL definition . . . . .	13-9
ExclusiveConnectionPolicy . . . . .	13-9
IDL definition . . . . .	13-9
SyncScopePolicy . . . . .	13-10
IDL definition . . . . .	13-10
QoS exceptions . . . . .	13-11

## Chapter 14

### IOP and IIOP interfaces and classes 14-1

IIOP.ProfileBody . . . . .	14-1
IDL definition . . . . .	14-1
IIOP.ProfileBody variables . . . . .	14-2
IIOP.ProfileBody constructors . . . . .	14-2
IIOP.IORValue . . . . .	14-2
IDL definition . . . . .	14-3
IIOP.IORValue variables . . . . .	14-3
IOP.ServiceContext . . . . .	14-3
IDL definition . . . . .	14-4
IIOP.ServiceContext variables . . . . .	14-4
IIOP.ServiceContext constructors . . . . .	14-4
IOP.TaggedProfile . . . . .	14-4
IDL definition . . . . .	14-4
IIOP.TaggedProfile variables . . . . .	14-5
IIOP.TaggedProfile constructors . . . . .	14-5

## Chapter 15

### RMI interfaces and classes 15-1

PortableRemoteObject . . . . .	15-1
Constructors . . . . .	15-1
PortableRemoteObject methods . . . . .	15-2

## Chapter 16

### URL Naming interfaces and classes 16-1

Resolver . . . . .	16-1
Resolver methods . . . . .	16-2

## Chapter 17

### Location Service interfaces and classes 17-1

Agent . . . . .	17-1
IDL definition . . . . .	17-1
Agent methods . . . . .	17-2
Desc . . . . .	17-4
IDL definition . . . . .	17-5
Desc variables . . . . .	17-5
Desc constructor . . . . .	17-5
Desc methods . . . . .	17-6
Fail . . . . .	17-6
Fail variables . . . . .	17-6
TriggerDesc . . . . .	17-6
IDL definition . . . . .	17-7
TriggerDesc variables . . . . .	17-7
TriggerDesc constructor . . . . .	17-7
TriggerDesc methods . . . . .	17-7
TriggerHandler . . . . .	17-8
IDL definition . . . . .	17-8
TriggerHandler methods . . . . .	17-8

## Appendix A

### Using command-line options A-1

How to set ORB and BOA options . . . . .	A-1
Using vbj with command-line arguments . . . . .	A-1
Using vbj and starting your executable . . . . .	A-2
Applets . . . . .	A-2
Setting properties programmatically using methods . . . . .	A-2
BOA_init() method . . . . .	A-2
BOA options . . . . .	A-3
ORB.init() method . . . . .	A-4
ORB options . . . . .	A-5
ORBbackcompat option . . . . .	A-7
Location Service options . . . . .	A-7

## Appendix B

### Using VisiBroker properties B-1

JAVA RMI over IIOP properties . . . . .	B-1
OSAgent properties . . . . .	B-2
ORB properties . . . . .	B-3
POA properties . . . . .	B-7



Server Manager properties . . . . .	B-7	Server-Side Thread Session IIOP_TS/IIOP_TS	
Location Service properties . . . . .	B-7	Connection properties . . . . .	B-11
Event Service properties . . . . .	B-8	Server-Side Thread Session BOA_TS/BOA_TS	
Naming Service properties . . . . .	B-8	Connection properties . . . . .	B-12
OAD properties . . . . .	B-8	Server-Side Thread Pool IIOP_TP/IIOP_TP	
Interface Repository properties. . . . .	B-9	Connection properties . . . . .	B-13
URL Naming properties. . . . .	B-9	Server-Side Thread Pool BOA_TP/BOA_TP	
Client-Side Connection properties . . . . .	B-10	Connection properties . . . . .	B-14
Client-Side In Process Connection properties. .B-10		Properties that support bidirectional	
Server-Side Server Engine properties . . . . .	B-11	communication . . . . .	B-14



# Preface

VisiBroker allows you to develop and deploy distributed object-based applications, as defined in the Common Object Request Broker (CORBA) specification.

The VisiBroker for Java *Reference* provides a description of the classes and interfaces supplied with VisiBroker for Java, the programmer tools, and command-line options. It has been updated to reflect the latest VisiBroker release. It is written for Java programmers who are familiar with object-oriented development.

This Preface highlights the latest features, and identifies typographical and platform conventions used throughout the manual. It also tells you where to find additional information about Common Object Request Broker Architecture (CORBA) and the remaining VisiBroker for Java documentation set, and how to contact Inprise developer support.

## What's new in 4.0

---

The VisiBroker 4.0 features and enhancements include:

- **CORBA 2.3 compliance:** VisiBroker for Java is fully compliant with the CORBA specification (version 2.3) from the Object Management Group (OMG). For more details or the latest information, refer to the CORBA specification located at <http://www.omg.org/>.
- **CORBA 2.3 language mappings:** These includes the new typecodes and new methods for typecodes. The new typecodes are designed to be more maintainable by having multiple typecode classes that deal with functionality specific to each typecode. See Chapter 3, "IDL to Java mapping," for more information on typecodes.
- **Naming Service:** The new VisiBroker Naming Service provides clustering and fault tolerance features and persistence is handled differently.

Clustering allows you to associate a number of bindings with a single name. Fault tolerance features include failover and load balancing capabilities. To maintain persistence, you now can keep the namespace in a relational database. Previous versions stored the namespace in a flat logging file. The `corbaloc` and `corbaname` provide stringified object references which can be used in an Internet environment. This allows you to refer to objects by a URL. See Chapter 9, “Naming service interfaces and classes,” for a description of the Naming Service classes and interfaces and Chapter 18, “Using the Naming Service,” in the *VisiBroker for Java Programmer's Guide* for a description of how to use the Naming Service.

- **Portable Object Adaptor (POA):** The POA offers portability on the server side. This feature replaces the Basic Object Adapter (BOA). Although BOA is being deprecated, VisiBroker 4.x still supports BOA functionality. See Chapter 5, “Core interfaces and classes,” for a description of the POA classes and interfaces and Chapter 7, “Using POAs,” in the *VisiBroker for Java Programmer's Guide* for an explanation of how to use the POA interfaces.
- **Objects by value (OBV) or Value types:** Previous versions of CORBA allowed you to pass objects between clients and servers by reference. However, CORBA 2.3 allows you to pass objects by value between clients and servers using VisiBroker. See Chapter 3, “IDL to Java mapping,” for more information.
- **Property Management:** This feature provides you with a way to centralize management of properties. You can set properties that are available to you in the Server Manager Console. See Appendix B, “Using VisiBroker properties,” for a list of the VisiBroker for Java properties and Chapter 13, “Using the Server Manager,” in the *VisiBroker for Java Programmer's Guide* for more information on using the Server Manager browser in the VisiBroker Console.
- **Quality of Service (QoS):** This feature allows you to define properties that influence how connections are made. You perform client- side policy management by setting properties that are associated with connections or client/server pairs. See Chapter 13, “Quality of Service interfaces and classes,” for a description of the QoS interfaces, classes, and policies.
- **Interceptors and Object Wrappers:** This feature has been upgraded to CORBA 2.3 specifications. The ORB provides a set of APIs known as interceptors which provide a way to plug in additional ORB behavior such as support for transactions and security, which may be defined on either the client or server side. One of the main differences in this release is that interceptors now have scope. See Chapter 12, “Interceptor and object wrapper interfaces and classes,” for more information on interceptor and object wrapper classes and interfaces.

## What's new in 4.1

---

The VisiBroker 4.1 features and enhancements include:

- **Abstract base support:** This feature provides CORBA 2.3.1-compliant support for VisiBroker 4.x and ensures that its CORBA 2.3.- compliant Interface Repository is compatible with VisiBroker 3.x clients.
- **Class downloading:** This feature allows the client to receive or find implementations of some classes that are not stored locally on the system. Depending on the security policy configured for your server, clients will be able to download classes from one source but will be denied the ability to download them from another source.
- **Redesigned Interface Repository:** This feature has been enhanced to make sure that it is compatible with the VisiBroker for Java 3.4 Interface Repository.
- **Connection timeout:** This feature allows you to indicate a timeout after which attempts to connect to an object using one of the available endpoints will be aborted.

## What's new in 4.5

---

New features and enhancements in Visibroker 4.5 include:

- **Bidirectional support:** This feature makes it possible for a server to asynchronously connect with a client. For more information about this feature, see the *VisiBroker for Java Programmer's Guide*. A number of policies have been added to support this feature. These are listed in "Using VisiBroker properties" on page B-1.
- **SyncScope support:** Support for CORBA 2.4-compliant SyncScope capabilities is now provided (see "SyncScopePolicy" on page 13-10).
- **Ability to create exclusive connections.** For more information see "ExclusiveConnectionPolicy" on page 13-9.
- This manual includes a new section that describes the vbjc tool ("vbjc" on page 2-10), a section describing generation of classpath ("Specifying the classpath" on page 2-10) and a section describing specification of the JVM ("Specifying the JVM" on page 2-11) by the vbj and vbjc launchers in Solaris and NT. The "Using VisiBroker properties" Appendix now describes a number of previously-undocumented properties.

# Manual conventions

---

This section identifies the VisiBroker for Java *Reference*'s typographical and platform conventions.

## Typographic conventions

---

This manual uses the following conventions:

Convention	Used for
<b>boldface</b>	Bold type indicates that syntax should be typed exactly as shown. For UNIX, used to indicate database names, file names, and similar terms.
<i>italics</i>	Italics indicates information that the user or application provides, such as variables in syntax diagrams. It is also used to introduce new terms.
<code>computer</code>	Computer typeface is used for sample command lines and code.
<b>bold computer</b>	In code samples, important statements appear in boldface.
UPPERCASE	Uppercase letters indicate Windows file names.
[ ]	Brackets indicate optional items.
...	An ellipsis indicates the continuation of previous lines of code or that the previous argument can be repeated.
	A vertical bar separates two mutually exclusive choices.

## Platform conventions

---

This manual uses the following conventions—where necessary—to indicate that information is platform-specific:

Windows	All Windows platforms including Windows 3.1, Windows NT, and Windows 95
WinNT	Windows NT only
Win95	Windows 95 only
Win98	Windows 98 only
Win2000	Windows 2000 only
UNIX	All UNIX platforms
Solaris	Solaris only
AIX	AIX only
HP-UX	HP-UX only
IRIX	IRIX only
Digital UNIX	Digital UNIX only

## Where to find additional information

---

For more information about VisiBroker for Java, refer to these information sources:

- VisiBroker for Java *Release Notes* contain late-breaking information about the current release of VisiBroker for Java.
- VisiBroker for Java *Installation Guide*. This guide contains the instructions for installing VisiBroker for Java on Windows and UNIX and information for system administrators who are configuring VisiBroker.
- VisiBroker for Java *Programmer's Guide* provides information on developing distributed object-based applications in Java for Windows and UNIX platforms.
- VisiBroker for Java *Visibroker Gatekeeper Guide*. This guide provides information about how to configure and use the Gatekeeper.
- The Inprise web site also provides a variety of useful information to developers and others who are interested in evaluating our products and ORB technology. From the web site you can view information specific to developers using the VisiBroker ORB. This includes a section listing Frequently Asked Questions (FAQ) and their answers. Visit the Inprise web site at <http://www.borland.com/visibroker/>.

For more information about the CORBA specification, refer to *The Common Object Request Broker: Architecture and Specification*. This document is available from the Object Management Group and describes the architectural details of CORBA. You can access the CORBA specification at the OMG web site: <http://www.omg.org/>.

## Contacting developer support

---

Inprise offers a variety of support options. These include free services on the Internet, where you can search our extensive information base and connect with other users of Inprise products. In addition, you can choose from several categories of telephone support, ranging from support on installation of the Inprise product to fee-based consultant-level support and detailed assistance.

For more information about Inprise's developer support services, please see our web site at <http://www.borland.com/devsupport/>, call Inprise Assist at 800-523-7070, or contact our Sales Department at 800-632-2864.

When you contact developer support, you will be asked to provide the following information:

- Your Access ID number
- Product name and version (for example, VisiBroker for Java, version 4.5)
- Operating system and version (for example, Windows NT Server 4.0 with Service Pack 5)

- Your desired priority (low, medium, high)
- Brief description of the problem
- Details of any error messages or exceptions raised



# Programmer tools

This chapter describes the programmer tools offered by VisiBroker for Java. For information about the syntax used with these tools, see Chapter 1, “Preface.”

## Options

---

All programmer’s tools have both general and specific options. The specific options for each tool are listed in the section for the tool. The general options are listed below.

### General options

---

The following options are common to all programmer tools:

Option	Description
<code>-J&lt;java_option&gt;</code>	Passes the <i>java_option</i> directly to the Java virtual machine.
<code>-VBJversion</code>	Prints the version.
<code>-VBJprop</code>	Passes a property to VBJ
<code>-VBJdebug</code>	Prints the debug information.

---

## idl2ir

---

This command allows you to populate an interface repository with objects defined in an Interface Definition Language (IDL) source file.

**Syntax**    `idl2ir [options] {idl filename}`

**Example**    `idl2ir -irep my_repository -replace java_examples/bank/Bank.idl`

**Description** The `idl2ir` command takes an IDL file as input, binds itself to an interface repository server and populates the repository with the IDL constructs contained in `idl filename`. If the repository already contains an item with the same name as an item in the IDL file, the old item will be modified.

**Options** The following options are available for `idl2ir`.

Option	Description
<code>-D, -define foo[=bar]</code>	Defines a preprocessor macro <code>foo</code> , optionally with a value <code>bar</code> .
<code>-I, -include &lt;dir&gt;</code>	Specifies an additional directory for <code>#include</code> searching.
<code>-P, -no_line_directives</code>	Suppresses the generation of line number information. The default is off.
<code>-H, -list_includes</code>	Prints the full paths of included files on the standard error output. The default is off.
<code>-C, -retain_comments</code>	Retains comments from IDL file when the Java code is generated. Otherwise, the comments will not appear in the Java code.
<code>-U, -undefine foo</code>	Undefines a preprocessor macro <code>foo</code> .
<code>-[no_]idl_stirct</code>	Specifies a strict OMG standard interpretation of IDL source. The default is off.
<code>-[no_]warn_unrecognized_pragmas</code>	Displays a warning that appears if a <code>#pragma</code> is not recognized. The default is on.
<code>-[no_]back_compat_mapping</code>	Specifies the use of mapping that is compatible with 3.x.
<code>-irep &lt;irep name&gt;</code>	Specifies the name of the interface repository.
<code>-deep</code>	Applies a deep (versus shallow) merge. The default is off.
<code>-replace</code>	Replaces entire repository instead of merging. The default is off.
<code>-h, -help, -usage, -?</code>	Prints help information.
<code>file1 [file2] ...</code>	Specifies the one or more files to be processed.

In addition, you can use any of the command line options for the `vbj` command as command line options for the `idl2ir` command. For a complete list of the `vbj` options, see “Options” on page 2-9.

## ir2idl

This command allows you to create an Interface Definition Language (IDL) source file with objects from an interface repository.

**Syntax** `ir2idl [options]`

**Example** `ir2idl -irep my_repository -o my_file`

**Description** The `ir2idl` command binds to the IR and prints the contents in IDL format.

**Options** The following options are available for `ir2idl`.

Option	Description
<code>-irep &lt;irep name&gt;</code>	Specifies the name of the interface repository.
<code>-o &lt;file&gt;</code>	Specifies the name of the output file, or “-” for stdout.
<code>-strict</code>	Specifies strict adherence to OMG-standard code generation. The default is on.
<code>-version</code>	Displays or prints out the version of VisiBroker for Java that you are currently running
<code>-h, -help, -usage, -?</code>	Prints help information.

## idl2java

This command generates Java source code from an IDL source file.

**Syntax** `idl2java [options] {filename.idl}`

**Example** `idl2java -no_tie Bank.idl`

**Description** The `idl2java` command, a Java-based preprocessor, compiles an IDL source file and creates a directory structure containing the Java mappings for the IDL declarations. Typically, one IDL file will be mapped to many java files because Java allows only one public interface or class per file. IDL file names must end with the `.idl` extension.

**Options** The following options are available for `idl2java`.

Option	Description
<code>-D, -define foo[=bar]</code>	Defines a preprocessor macro <code>foo</code> , optionally with a value <code>bar</code> .
<code>-I, -include &lt;dir&gt;</code>	Specifies the full or relative path to the directory for <code>#include</code> files. Used in searching for include files.
<code>-P, -no_line_directives</code>	Suppresses the generation of line number information in the generated code. The default is off.
<code>-H, -list_includes</code>	Prints the full paths of included files on the standard error output.
<code>-C, -retain_comments</code>	Retains comments from IDL file when the Java code is generated. Otherwise, the comments will not appear in the Java code. The default is off.
<code>-compilerflag</code>	Specifies the flags that can be set.
<code>-compiler options</code>	Specifies the option for the compiler.
<code>-U, -undefine foo</code>	Undefines a preprocessor macro <code>foo</code> .
<code>-[no_]idl_strict</code>	Specifies strict adherence to OMG standard interpretation of idl source. The default is off.

Option	Description
-[no_]warn_unrecognized_pragmas	Displays a warning that appears if a <code>#pragma</code> is not recognized. The default is on.
-[no_]back_compat_mapping	Specifies the use of IDL mapping that is compatible with 3.x caffeine compiles.
-[no_]boa	Specifies BOA-compatible code generation. The default is off.
-[no_]comments	Suppresses the generation of comments in the code. The default is on.
-[no_]examples	Suppresses the generation of the <code>_example</code> classes. The default is off.
-gen_included_files	Generates code for <code>#included</code> files. The default is off.
-list_files	Lists files written during code generation. The default is off.
-[no_]obj_wrapper	Generates support for object wrappers. The default is off.
-root_dir <i>&lt;path&gt;</i>	Specifies the directory in which the generated files reside.
-[no_]servant	Generates servant (server-side) code. The default is on.
-tie	Generates <code>_tie</code> classes. The default is on.
-[no_]warn_missing_define	Warns if any forward declared interfaces were not defined. The default is on.
-[no_]bind	Suppresses the generation of <code>bind()</code> methods in the generated Helper class. The default is off.
-[no_]compile	When set to on, automatically compiles the java files. The default is off.
-dynamic_marshall	Specifies that marshalling use DSI/DII model. The default is off.
-idl2package <i>&lt;IDL name&gt;</i> <i>&lt;pkg&gt;</i>	Overrides default package for a given IDL container type.
-[no_]invoke_handler	Generates invocation handler class for EJB. Default is on.
-[no_]narrow_compliance	Generated code for narrow is compliant. the default is on. When this options is set to off, generated code is backward compatible.
-[no_]Object_method	Overrides certain methods on <code>java.lang. Object</code> . The default is on.
-package <i>&lt;pkg&gt;</i>	Specifies the root package for generated code.
-stream_marshall	Specifies that marshaling use the stream model. The default is on.
-strict	Specifies strict adherence to OMG standard for code generation. The default is off.
-version	Displays the software version number of VisiBroker for Java.
-map_keyword <i>&lt;kw&gt;</i> <i>&lt;replacement&gt;</i>	Specifies the keyword to avoid and designates its replacement.
-h, -help, -usage, -?	Prints help information.
file1 [file2] ...	Specifies the one or more files to be processed.

In addition, you can use any of the command-line options for the `vbj` command as command line options for the `idl2java` command. For a complete list of the `vbj` options, see “Options” on page 2-9.

## java2idl

This command generates an IDL from a Java class file (in Java byte code). You can enter one or more Java classes (in byte codes). If you enter more than one class name, make sure you include spaces in between the class names.

If you use a class that extends `org.omg.CORBA.IDLEntity` in some Java remote interface definition, it must have the following:

- an IDL file that contains the IDL definition for that type because the `org.omg.CORBA.IDLEntity` interface is a signature interface that marks all IDL data types mapped to Java.
- all related (supporting) classes according to the CORBA 2.3 IDL2Java Specification from the Object Management Group (OMG).

If you use a class that extends `org.omg.CORBA.IDLEntity` in some Java remote interface definition, use the `-import <IDL files>` directive in the `java2idl` tool's command line.

For more information, refer to the CORBA 2.3 IDL2Java Specification located at <http://www.omg.org/>.

**Note** To use this command, you must have a virtual machine supporting JDK 1.1 or later.

**Syntax** `java2idl [options] {class}`

**Example** `java2idl Account Client Server`

**Description** Use this command when you want to generate an IDL from your Java byte code. You might want to use this when you have existing Java byte code and want to create an IDL file from it so it can be used with some other programming language like C++, COBOL, or Smalltalk.

**Options** The following options are available for `java2idl`.

Option	Description
<code>-D, -define foo[=bar]</code>	Defines a preprocessor macro <code>foo</code> , optionally with a value <code>bar</code> .
<code>-I, -include &lt;dir&gt;</code>	Specifies the full or relative path to the directory for <code>#include</code> files. Used in searching for include files.
<code>-P, -no_line_directives</code>	Suppresses the generation of line number information in the generated code. The default is off.
<code>-H, -list_includes</code>	Prints the full paths of included files on the standard error output.

Option	Description
-C, -retain_comments	Retains comments from IDL file when the Java code is generated. Otherwise, the comments will not appear in the Java code. The default is off.
-U, -undefine foo	Undefines a preprocessor macro <i>foo</i> .
-[no_]idl_strict	Specifies strict adherence to OMG standard interpretation of idl source. The default is off.
-[no_]warn_unrecognized_pragmas	Displays a warning that appears if a <i>#pragma</i> is not recognized. The default is on.
-[no_]back_compat_mapping	Specifies the use of mapping that is compatible with 3.x caffeine compile.
-exported <pkg>	Specifies the name of an exported package.
-[no_]export_all	Exports all packages. The default is off.
-import <IDL file name>	Loads extra IDL definitions.
-imported <pkg> <IDL file name>	Specifies the name of an imported package.
-o <file>	Specifies the name of an output file, or “-” for stdout.
-strict	Specifies strict adherence to OMG standard for code generation. The default is off.
-version	Displays the software version number of VisiBroker for Java.
-h, -help, -usage, -?	Prints help information.
class1 [class2] ...	Specifies the one or more Java classes to be processed.

In addition, you can use any of the command line options for the `vbj` command as command line options for the `java2idl` command. For a complete list of the `vbj` options, see “Options” on page 2-9.

## java2iiop

This command allows you to use the Java language to define IDL interfaces instead of using IDL. You can enter one or more Java class names (in Java byte code). If you enter more than one class name, make sure you include spaces in between the class names. Use fully scoped class names.

Note To use this command, you must have a virtual machine supporting JDK 1.1 or later.

If you use a class that extends `org.omg.CORBA.IDLEntity` in some Java remote interface definition, it must have the following:

- an IDL file that contains the IDL definition for that type because the `org.omg.CORBA.IDLEntity` interface is a signature interface that marks all IDL data types mapped to Java.
- all related (supporting) classes according to the CORBA 2.3 IDL2Java Specification from the Object Management Group (OMG).

If you use a class that extends `org.omg.CORBA.IDLEntity` in some Java remote interface definition, use the `-import <IDL files>` directive in the `java2iiop` tool's command line.

For more information, refer to the CORBA 2.3 IDL2Java Specification located at <http://www.omg.org/>.

**Syntax** `java2iiop [options] {class name}`

**Example** `java2iiop -no_tie Account Client Server`

**Description** Use `java2iiop` if you have existing Java byte code that you wish to adapt to use distributed objects or if you do not want to write IDL. By using `java2iiop`, you can generate the necessary container classes, client stubs, and server skeletons from Java byte code.

**Note** The `java2iiop` compiler does not support overloaded methods on CORBA interfaces.

**Options** The following options are available for `java2iiop`.

Option	Description
<code>-D, -define foo[=bar]</code>	Defines a preprocessor macro <code>foo</code> , optionally with a value <code>bar</code> .
<code>-I, -include &lt;dir&gt;</code>	Specifies the full or relative path to the directory for <code>#include</code> files. Used in searching for include files.
<code>-P, -no_line_directives</code>	Suppresses the generation of line number information in the generated code. The default is off.
<code>-H, -list_includes</code>	Prints the full paths of included files on the standard error output.
<code>-C, -retain_comments</code>	Retains comments from IDL file when the Java code is generated. Otherwise, the comments will not appear in the Java code. The default is off.
<code>-U, -undefine foo</code>	Undefines a preprocessor macro <code>foo</code> .
<code>-[no_]idl_strict</code>	Specifies strict adherence to OMG standard interpretation of idl source. The default is off.
<code>-[no_]warn_unrecognized_pragmas</code>	Displays a warning that appears if a <code>#pragma</code> is not recognized. The default is on.
<code>-[no_]back_compat_mapping</code>	Specifies the use of mapping that is compatible with 3.x.
<code>-exported &lt;pkg&gt;</code>	Specifies the name of an exported package.
<code>-[no_]export_all</code>	Exports all packages. The default is off.
<code>-import &lt;IDL file name&gt;</code>	Loads extra IDL definitions.
<code>-imported &lt;pkg&gt; &lt;IDL file name&gt;</code>	Specifies the name of an imported package.
<code>-[no_]boa</code>	Specifies BOA-compatible code generation. The default is off.
<code>-[no_]comments</code>	Suppresses the generation of comments in the code. The default is on.

Option	Description
-[no_]examples	Suppresses the generation of the <code>_example</code> classes. The default is off.
-gen_included_files	Generates code for <code>#included</code> files. The default is off.
-list_files	Lists files written during code generation. The default is off.
-[no_]obj_wrapper	Generates support for object wrappers. The default is off.
-root_dir <i>&lt;path&gt;</i>	Specifies the directory in which the generated files reside.
-[no_]servant	Generates servant (server-side) code. The default is on.
-tie	Generates <code>_tie</code> classes. The default is on.
-[no_]warn_missing_define	Warns if any forward declared file names were not defined. The default is on.
-[no_]bind	Suppresses the generation of <code>bind()</code> methods in the generated Helper class. The default is off.
-[no_]compile	Automatically generates java files. When set to on, also automatically compiles the java files. The default is off.
-compiler	Specifies the java compiler to be used. This option is ignored if the <code>-compile</code> option is not set.
-compiler flags	Specifies the java compiler flags to be passed to the java compiler. This option is ignored if the <code>-compile</code> option is not set.
-dynamic_marshall	Specifies that marshalling use DSI/DII model. The default is off.
-idl2package <i>&lt;IDL name&gt;</i> <i>&lt;pkg&gt;</i>	Overrides default package for a given IDL container type.
-[no_]invoke_handler	Generates invocation handler class for EJB. Default is on.
-[no_]narrow_compliance	Generated code is compliant. the default is on.
-[no_]Object_method	Generates all methods defined in <code>java.lang.Object</code> methods, such as <code>string</code> and <code>equals</code> . The default is on.
-package <i>&lt;pkg&gt;</i>	Specifies the root package for generated code.
-stream_marshall	Specifies that marshaling use the stream model. The default is on.
-strict	Specifies strict adherence to OMG standard for code generation. The default is off.
-version	Displays the software version number of VisiBroker for Java.
-map_keyword <i>&lt;kw&gt;</i> <i>&lt;replacement&gt;</i>	Specifies the keyword to avoid and designates its replacement.
-h, -help, -usage, -?	Prints help information.
class1 [class2] ...	Specifies the one or more Java classes to be processed.

In addition, you can use any of the command line options for the `vbj` command as command line options for the `java2iiop` command. For a complete list of the `vbj` options, see “Options” on page 2-9.



# vbj

This command starts the local Java interpreter.

**Syntax** `vbj [options] [arguments normally sent to java VM] {class} [arg1 arg2 ...]`

`{class}` Specifies the name of the class to be executed.

`[arg1 arg2 ...]` Specific arguments that are to be passed to the class.

**Example** `vbj Server`

**Description** Java applications have certain limitations not faced by applications written in other languages. The `vbj` command provides options to work around some of these limitations, and is the preferred method to launch VisiBroker for Java applications. The `vbj` command

- Checks environment variables and Windows registry settings.
- Can optionally locate an osagent for those Java virtual machines that lack UDP broadcast support.
- Automatically sets the CLASSPATH to work correctly with the VisiBroker for Java runtime.

In addition, the `vbj` command sets two ORB properties and passes them to the ORB runtime.

- `vbroker.agent.addr` is set to the value of the `OSAGENT_ADDR` environment variable or registry setting.
- `vbroker.agent.port` is set to the value of the `OSAGENT_PORT` environment variable or registry setting.

**Note** If you do not use the `vbj` command, you must explicitly set the values of `vbroker.agent.addr` and `vbroker.agent.port`.

**Options** The following options are available for `vbj`.

Option	Description
<code>-version</code>	Displays or prints out the version of VisiBroker for Java that you are currently running.
<code>-VBJprop name=value</code>	Passes the property name and value pair into the Java VM as a System Property by adding it as a <code>-D&lt;name&gt;[=&lt;value&gt;]</code> parameter to the executed "java."
<code>-VBJjavavm vmpath</code>	Specifies the Java vm to be used.
<b>Windows</b> <code>-VBJtag tag</code>	Specifies the tag to use for the parameters being passed to the Java VM. The default tag value is <code>-D</code> .
<b>Windows</b> <code>-VBJquoteSpaces 0</code>	Turns off the automatic placement of quotation marks around arguments that contain spaces.

	Option	Description
Windows	-VBJclassPath <i>classpath</i>	Specifies an explicit setting for classpath.
	-VBJdebug	Turns on debugging information.

Additional options that may be passed to this command are defined by the Java virtual machine that is installed on your system. For example, to view all of the options for the JavaSoft VM, enter `java` with no options.

```
prompt>java
```

A list of options available for the Java interpreter appears.

- Windows
- In Windows environments, you can only use the `vregedit` tool to set the following options in the registry:
- VBROKER\_ADM
  - OSAGENT\_PORT

## vbjc

This command is used to compile Java source code that may import VisiBroker classes. When called, it:

- 1: Locates the VisiBroker library path
- 2: Adds the VisiBroker-standard jar files into the CLASSPATH
- 3: Launches javac

**Syntax**    `vbjc [arguments normally passed to javac]`

**Example**    `vbjc Server.java`

## Specifying the classpath

Eight possible sources of information may affect the classpath:

- vbroker library files (`$VBROKERDIR/lib/*.jar`)
- JDK library files (`$JAVAHOME/lib/*.jar`, `$JAVAHOME/jre/lib/*.jar`, `classes.zip`)
- Environment variable `CLASSPATH` (`$CLASSPATH`)
- Command line argument `-VBJclasspath <path string>`
- Command line argument `-classpath <path string>`
- Command line argument `-Djava.class.path=<path string>`
- Command line argument `-Denv.class.path=<path string>`

- Command line argument `-VBJaddJar <jar file name>` (The jar file should reside in `vbroker/lib` directory.)

Usually, a subset of these is merged into one classpath before the JVM is started. The classpath is generated differently on different platforms.

The following sources are merged together in the following order:

- 1 The classpath specified in `-VBJclasspath`
- 2 The `$CLASSPATH` exported in the environment
- 3 Visibroker-standard jar files (determined based on where `vbj` was found)
- 4 jar files added through `VBJaddJar` and assumed to be located in the `<vbroker>/lib` directory.
- 5 The current directory

**Unix** The merged classpath is exported and is not passed as parameter to the JVM.

The other classpath sources are passed to the JVM untouched, along with other non-classpath parameters.

**Windows** The merged classpath is passed using `-Djava.class.path`. The other classpaths are treated as normal arguments and are passed without alteration.

**Note:** Tools defined by NT as SERVICES don't recognize `-classpath`. Some of these tools ignore this argument, but some stop on it and generate errors. (e.g. `nameserv`, `irep`)

## Specifying the JVM

---

**Unix** The default JVM is `java`. You can specify a different JVM by including

`-VBJjavavm <jvm name>`

as a command line argument. No matter which JVM is specified, its actual existence is checked and the program aborts if it fails to locate the JVM. If the JVM you specified does not exist, no attempt is made to locate the default one.

**NT** If you do not explicitly specify the JVM, VisiBroker for Java will see if there is path information set in the environment. If the `PATH` variable is set, it checks each directory in the `PATH` that contains `\bin` to see if a JVM dll can be found there. It aborts if `PATH` is set but no JVM is found.

If `PATH` is not set, VisiBroker for Java looks into the Windows registry for the location of the currently installed JVM.

It uses the following names for the dll: for JVM 1.2 `jvm.dll`; for JVM 1.1 it uses `javai.dll`.

You can specify a different JVM by including

`-VBJjavavm <jvm name>`

as a command line argument. The `<jvm name>` must include the full path to the application; *i.e.* it should be in the form

`*\*\*java*`

where `*` means empty or non-empty string. For instance, `C:\jdk\bin\java`. The program aborts with an error message if the `<jvm name>` is not in the proper format or if it does not find a file with the `<jvm name>`.

# Other tools

---

The other tools that get installed with are identified in the following table. These tools are used to deploy and run the clients and servers that you develop.

Tool	Description
osagent	The OSagent or Smart Agent is a service to which you register your client programs in order to find object implementations. A Smart Agent must be started on at least one host within your local network. See Chapter 16, “Using the Smart Agent,” in the VisiBroker for Java <i>Programmer’s Guide</i> for more information the Smart Agent options.
locserv	The Location Service enables you to find object instances based on particular attributes. It works with the Smart Agents to notify you of what objects are presently accessible on the network, and where they reside. See Chapter 17, “Using the Location Service,” in the VisiBroker for Java <i>Programmer’s Guide</i> for more information on the Location Service.
irep	The Interface Repository (irep) contains descriptions of your CORBA object interfaces but it is organized for runtime access by clients. You can view the information in the repository in a browser. See Chapter 2, “Programmer tools,” in the VisiBroker for Java <i>Programmer’s Guide</i> for more information on the Interface Repositories.
oad	The Object Activation Daemon (OAD) allows you to register objects that are to be started automatically when clients attempt to access them. See Chapter 20, “Using the Object Activation Daemon,” in the VisiBroker for Java <i>Programmer’s Guide</i> .
oadutil	The OAD utility provides you with commands that let you manually register, unregister, and list the object implementations available on your system from either command line or from within a script. See Chapter 20, “Using the Object Activation Daemon,” of the VisiBroker for Java <i>Programmer’s Guide</i> for more information on this tool and its options.

Tool	Description
osfind	Osfind is a command line tool that provides you with an overview of the processes that you are currently running.
gatekeeper	The Gatekeeper enables applets to communicate with object servers across networks, while still conforming to the security restrictions imposed by web browsers and firewalls. See the <i>VisiBroker for Java Visibroker Gatekeeper Guide</i> for more information on the Gatekeeper.

---



## IDL to Java mapping

This chapter describes the basics of VisiBroker for Java's current IDL-to-Java language mapping, as implemented by the `idl2java` compiler. VisiBroker for Java conforms with the *OMG IDL/Java Language Mapping Specification*. A copy of the *OMG IDL/Java Language Mapping Specification* is available from the Inprise web site at <http://www.borland.com/visibroker/>.

See the latest version of the *OMG IDL/Java Language Mapping Specification* for complete information and, especially, for information about the following:

- Mapping pseudo-objects to Java
- Server-side mapping
- Java ORB portability interfaces

### Names

---

In general, IDL names and identifiers are mapped to Java names and identifiers with no change.

If a name collision might be generated in the mapped Java code, the name collision is resolved by prepending an underscore (`_`) to the mapped name.

In addition, because of the nature of the Java language, a single IDL construct may be mapped to several (differently named) Java constructs. The "additional" names are constructed by appending a descriptive suffix. For example, the IDL interface `AccountManager` is mapped to the Java interface `AccountManager` and additional Java classes `AccountManagerOperations`, `AccountManagerHelper`, and `AccountManagerHolder`.

In those exceptional cases that the "additional" names could conflict with other mapped IDL names, the resolution rule described above is applied to the other mapped IDL names. In other words, the naming and use of required "additional" names takes precedence.

For example, an interface whose name is `fooHelper` or `fooHolder` is mapped to `_fooHelper` or `_fooHolder` respectively, regardless of whether an interface named `foo` exists. The helper and holder classes for interface `fooHelper` are named `_fooHelperHelper` and `_fooHelperHolder`.

IDL names that would normally be mapped unchanged to Java identifiers that conflict with Java reserved words will have the collision rule applied.

## Reserved names

---

The mapping reserves the use of several names for its own purposes. The use of any of these names for a user-defined IDL type or interface (assuming it is also a legal IDL name) will result in the mapped name having an underscore (`_`) prepended. Reserved names are as follows:

- The Java class `<type>Helper`, where `<type>` is the name of an IDL user-defined type.
- The Java class `<type>Holder`, where `<type>` is the name of an IDL user-defined type (with certain exceptions such as `typedef` aliases).
- The Java classes `<basicJavaType>Holder`, where `<basicJavaType>` is one of the Java primitive data types that is used by one of the IDL basic data types.
- The nested scope Java package name `<interface>Package`, where `<interface>` is the name of an IDL interface.
- The Java classes `<interface> Operations`, `<interfaces> POA`, and `<interface>POATie`, when `<interface>` is the name of an IDL interface type.

## Reserved words

---

The use of any of these words for a user-defined IDL type or interface (assuming it is also a legal IDL name) will result in the mapped name having an underscore (`_`) prepended. The reserved keywords in the Java language are as follows:

<code>abstract</code>	<code>abstractBase</code>	<code>boolean</code>	<code>break</code>
<code>byte</code>	<code>case</code>	<code>catch</code>	<code>char</code>
<code>class</code>	<code>const</code>	<code>continue</code>	<code>default</code>
<code>do</code>	<code>double</code>	<code>else</code>	<code>extends</code>
<code>false</code>	<code>final</code>	<code>finally</code>	<code>float</code>
<code>for</code>	<code>goto</code>	<code>if</code>	<code>implements</code>
<code>import</code>	<code>instanceof</code>	<code>int</code>	<code>interface</code>
<code>long</code>	<code>native</code>	<code>new</code>	<code>null</code>
<code>package</code>	<code>private</code>	<code>protected</code>	<code>public</code>



return	short	static	super
switch	synchronized	this	throw
throws	transient	true	try
void	volatile	while	

# Modules

---

An IDL module is mapped to a Java package with the same name. All IDL type declarations within the module are mapped to corresponding Java class or interface declarations within the generated package.

IDL declarations not enclosed in any modules are mapped into the (unnamed) Java global scope.

Code sample 3.1 shows the Java code generated for a type declared within an IDL module.

**Code sample 3.1** Mapping an IDL module to a Java package

```
/* From Example.idl: */
module Example { .... };

// Generated java

package Example;

...

```

# Basic types

---

The following table shows how the defined IDL types map to basic Java types.

**Table 3.1** Basic type mappings

IDL type	Java type
boolean	boolean
char	char
wchar	char
octet	byte
string	java.lang.String
wstring	java.lang.String
short	short
unsigned short	short
long	int
unsigned long	int
longlong	long
unsigned longlong	long

**Table 3.1** Basic type mappings (continued)

IDL type	Java type
float	float
double	double

When there is a potential mismatch between an IDL type and its mapped Java type, a standard CORBA exception can be raised. For the most part, exceptions are in two categories,

- Range of the Java type is larger than the IDL type. For example, Java chars are a superset of IDL chars.
- Because there is no support in Java for unsigned types, the developer is responsible for ensuring that large unsigned IDL type values are handled correctly as negative integers in Java.

Additional details are described in the following sections.

## IDL type extensions

This section summarizes VisiBroker’s support for IDL type extensions. The first table provides a summary for quick look-ups. This is followed by a table summarizing support for new types.

**Table 3.2** Summary of supported IDL extensions

Type	Supported in VisiBroker for Java?
longlong	yes
unsigned longlong	yes
long double	no <sup>1</sup>
wchar	yes <sup>2</sup>
wstring	yes <sup>2</sup>
fixed	no <sup>1</sup>

1. VisiBroker for Java will support in a future release when the OMG makes a decision about how to implement.
2. UNICODE is used “on the wire.”

**Table 3.3** IDL extensions for new types

New types	Description
longlong	64-bit signed 2’s complements integers
unsigned longlong	64-bit unsigned 2’s complements integers
long double	IEEE Standard 754-1985 double extended floating point
wchar	Wide characters
wstring	Wide strings
fixed	Fixed-point decimal arithmetic (31 significant digits)

## Holder classes

---

Holder classes support OUT and INOUT parameter passing modes and are available for all the basic IDL data types in the `org.omg.CORBA` package. Holder classes are generated for all named user-defined types except those defined by `typedefs`. For more information about Holder classes, see Chapter 4, “Generated interfaces and classes.”

For user-defined IDL types, the holder class name is constructed by appending `Holder` to the mapped Java name of the type.

For the basic IDL data types, the holder class name is the Java type name (with its initial letter capitalized) to which the data type is mapped with an appended `Holder`, for example, `IntHolder`.

Each holder class has a constructor from an instance, a default constructor, and has a public instance member, `value`, which is the typed value. The default constructor sets the value field to the default value for the type as defined by the Java language.

- `false` for boolean
- `null` for values
- `0` for numeric and `char` types
- `null` for strings
- `null` for object references

To support portable stubs and skeletons, Holder classes for user-defined types also implement the `org.omg.CORBA.portable.Streamable` interface.

The holder classes for the basic types are defined in Code sample 3.2. They are in the `org.omg.CORBA` package.

### Code sample 3.2 Holder classes

```
// Java
package org.omg.CORBA;

final public class ShortHolder implements Streamable {
    public short value;
    public ShortHolder() {}
    public ShortHolder(short initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class IntHolder implements Streamable {
    public int value;
    public IntHolder() {}
    public IntHolder(int initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}
```

```
final public class LongHolder implements Streamable {
    public long value;
    public LongHolder() {}
    public LongHolder(long initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class ByteHolder implements Streamable {
    public byte value;
    public ByteHolder() {}
    public ByteHolder(byte initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class FloatHolder implements Streamable {
    public float value;
    public FloatHolder() {}
    public FloatHolder(float initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class DoubleHolder implements Streamable {
    public double value;
    public DoubleHolder() {}
    public DoubleHolder(double initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class CharHolder implements Streamable {
    public char value;
    public CharHolder() {}
    public CharHolder(char initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class BooleanHolder implements Streamable {
    public boolean value;
    public BooleanHolder() {}
    public BooleanHolder(boolean initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}
final public class StringHolder implements Streamable {
```

```

    public java.lang.String value;
    public StringHolder() {}
    public StringHolder(java.lang.String initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class ObjectHolder implements Streamable {
    public org.omg.CORBA.Object value;
    public ObjectHolder() {}
    public ObjectHolder(org.omg.CORBA.Object initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class ValueBaseHolder implements Streamable {
    public java.io.Serializable value;
    public ValueBaseHolder() {}
    public ValueBaseHolder(java.io.Serializable initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class AnyHolder implements Streamable {
    public Any value;
    public AnyHolder() {}
    public AnyHolder(Any initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class TypeCodeHolder implements Streamable {
    public TypeCode value;
    public typeCodeHolder() {}
    public TypeCodeHolder(TypeCode initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

final public class PrincipalHolder implements Streamable {
    public Principal value;
    public PrincipalHolder() {}
    public PrincipalHolder(Principal initial) {
        value = initial;
    }
    ...//implementation of the streamable interface
}

```

The Holder class for a user-defined type <foo> is shown below.

**Code sample 3.3** Holder class for a user-defined type

```
// Java
final public class <foo>Holder
    implements org.omg.CORBA.portable.Streamable {
    public <foo> value;
    public <foo>Holder() {}
    public <foo>Holder(<foo> initial) {}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

**Java null**

The Java `null` may only be used to represent null CORBA object references and `valuetypes` (including recursive `valuetypes`). For example, a zero length string, rather than `null` must be used to represent the empty string. This is also true for arrays and any constructed type, except for `valuetypes`. If you attempt to pass a `null` for a structure, it will raise a `NullPointerException`.

**Boolean**

---

The IDL type `boolean` is mapped to the Java type `boolean`. The IDL constants `TRUE` and `FALSE` are mapped to the Java constants `true` and `false`.

**Char**

---

IDL characters are 8-bit quantities representing elements of a character set while Java characters are 16-bit unsigned quantities representing Unicode characters. To enforce type-safety, the Java CORBA runtime asserts range validity of all Java `chars` mapped from IDL `chars` when parameters are marshaled during method invocation. If the `char` falls outside the range defined by the character set, a `CORBA::DATA_CONVERSION` exception is thrown.

The IDL `wchar` maps to the Java `char` type.

**Octet**

---

The IDL type `octet`, an 8-bit quantity, is mapped to the Java type `byte`.

**String**

---

The IDL type `string`, both bounded and unbounded variants, is mapped to the Java type `java.lang.String`. Range checking for characters in the string as well as bounds checking of the string is done at marshal time.

## WString

---

The IDL type `wstring`, used to represent Unicode strings, is mapped to the Java type `java.lang.String`. Bounds checking of the string is done at marshal time.

## Integer types

---

IDL `short` and `unsigned short` map to Java type `short`. IDL `long` and `unsigned long` map to Java's `int`.

**Note** Because there is no support in Java for unsigned types, the developer is responsible for ensuring that negative integers in Java are handled correctly as large unsigned values.

## Floating point types

---

The IDL floating point types `float` and `double` map to a Java class containing the corresponding data type.

## Helper classes

---

All user-defined IDL types have an additional “helper” Java class with the suffix `Helper` appended to the type name generated. Several static methods needed to manipulate the type are supplied.

- `Any` insert and extract operations for the type
- Getting the repository id
- Getting the typecode
- Reading and writing the type from and to a stream

For any user-defined IDL type, `<typename>`, the following is the Java code generated for the type. The helper class for a mapped IDL interface has a `narrow` operation defined for them.

**Code sample 3.4** Helper class: Java code generated for user-defined type

```
// generated Java helper
public class <typename>Helper {
    public static void insert(org.omg.CORBA.Any a, <typename> t);
    public static <typename> extract(org.omg.CORBA.Any a);
    public static org.omg.CORBA.TypeCode type();
    public static String id();
    public static <typename> read( org.omg.CORBA.portable.InputStream istream);
    {...}

    public static void write(
        org.omg.CORBA.portable.OutputStream ostream, <typename> value)
    {...}
```

```
// only for interface helpers
public static <typename> narrow(org.omg.CORBA.Object obj);
```

### Code sample 3.5 Mapping of a named type to Java helper class

```
// IDL - named type
struct st {long f1, String f2};
// generated Java
public class stHelper {
    public static void insert(org.omg.CORBA.Any any,
        st s) {...}
    public static st extract(org.omg.CORBA.Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static st read(org.omg.CORBA.InputStream is) {...}
    public static void write(org.omg.CORBA.OutputStream os,
        st s) {...}
}
```

### Code sample 3.6 Mapping of a typedef sequence to Java helper class

```
// IDL - typedef sequence
typedef sequence <long> IntSeq;
// generated Java helper
public class IntSeqHelper {
    public static void insert(org.omg.CORBA.Any any,
        int[] seq);
    public static int[] extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode type(){...}
    public static String id(){...}
    public static int[] read(
        org.omg.CORBA.portable.InputStream is)
    {...}
    public static void write(
        org.omg.CORBA.portable.OutputStream os,
        int[] seq)
    {...}
}
```

## Constants

---

Constants are mapped depending upon the scope in which they appear.

### Constants within an interface

---

Constants declared within an IDL interface are mapped to `public static final` fields in the Java interface `Operations` class corresponding to the IDL interface.

### Code sample 3.7 Mapping an IDL constant within a module to a Java class

```
/* From Example.idl: */
module Example {
    interface Foo {
        const long aLongerOne = -321;
    }
}
```



```

    };
};
// Foo.java
package Example;
public interface Foo extends com.inprise.vbroker.CORBA.Object,
    Example.FooOperations,
    org.omg.CORBA.portable.IDLEntity {
}
// FooOperations.java
package Example;
public interface FooOperations {
    public final static int aLongerOne = (int)-321;
}

```

## Constants NOT within an interface

---

Constants declared within an IDL module are mapped to a public interface with the same name as the constant and containing a `public static final` field named `value`. This field holds the constant's value.

**Note** The Java compiler normally inlines the value when the class is used in other Java code.

**Code sample 3.8** Mapping an IDL constant within a module to a Java class

```

/* From Example.idl: */
module Example {
    const long aLongOne = -123;
};
// Generated java
package Example;
public interface aLongOne {
    public final static int value = (int) -123;
}

```

## Constructed types

---

IDL constructed types include `enum`, `struct`, `union`, `sequence`, and `array`. The types `sequence` and `array` are both mapped to the Java `array` type. The IDL constructed types `enum`, `struct`, and `union` are mapped to a Java class that implements the semantics of the IDL type. The Java class generated will have the same name as the original IDL type.

### Enum

---

An IDL `enum` is mapped to a Java `final` class with the same name as the enum type which declares a value method, two static data members per label, an integer conversion method, and a private constructor. An example follows.

**Code sample 3.9** IDL enum mapped to a Java final class

```
// Generated java

public final class <enum_name> {
    //one pair for each label in the enum
    public static final int _<label> = <value>;
    public static final <enum_name> <label> =
        new <enum_name>(_<label>);

    public int value() {...}

    //get enum with specified value
    public static <enum_name> from_int(int value);

    //constructor
    protected <enum_name>(int) {...}
}
```

One of the members is a `public static final` that has the same name as the IDL enum label. The other has an underscore (`_`) prepended and is intended to be used in switch statements.

The value method returns the integer value. Values are assigned sequentially starting with 0. If the enum has a label named `value`, there is no conflict with the `value()` method in Java.

There will be only one instance of an enum. Since there is only one instance, pointer equality tests will work correctly; that is, the default `java.lang.Object` implementation of `equals()` and `hash()` will automatically work correctly for an enumeration's singleton object.

The Java class for the enum has an additional method, `from_int()`, which returns the enum with the specified value.

The holder class for the enum is also generated. Its name is the enumeration's mapped Java classname with `Holder` appended to it as follows:

**Code sample 3.10** Holder class for the enum

```
public class <enum_name>Holder implements
    org.omg.CORBA.portable.Streamable {
    public <enum_name> value;
    public <enum_name>Holder() {}
    public <enum_name>Holder(<enum_name> initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

**Code sample 3.11** IDL mapped to Java for enum

```
// IDL
module Example {
    enum EnumType { first, second, third };
};
```

```
// generated Java
public final class EnumType
    implements org.omg.CORBA.portable.IDLEntity {
    public static final int _first = 0;
    public static final int _second = 1;
    public static final int _third = 2;

    public static final EnumType first = new EnumType(_first);
    public static final EnumType second = new EnumType(_second);
    public static final EnumType third = new EnumType(_third);

    protected EnumType (final int _vis_value) { . . . }

    public int value () { . . . }
    public static EnumType from_int (final int _vis_value) { . . . }
    public java.lang.String toString() { . . . }
}

public final class EnumTypeHolder
    implements org.omg.CORBA.portable.Streamable {
    public OtherExample.EnumType value;
    public EnumTypeHolder () { . . . }
    public EnumTypeHolder (final OtherExample.EnumType _vis_value) { . . . }
    public void _read (final org.omg.CORBA.portable.InputStream input) { . . . }
    public void _write (final org.omg.CORBA.portable.OutputStream output) { . . . }
    public org.omg.CORBA.TypeCode _type () { . . . }
    public boolean equals (java.lang.Object o) { . . . }
}
```

## Struct

---

An IDL struct is mapped to a final Java class with the same name that provides instance variables for the fields in IDL member ordering and a constructor for all values. A null constructor is also provided that allows the structure's fields to be initialized later. The `Holder` class for the struct is also generated. Its name is the struct's mapped Java classname with `Holder` appended to it as follows:

### Code sample 3.12 Holder class for a struct

```
final public class <class>Holder implements
    org.omg.CORBA.portable.Streamable {
    public <class> value;
    public <class>Holder() {}
    public <class>Holder(<class> initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

**Code sample 3.13** Mapping an IDL struct to Java

```

/* From Example.idl: */
module Example {
    struct StructType {
        long field1;
        string field2;
    };
};
// generated Java
public final class StructType
    implements org.omg.CORBA.portable.IDLEntity {
    public int field1;
    public java.lang.String field2;
    public StructType () { . . . }
    public StructType (final int field1,
        final java.lang.String field2) { . . . }
    public java.lang.String toString() { . . . }
    public boolean equals (java.lang.Object o) {...}

    public final class StructTypeHolder implements org.omg.CORBA.portable.Streamable {
        public Example.StructType value;
        public StructTypeHolder () { . . . }
        public StructTypeHolder (final Example.StructType _vis_value)
            { . . . }
        public void _read (final org.omg.CORBA.portable.InputStream input)
            { . . . }
        public void _write (final org.omg.CORBA.portable.OutputStream output)
            { . . . }
        public org.omg.CORBA.TypeCode _type () { . . . }
    }
}

```

## Union

---

An IDL union is given the same name as the final Java class and mapped to it; it provides the following:

- Default constructor
- Accessor method for the union's discriminator, named `discriminator()`
- Accessor method for each branch
- Modifier method for each branch
- Modifier method for each branch having more than one case label
- Default modifier method, if needed

If there is a name clash with the mapped union type name or any of the field names, the normal name conflict resolution rule is used: prepend an underscore for the discriminator.

The branch accessor and modifier methods are overloaded and named after the branch. Accessor methods shall raise the `CORBA::BAD_OPERATION` system exception if the expected branch has not been set.

If there is more than one case label corresponding to a branch, the simple modifier method for that branch sets the discriminant to the value of the first

case label. In addition, an extra modifier method which takes an explicit discriminator parameter is generated.

If the branch corresponds to the `default` case label, then the modifier method sets the discriminator to a value that does not match any other case labels.

It is illegal to specify a union with a default case label if the set of case labels completely covers the possible values for the discriminator. It is the responsibility of the Java code generator (for example, the IDL compiler, or other tool) to detect this situation and refuse to generate illegal code.

A default method `_default()` is created if there is no explicit default case label, and the set of case labels does not completely cover the possible values of the discriminator. It will set the value of the union to be an out-of-range value.

The holder class for the union is also generated. Its name is the union's mapped Java classname with `Holder` appended to it as follows:

**Code sample 3.14** Holder class for a union

```
final public class <union_class>Holder
    implements org.omg.CORBA.portable.Streamable {
    public <union_class> value;
    public <union_class>Holder() {}
    public <union_class>Holder(<union_class> initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

**Code sample 3.15** Mapping an IDL union to Java

```
/* From Example.idl: */
module Example {
    enum EnumType { first, second, third, fourth, fifth, sixth };
    union UnionType switch (EnumType) {
        case first: long win;
        case second: short place;
        case third:
        case fourth: octet show;
        default: boolean other;
    };
};
// Generated java

final public class UnionType {
    //constructor
    public UnionType() {...}
    //discriminator accessor
    public int discriminator() { ... }

    //win
    public int      win() { ... }
    public void     win(int value) { ... }
```

```

//place
public short place() { ... }
public void place(short value) { ... }

//show
public byte show() { ... }
public void show(byte value) { ... }
public void show(int discriminator, byte value) { ... }

//other
public boolean other() {...}
public void other(boolean value) { ... }
public java.lang.String to String () { . . .}
public boolean equals (java.lang.Object o) { . . .}
}

final public class UnionTypeHolder {
    implements org.omg.CORBA.portable.Streamable {
    public UnionType value;
    public UnionTypeHolder() {}
    public UnionTypeHolder(UnionType initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

```

## Sequence

---

An IDL sequence is mapped to a Java array with the same name. In the mapping, anywhere the sequence type is needed, an array of the mapped type of the sequence element is used.

The holder class for the sequence is also generated. Its name is the sequence's mapped Java classname with `Holder` appended to it as follows:

### Code sample 3.16 Holder class for a sequence

```

final public class <sequence_class>Holder {
    public <sequence_element_type>[] value;
    public <sequence_class>Holder() {};
    public <sequence_class>Holder(
        <sequence_element_type>[] initial) {...};
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

```

**Code sample 3.17** Mapping an IDL sequence to Java

```
// IDL
typedef sequence<long>UnboundedData;
typedef sequence<long, 42>BoundedData;
// generated Java
final public class UnboundedDataHolder
    implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public UnboundedDataHolder() {};
    public UnboundedDataHolder(final int[] initial) { . . . };
    public void _read(org.omg.CORBA.portable.InputStream i)
        { . . . }
    public void _write(org.omg.CORBA.portable.OutputStream o)
        { . . . }
    public org.omg.CORBA.TypeCode _type() { . . . }
}

final public class BoundedDataHolder
    implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public BoundedDataHolder() {};
    public BoundedDataHolder(final int[] initial) { . . . };
    public void _read(org.omg.CORBA.portable.InputStream i)
        { . . . }
    public void _write(org.omg.CORBA.portable.OutputStream o)
        { . . . }
    public org.omg.CORBA.TypeCode _type() { . . . }
}
```

## Array

---

An IDL array is mapped the same way as an IDL bounded sequence. In the mapping, anywhere the array type is needed, an array of the mapped type of array element is used. In Java, the natural Java subscripting operator is applied to the mapped array. The length of the array can be made available in Java, by bounding the array with an IDL constant, which will be mapped as per the rules for constants.

The holder class for the array is also generated. Its name is the array's mapped Java classname with *Holder* appended to it as follows:

**Code sample 3.18** Holder class for an array

```
final public class <array_class>Holder
    implements org.omg.CORBA.portable.Streamable {
    public <array_element_type>[] value;
    public <array_class>Holder() {}
    public <array_class>Holder(
        <array_element_type>[] initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
}
```

```

    public org.omg.CORBA.TypeCode _type() {...}
}

```

### Code sample 3.19 Mapping for an array

```

// IDL
const long ArrayBound = 42;
typedef long larray[ArrayBound];
// generated Java

final public class larrayHolder
    implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public larrayHolder() {}
    public larrayHolder(int[] initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

```

## Interfaces

---

IDL interfaces are mapped to the two following public Java interfaces:

- Operations interface, which contains only the operations and constants declared in the IDL interfaces.
- CORBA Object declaration that extends all base interface operations, this interface operation, and `org.omg.CORBA.Object`.

An additional “helper” Java class with the suffix `Helper` is appended to the interface name. The Java interface extends the mapped, base `org.omg.CORBA.Object` interface.

The Java interface contains the mapped operation signatures. Methods can be invoked on an object reference to this interface.

The helper class declares a static narrow method that allows an instance of `org.omg.CORBA.Object` to be narrowed to the object reference of a more specific type. The IDL exception `CORBA::BAD_PARAM` is thrown if the narrow fails because the object reference doesn’t support the request type. A different system exception is raised to indicate other kinds of errors. Trying to narrow a null will always succeed with a return value of null.

There are no special “nil” object references. Java `null` can be passed freely wherever an object reference is expected.

Attributes are mapped to a pair of Java accessor and modifier methods. These methods have the same name as the IDL attribute and are overloaded. There is no modifier method for IDL readonly attributes.

The holder class for the interface is also generated. Its name is the interface’s mapped Java classname with `Holder` appended to it as follows:



**Code sample 3.20** Holder class for an interface

```

final public class <interface_class>Holder
    implements org.omg.CORBA.portable.Streamable {
    public <interface_class> value;
    public <interface_class>Holder() {}
    public <interface_class>Holder(
        <interface_class> initial) {
        value = initial;
    }
    public void _read(org.omg.CORBA.portable.InputStream i)
    {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
    {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

```

**Code sample 3.21** Mapping an IDL interface to Java

```

/* From Example.idl: */
module Example {
    interface Foo {
        long method(in long arg) raises(AnException);
        attribute long assignable;
        readonly attribute long nonassignable;
    };
};
// Generated java
package Example;

public interface Foo extends com.inprise.vbroker.CORBA.Object,
    Example.FooOperations,
    org.omg.CORBA.portable.IDLEntity {
}

public interface FooOperations {
    public int method (int arg) throws Example.AnException;
    public int assignable ();
    public void assignable (int assignable);
    public int nonassignable ();
}

public final class FooHelper {
    // ... other standard helper methods
    public static Foo narrow(org.omg.CORBA.Object obj)
    { . . . }
    public static Example.Foo bind (org.omg.CORBA.ORB orb,
        java.lang.String name,
        java.lang.String host,
        com.inprise.vbroker.CORBA.BindOptions _options) { . . . }
    public static Example.Foo bind (org.omg.CORBA.ORB orb,
        java.lang.String fullPoaName, byte[] oid) { . . . }
    public static Example.Foo bind (org.omg.CORBA.ORB orb,
        java.lang.String fullPoaName, byte[] oid,
        java.lang.String host,
        com.inprise.vbroker.CORBA.BindOptions _options) { . . . }
    public Foo read (org.omg.CORBA.portable.InputStream in) { . . . }
    public void write (org.omg.CORBA.portable.OutputStream out, Foo foo) { . . . }
}

```

```
public Foo extract (org.omg.CORBA.Any any) { . . . }
public void insert (org.omg.CORBA.Any any, Foo foo) { . . . }
}

public final class FooHolder
    implements org.omg.CORBA.portable.Streamable {
    public Foo value;
    public FooHolder() {}
    public FooHolder(final Foo initial) { . . . }
    public void _read(org.omg.CORBA.portable.InputStream i)
        { . . . }
    public void _write(org.omg.CORBA.portable.OutputStream o)
        { . . . }
    public org.omg.CORBA.TypeCode_type() { . . . }
}
```

## Passing parameters

---

IDL `in` parameters are mapped to normal Java actual parameters. The results of IDL operations are returned as the result of the corresponding Java method.

IDL `out` and `inout` parameters cannot be mapped directly into the Java parameter passing mechanism. This mapping defines additional holder classes for all the IDL basic and user-defined types which are used to implement these parameter modes in Java. The client supplies an instance of the appropriate holder Java class that is passed (by value) for each IDL `out` or `inout` parameter. The contents of the holder instance (but not the instance itself) are modified by the invocation, and the client uses the (possibly) changed contents after the invocation returns.

### Code sample 3.22 IN parameter mapping to Java actual parameters

```
/* From Example.idl: */
module Example {
    interface Modes {
        long operation(in long inArg, out long outArg, inout long inoutArg);
    };
};

// Generated Java:
package Example;
public interface Modes extends com.inprise.vbroker.CORBA.Object,
    Example.ModesOperations,
    org.omg.CORBA.portable.IDLEntity {
}
public interface ModesOperations {
    public int operation (int inArg,
        org.omg.CORBA.IntHolder outArg,
        org.omg.CORBA.IntHolder inoutArg);
}
```

In the above, the result comes back as an ordinary result and the actual `in` parameters only an ordinary value. But for the `out` and `inout` parameters, an

appropriate holder must be constructed. A typical use case might look as follows:

**Code sample 3.23** Holder for out and inout parameters

```
// user Java code
// select a target object
Example.Modes target = ...;
// get the in actual value
int inArg = 57;
// prepare to receive out
IntHolder outHolder = new IntHolder();
// set up the in side of the inout
IntHolder inoutHolder = new IntHolder(131);
// make the invocation
int result =target.operation(inArg, outHolder, inoutHolder);
// use the value of the outHolder
... outHolder.value ...
// use the value of the inoutHolder
... inoutHolder.value ...
```

Before the invocation, the input value of the inout parameter must be set in the holder instance that will be the actual parameter. The inout holder can be filled in either by constructing a new holder from a value, or by assigning to the value of an existing holder of the appropriate type. After the invocation, the client uses the `outHolder.value` to access the value of the out parameter, and the `inoutHolder.value` to access the output value of the inout parameter. The return result of the IDL operation is available as the result of the invocation.

## Server implementation with inheritance

---

Using inheritance is the simplest way to implement a server because server objects and object references look the same, behave the same, and can be used in exactly the same contexts. If a server object happens to be in the same process as its client, method invocations are an ordinary Java function call with no transport, indirection, or delegation of any kind.

Each IDL interface is mapped to a Java POA abstract class that implements the Java version of the IDL interface.

**Note** The POA class does not “truly” extend the IDL interface, meaning that POA is not a CORBA object. It is a CORBA servant and it can be used to create a “true” CORBA object. See Chapter 5, “Core interfaces and classes” for more information on the POA class.

User-defined server classes are then linked to the ORB by extending the `<interface>POA` class, as shown in Code sample 3.24.

**Note** The POA class itself is abstract and cannot be instantiated. To instantiate it, your implementation must implement its declared IDL interface operations.

**Code sample 3.24** Server implementation in Java using inheritance

```
/* From Bank.idl: */
module Bank {
```

```

        interface Account {
        };
    };

    // Generated java
    package Bank;
    public abstract class AccountPOA extends org.omg.PortableServer.Servant implements
        org.omg.CORBA.portable.InvokeHandler,
        Bank.AccountOperations { . . . }
    // Linking an implementation to the ORB :
    public class AccountImpl extends Bank.AccountPOA { . . . }

```

## Server implementation with delegation

---

The use of inheritance to implement a server has one drawback: since the server class extends the POA skeleton class, it cannot use implementation inheritance for other purposes because Java only supports single inheritance. If the server class needs to use the sole inheritance link available for another purpose, the delegation approach must be used.

When server classes are implemented using delegation some extra code is generated.

- Each interface is mapped to a **Tie** class that extends the POA skeleton and provides the delegation code.
- Each interface is also mapped to an **Operations** interface that is used to defined the type of object the **Tie** class is delegating.

The delegated implementation must implement the **Operation's** interface and has to be stored in a **Tie** class instance. Storing the instance of the **Operation** interface in the **Tie** object is done through a constructor provided by the **Tie** class. Code sample 3.25 shows an example of how delegation is used.

### Code sample 3.25 Server implementation in Java using delegation

```

/* From Bank.idl: */
module Bank {
    interface AccountManager {
        Account open(in string name);
    };
};

// Generated java
package Bank;
public interface AccountManagerOperations {
    public Example.Account open(java.lang.String name);
}

// Generated java
package Bank;
public class AccountManagerPOATie extends AccountManagerPOA {
    public AccountManagerPOATie (final Bank.AccountManagerOperations _delegate)
    { . . . }
    public AccountManagerPOATie (final Bank.AccountManagerOperations _delegate,
        final org.omg.PortableServer.POA _poa) { . . . }
    public Bank.AccountManagerOperations _delegate () { . . . }
}

```

```

    public void _delegate (final Bank.AccountManagerOperations delegate) { . . . }
    public org.omg.PortableServer.POA _default_POA () { . . . }
    public float open () { . . . }
}
// Linking an implementation to the ORB :
class AccountImpl implements AccountManager Operations
public class Server {
    public static main(String args) {
        // ...
        AccountManagerPOATie managerServant = new AccountManagerPOATie(new
AccountManagerImpl());
        // ...
    }
}

```

## Interface scope

---

OMG IDL to Java mapping specification does not allow declarations to be nested within an interface scope, nor does it allow packages and interfaces to have the same name. Accordingly, interface scope is mapped to a package with the same name with a “Package” suffix.

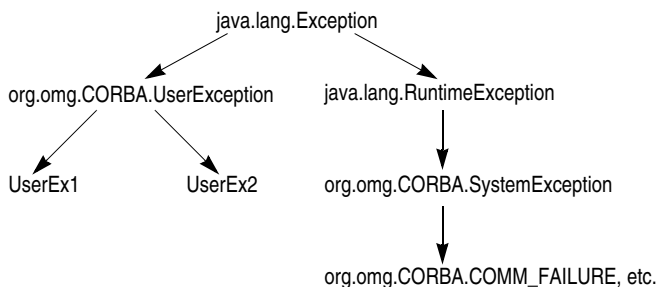
## Mapping for exceptions

---

IDL exceptions are mapped very similarly to structs. They are mapped to a Java class that provides instance variables for the fields of the exception and constructors.

CORBA system exceptions are unchecked exceptions. They inherit (indirectly) from `java.lang.RuntimeException`.

User defined exceptions are checked exceptions. They inherit (indirectly) from `java.lang.Exception`.



## User-defined exceptions

---

User-defined exceptions are mapped to final Java classes that extend `org.omg.CORBA.UserException` and are otherwise mapped just like the IDL struct type, including the generation of Helper and Holder classes.

If the exception is defined within a nested IDL scope (essentially within an interface) then its Java class name is defined within a special scope. Otherwise its Java class name is defined within the scope of the Java package that corresponds to the exception's enclosing IDL module.

**Code sample 3.26** Mapping user-defined exceptions

```
// IDL
module Example {
    exception AnException {
        string reason;
    };
};

// Generated Java
package Example;

public final class AnException extends org.omg.CORBA.UserException {
    public java.lang.String extra;
    public AnException () { . . . }
    public AnException (java.lang.String extra) { . . . }
    public AnException (java.lang.String _reason, java.lang.String extra) { . . . }
    public java.lang.String to String () { . . . }
    public boolean equals (java.lang.Object o) { . . . }
}

public final class AnExceptionHandler implements
    org.omg.CORBA.portable.Streamable {
    public Example.AnException value;
    public AnExceptionHandler () { }
    public AnExceptionHandler (final Example.AnException _vis_value) { . . . }
    public void _read (final org.omg.CORBA.portable.InputStream input) { . . . }
    public void _write (final org.omg.CORBA.portable.OutputStream output) { . . . }
    public org.omg.CORBA.TypeCode _type () { . . . }
}
```

## System exceptions

---

The standard IDL system exceptions are mapped to final Java classes that extend `org.omg.CORBA.SystemException` and provide access to the IDL major and minor exception code, as well as a string describing the reason for the exception. There are no public constructors for `org.omg.CORBA.SystemException`; only classes that extend it can be instantiated.

The Java class name for each standard IDL exception is the same as its IDL name and is declared to be in the `org.omg.CORBA` package. The default constructor supplies 0 for the minor code, `COMPLETED_NO` for the completion code, and the empty string ("" ) for the reason string. There is also a constructor which takes the reason and uses defaults for the other fields, as well as one which requires all three parameters to be specified.

## Mapping for the Any type

---

The IDL type `Any` maps to the Java class `org.omg.CORBA.Any`. This class has all the necessary methods to insert and extract instances of predefined types. If the extraction operations have a mismatched type, the `CORBA::BAD_OPERATION` exception is thrown.

In addition, insert and extract methods which take a holder class are defined to provide a high speed interface for use by portable stubs and skeletons. There is an insert and extract method defined for each primitive IDL type as well as a pair for a generic streamable to handle the case of non-primitive IDL types. For more information about the insert and extract methods, see “Any Insertion methods” on page 6-3 and “Any extraction methods” on page 6-2.

The insert operations set the specified value and reset the `Any`’s type if necessary.

Setting the typecode via the `type()` accessor wipes out the value. An attempt to extract before the value is set will result in a `CORBA::BAD_OPERATION` exception being raised. This operation is provided primarily so that the type may be set properly for IDL out parameters.

## Mapping for certain nested types

---

IDL allows type declarations nested within interfaces. Java does not allow classes to be nested within interfaces. Hence those IDL types that map to Java classes and that are declared within the scope of an interface must appear in a special “scope” package when mapped to Java.

IDL interfaces that contain these type declarations generate a scope package to contain the mapped Java class declarations. The scope package name is constructed by appending `Package` to the IDL type name.

**Code sample 3.27** Mapping for certain nested types

```
// IDL
module Example {
    interface Foo {
        exception e1 {};
    };
}

// generated Java
package Example.FooPackage;
final public class e1 extends org.omg.CORBA.UserException {...}
```

## Mapping for Typedef

---

Java does not have a typedef construct.

## Simple IDL types

---

IDL types that are mapped to simple Java types may not be subclassed in Java. Hence any typedefs that are type declarations for simple types are mapped to the original (mapped type) everywhere the typedef type appears. For simple types, `Helper` classes are generated for all typedefs.

## Complex IDL types

---

Typedefs for non arrays and sequences are “unwound” to their original type until a simple IDL type or user-defined IDL type (of the non typedef variety) is encountered.

Holder classes are generated for sequence and array typedefs.

**Code sample 3.28** Mapping a complex idl typedef

```
// IDL
struct EmpName {
    string firstName;
    string lastName;
};
typedef EmpName EmpRec;
// generated Java
    // regular struct mapping for EmpName
    // regular helper class mapping for EmpRec

final public class EmpName {
    ...
}

public class EmpRecHelper {
    ...
}
```



# Generated interfaces and classes

This chapter describes the classes that can be generated by the `idl2java` compiler.

## Overview

---

The Helper and Holder classes are provided for most the classes in the `org.omg.CORBA` package. They are also generated by the `idl2java` compiler for user-defined types and are given the name of the class that is generated for the type with an additional `Holder` or `Helper` suffix. Given a user-defined type named `MyType`, the `idl2java` compiler will generate the following:

- `public class MyType`
- `public class MyTypeOperations`
- `public class MyTypeHelper`
- `public class MyTypeHolder`
- `public class MyTypePOA`
- `public class MyTypePOATie`
- `public class _MyTypeStub`

## Signature and operations classes

---

These two classes, `MyType` and `MyTypeOperations`, provide the complete signature of your IDL interface when mapped to Java.

### Signature class

This class defines the signature interface for each interface you declare in your IDL file.

## Operations class

This class defines all of the methods that must be implemented by the object implementation. This class acts as the delegate object for the associated tie class when the tie mechanism is used.

## Ancillary classes

---

For each user-defined type a Helper and a Holder class are generated.

### Helper class

An abstract `Helper` class is generated by the `idl2java` compiler and contains the utility methods for operating on the associated object. The motivation for the `Helper` class is to avoid loading the methods that the class offers if they are not needed.

For objects like mapped structures, enumerations, unions, exceptions, valuetypes, and valueboxes, the `Helper` class provides methods for reading and writing the object to a stream and returning the object's repository identifier. The `Helper` classes generated for interfaces contain additional methods, like `bind` and `narrow`.

### Holder class

Since the Java language only allows parameters to be passed by value and not by reference, `Holder` classes are used to support the passing of `out` and `inout` parameters associated with operation requests. The interface of the `Holder` class is consistent for all types.

## Portability stub and skeleton interfaces

---

The IDL to Java language mapping defines a stub class that may be used for both the local and the remote invocation of objects. Skeleton classes may be either stream-based or DSI-based.

### Stub class

The stub class provides stub implementation for `<interface_name>` which the client calls. All stubs inherit from `org.omg.CORBA.portable.ObjectImpl`.

### POA class

Stream-based skeletons extend `org.omg.PortableServer.Servant` and DSI-based ones extend `org.omg.PortableServer.DynamicImpementation`. This abstract class provides the interface to the server-side implementation of your class.

### POATie class

The `POATie` class extends the `<interface>POA` class and allows you to delegate calls to any of the `<interface>Operations` methods to another object which implements the same methods. You do not have to use the `POATie` class if you

do not want, but instead you may have your <interface>Impl class directly extend the <interface>POA abstract class.

## <interface\_name>Operations

---

abstract public interface <interface\_name>Operations

An Operations class is generated by the idl2java compiler and contains the interface definitions of the methods and constants declared for <interface\_name> in the IDL file.

## <type\_name>Helper

---

abstract public class <type\_name>Helper

A Helper class is provided for most classes in the org.omg.CORBA package. Helper classes are also generated by the idl2java compiler for all user-defined types. The suffix Helper is added to the class name that is generated for the type. A variety of static methods are provided to manipulate the class.

### Methods for all Helper classes

---

public static <interface\_name> **extract**(org.omg.CORBA.Any any)

This method extracts the type from the specified Any object.

Parameter	Description
any	The Any object to contain the object.

public static String **id**()

This method gets the repository id for this object.

public static void **insert**(org.omg.CORBA.Any any, <type\_name> value)

This method insert a type into the specified Any object.

Parameter	Description
any	The Any object to contain the type.
value	The type to insert.

public static <type\_name> **read**(org.omg.CORBA.portable.InputStream input)

This method reads a type from the specified input stream.

Parameter	Description
input	The input stream from which the object is read.

```
public static org.omg.CORBA.TypeCode type()
```

This method returns the `TypeCode` associated with this object. For a list of possible return values see “`TCKind`” on page 6-30.

```
public static void write(org.omg.CORBA.portable.OutputStream output, <type_name> value)
```

This method writes a type to the specified output stream.

Parameter	Description
output	The output stream to which the object is written.
value	The type to be written to the output stream.

## Methods generated for interfaces

```
public static <interface_name> bind(org.omg.CORBA.ORB orb)
```

This method attempts to bind to any instance of an object of type `<interface_name>`.

```
public static <interface_name> bind(org.omg.CORBA.ORB orb, String name)
```

This method attempts to bind to an object of type `<interface_name>` that has the specified instance name.

Parameter	Description
name	The instance name of the desired object.

```
public static <interface_name> bind(org.omg.CORBA.ORB orb, String name, String host)
```

This method attempts to bind to an object of type `<interface_name>` that has the specified instance name and which is located on the specified host.

Parameter	Description
name	The instance name of the desired object.
host	The host name where the desired object is located.

```
public static <interface_name> bind(org.omg.CORBA.ORB orb, String name, String host,  
org.omg.CORBA.BindOptions options)
```

This method attempts to bind to an object of type `<interface_name>` that has the specified instance name and which is located on the specified host, using the specified `BindOptions`. `BindOptions` are described in “public class `BindOptions`” on page 5-1.

Parameter	Description
name	The instance name of the desired object.
host	The optional host name where the desired object is located.
options	The bind options for this object.

```
public static <interface_name> narrow(org.omg.CORBA.Object object)
```

This method attempts to narrow a `org.omg.CORBA.Object` reference to an object of type `<interface_name>`. If the object reference cannot be narrowed, a `null` value is returned.

Parameter	Description
object	The object to be narrowed to the type <code>&lt;interface_name&gt;</code> .

## Methods generated for object wrappers

The following methods are generated for helper classes when you invoke the `idl2java` command with the `-obj_wrapper` option. For complete details on using the object wrapper feature, see the VisiBroker for Java *Programmer's Guide*.

```
public static void addClientObjectWrapperClass(org.omg.CORBA.ORB orb, java.lang.Class c)
```

Adds a typed object wrapper from a client application. If more than one typed object wrapper is installed, they are added in the order in which they were registered.

Note This method should only be invoked by a client application.

Parameter	Description
c	The object wrapper that you want to add.
orb	The ORB the client wishes to use, obtained by the invoking the <code>ORB.init</code> method.

```
public static void addServerObjectWrapperClass(org.omg.CORBA.ORB orb, java.lang.Class c)
```

Adds a typed object wrapper from a server application. If more than one typed object wrapper is installed, they are added in the order in which they were registered.

Note This method should only be invoked by a server application.

Parameter	Description
c	The object wrapper that you want to add.
orb	The ORB the server wishes to use, obtained by the invoking the <code>ORB.init</code> method.

```
public static void removeClientObjectWrapperClass(org.omg.CORBA.ORB orb, java.lang.Class c)
```

Removes a typed object wrapper from a client application.

Note        This method should only be invoked by a client application.

Parameter	Description
c	The object wrapper that you want to remove.
orb	The ORB the client wishes to use, obtained by the invoking the ORB.init method.

public static void **removeServerObjectWrapperClass**(org.omg.CORBA.ORB orb, java.lang.Class c)  
Removes a typed object wrapper from a server application.

Note        This method should only be invoked by a server application.

Parameter	Description
c	The object wrapper class that you want to remove.
orb	The ORB the server wishes to use, obtained by the invoking the ORB.init method.

## <type\_name>Holder

public final class <interface\_name>Holder

A Holder class is provided for all basic IDL types in the org.omg.CORBA package. Holder classes are also generated by the idl2java compiler for all user-defined types. The suffix Holder is added to the class name that is generated for user-defined types. Each Holder has a set of constructors and a value member, which is the typed value.

The holder classes for the basic types are defined below. They are in the org.omg.CORBA package.

- public class ShortHolder
- public class IntHolder
- public class LongHolder
- public class ByteHolder
- public class FloatHolder
- public class DoubleHolder
- public class CharHolder
- public class BooleanHolder
- public class StringHolder
- public class ObjectHolder
- public class AnyHolder
- public class TypeCodeHolder
- public class PrincipalHolder

The Holder class for a user-defined type <type\_name> follows.

**Code sample 4.1    Holder class**

```
// Java
final public class <type_name>Holder
    implements org.omg.CORBA.portable.Streamable {
```

```

public <type_name> value;
public <type_name>Holder() {}
public <type_name>Holder(<type_name> initial) {}
public void _read(org.omg.CORBA.portable.InputStream i)
    {...};
public void _write(org.omg.CORBA.portable.OutputStream o)
    {...};
public org.omg.CORBA.TypeCode _type() {...}
}

```

## Member data

---

public <type\_name> **value**

This value represents the type contained by this object.

## Methods

---

public <type\_name>**Holder**()

This default constructor is useful for `out` parameters. The default constructor sets the value field to the default value for the type as defined by the Java language. The value is set to `false` for boolean types, 0 for integral and `char` types, `null` for strings, and `null` for object references.

public <type\_name>**Holder**(<interface\_name> **initial**)

The value constructor is useful for `inout` parameters. The value field is copied from the value field of the specified `Any` object.

Parameter	Description
initial	The other object the Holder is containing.

## \_\_<interface\_name>Stub

---

abstract public class \_\_<interface\_name>**Stub**

A stub class is generated by the `idl2java` compiler to provide a stub implementation for <interface\_name> which the client calls. This class provides the implementation for transparently acting on an object implementation.

## <interface\_name>POA

---

abstract public class <interface\_name>**POA**

This class provides the POA skeleton (servant) class for your interface. It does not provide any implementation for the <interface>`Operations` interface. You extend this class when implementing your <interface>`Impl` class.

`<interface_name>POATie`

## `<interface_name>POATie`

---

abstract public class `<interface_name>POATie`

A tie class is generated by the `idl2java` compiler for creating a delegator class for `<interface_name>POA`.

### Methods

---

public `<interface>POATie` (final `<interface>Operations _delegate`)

This method is the constructor. You pass it an object which implements the `<interface>Operations` interface and which will be used as the delegate object.

public `<interface>POATie` (final `<interface>Operations _delegate`,  
final `org.omg.PortableServer.POA _poa`)

This method is the constructor which initializes the delegate object and the default POA servant.



# Core interfaces and classes

This chapter describes the core interfaces and classes.

## BindOptions

---

Note Deprecated in VisiBroker 4.x

public class **BindOptions**

The `BindOptions` class represents the options to be used when a client is binding to a server. These parameters are passed to the `bind` method provided by an interface's `Helper` class.

The `defer_bind` parameter, when set to `true`, allows the client to delay the establishment of a connect to the object implementation until the first operation request is issued for that object. If set to `false`, the connection establishment is attempted immediately.

The `enable_rebind` parameter determines if any attempt is to be made to reconnect to an object implementation if the connection is unexpectedly broken due to a network failure or other error. If set to `true`, an attempt is to be made to locate and bind to another server offering an object of the same type and name as the original.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they provide.

### IDL definition

---

```
struct BindOptions {  
    boolean defer_bind;  
    boolean enable_rebind;  
};
```

## BindOptions constructors

---

public **BindOptions**(boolean **defer\_bind**, boolean **enable\_rebind**)

This method creates a `BindOptions` object, setting the `defer_bind` and `enable_rebind` properties as specified.

Parameter	Description
<code>defer_bind</code>	If set to <code>true</code> , connection establishment is delayed until the first operation request is invoked on the object. The default value is <code>false</code> .
<code>enable_rebind</code>	If set to <code>true</code> and the connection is broken to the server, the client attempts to bind to another server. The default value is <code>true</code> .

---

## BOA

---

Note    Deprecated in VisiBroker 4.x

public abstract class **BOA** extends `Object`

The Basic Object Adapter is used by object implementations to activate and deactivate the objects they offer to clients. An object implementation invokes the `obj_is_ready` method to make the implementation visible to clients on the network. The `deactivate_obj` method is used to make an object implementation unavailable to clients.

The BOA also provides methods for obtaining the `Principal` associated with a client request and entering an event loop to wait for the receipt of client requests.

An instance of a BOA is obtained by using the `ORB.BOA_init()` method. For example,

**Code sample 5.1**    Using `BOA_init()` with no arguments

```
...
// Initialize the ORB.
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
// Initialize the BOA.
com.inprise.vbroker.CORBA.BOA boa =
    ((com.inprise.vbroker.orb.CORBA.ORB)orb).BOA_init();
// Create the account manager object.
Bank.AccountManager manager = new AccountManagerImpl("BankManager");
// Export the newly created object.
boa.obj_is_ready(manager);
System.out.println(manager + " is ready.");
// Wait for incoming requests
boa.impl_is_ready();
...
```

The method shown above, `orb.BOA_init()`, initializes a BOA and returns a reference to the BOA. If you use this method, rather than `BOA_init (String boaType, java.util.Properties properties)`, the thread policy will be `TPool`. The

`BOA_init` (String `boaType`, java.util.Properties `properties`) method initializes a particular type of BOA with optional properties. It returns the adapter corresponding to the `boaType`. For an example of usage, see Code sample 5.2. For more information about these methods, see “public static `org.omg.CORBA.BOA BOA_init()`” on page 5-23.

**Code sample 5.2** Using `BOA_init()` with arguments to set thread policy and properties

```
...
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    java.util.Properties props = new java.util.Properties();
props.put("OAtreadMax", "40");
org.omg.CORBA.BOA boa = orb.BOA_init("TPool", props);
// create a new implementation object
    Bank.AccountManager manager =
    new AccountManagerImpl("BankManager");
    // make the implementation Net visible
    boa.obj_is_ready(manager);
    // wait for incoming requests...
    boa.impl_is_ready();
    ...
}
```

## IDL definition

---

```
interface BOA {
    void obj_is_ready(in CORBA::Object object);
    void obj_is_ready(
        in CORBA::Object object
        in string service_name
        in byte ref_data[]);
    void deactivate_obj(in CORBA::Object object);
    void impl_is_ready();
    void impl_is_ready(
        in string service_name,
        in Activator activator);
    void impl_is_ready(
        in string service_name,
        in Activator activator,
        in boolean block);
    CORBA::Principal get_principal(in CORBA::Object object);
};
```

## BOA methods

---

public void **deactivate\_obj**(Object **object**)

This method makes a server’s implementation object *invisible* to the network. After invoking this method, requests received for this object will cause a `CORBA.OBJECT_NOT_EXIST` exception to be raised.

Parameter	Description
object	The server’s implementation object to be deactivated.

public Principal **get\_principal**(Object **object**)

This method returns the `Principal` object associated with the current operation request or null if the specified object is not the target of the current invocation.

Parameter	Description
object	The server object on which the current operation request is being called. Normally, this object reference is to this object.

public void **impl\_is\_ready**()

This method is typically called by a server after all the objects that it implements have been activated using `obj_is_ready` method. This method enters a loop waiting for client operation requests to arrive and does not return. Calling this method is not required if the server has some other non-daemon thread running. For example, if there is a GUI running, the GUI will control the lifetime of the program and this method need not be used.

public abstract void **impl\_is\_ready**(java.lang.String **service\_name**, Activator **activator**)

This method registers an Activator and associates a service name with it.

Parameter	Description
service_name	The service name that the object implementation is associated with.
activator	The activator that is controlling the object’s activation.

public abstract void **impl\_is\_ready**(java.lang.String **service\_name**, Activator **activator**, boolean **block**)

This method registers an Activator and associates a service name with it.

Parameter	Description
service_name	The service name that the object implementation is associated with.
activator	The activator that is controlling the object’s activation.
block	Indicates whether the call completes or blocks waiting for requests.

```
public void obj_is_ready(Object object)
```

This method makes an object implementation provided by a server visible on the network. Servers that implement more than one object must make a separate call to this method for each object that they offer.

Parameter	Description
<code>object</code>	The server's implementation object to be activated.

```
public abstract void obj_is_ready(Object object, java.lang.String service_name, byte ref_data[])
```

This method makes an object implementation provided by a server visible on the network. This method is used when you are activating objects which are also associated with an Activator.

Parameter	Description
<code>object</code>	The server's implementation object to be activated.
<code>service_name</code>	The service name associated with the object implementation to be activated.
<code>ref_data</code>	The reference data associated with the object to be activated.

## CompletionStatus

---

```
public final class CompletionStatus extends Object
```

This class works with `SystemException` and indicates whether the operation completed or not before the exception was raised.

### IDL definition

---

```
enum CompletionStatus {
    COMPLETED_YES,
    COMPLETED_NO,
    COMPLETED_MAYBE
};
```

### CompletionStatus methods

---

For more information about these methods, see “Enum” on page 3-11.

```
public final static int _COMPLETED_YES
public final static int _COMPLETED_NO
public final static int _COMPLETED_MAYBE
public final static CompletionStatus COMPLETED_YES
public final static CompletionStatus COMPLETED_NO
public final static CompletionStatus COMPLETED_MAYBE
public int value()
public static CompletionStatus from_int(int value)
```

# Context

---

public interface **Context**

The `Context` interface contains a property list for a client. This property list is propagated to the server when a client makes a request. The CORBA specification does not define the contents of a `Context` so the use of these properties are left to the user and implementor to define. `Context` objects are organized as a tree with each containing a pointer to its parent context. The root context is the *global default context*, whose parent is null. The default context is obtained by using the `ORB.get_default_context` method, described in “abstract public org.omg.CORBA.Context get\_default\_context()” on page 5-19.

## IDL definition

---

```
interface Context {
    CORBA::Identifier context_name();
    CORBA::Context parent();
    void set_one_value(
        in CORBA::Identifier prop_name,
        in any value
    );
    void set_values(
        in CORBA::NVList values
    );
    CORBA::NVList get_values(
        in CORBA::Identifier start_scope,
        in boolean restrict_scope,
        in CORBA::Identifier prop_name
    );
    void delete_values(
        in CORBA::Identifier prop_name
    );
    CORBA::Context create_child(
        in CORBA::Identifier context_name
    );
};
```

## Context methods

---

public java.lang.String **context\_name()**

This method returns the name of this `Context`.

public Context **create\_child**(java.lang.String **context\_name**)

This method creates a child (leaf) Context with the specified parent Context. This method returns the newly created child context.

Parameter	Description
context_name	The name of the child Context to be created.

public void **delete\_values**(java.lang.String **prop\_name**)

This method removes all properties with the specified name from the current Context. You can use an asterisk as a wildcard character at the end of the prop\_name.

Parameter	Description
prop_name	The name of the property to be removed.

public org.omg.CORBA.NVList **get\_values**(java.lang.String **start\_scope**, boolean **restrict\_scope**, java.lang.String **prop\_name**)

This method returns the properties associated with the current Context as an NVList of name-value pairs. Scope of the Context search may be limited using the start\_scope and restrict\_scope parameters. You can use an asterisk as a wildcard at the end of the prop\_name.

This method returns a name value list for the specified search. If the start\_scope is not null and the corresponding Context is not found, this method throws BAD\_PARAM.

Parameter	Description
start_scope	The name of the Context where the search is to begin.
restrict_scope	True indicates that the search is only for the current context or the scope matching start_scope, if not null. False indicates that the search includes the current context as well as its ancestors.
prop_name	The name of the property to be returned.

public org.omg.CORBA.Context **parent**()

This method returns the parent Context for this object. If this object is the default global context, NULL is returned.

public void **set\_one\_value**(java.lang.String **prop\_name**, org.omg.CORBA.Any **value**)

This method adds a new property to the current Context. The value of the property is represented by the Any class, described in “Any” on page 6-1.

Parameter	Description
prop_name	The name of the new property.
value	An Any object that contains the value of the new property.

```
public void set_values(org.omg.CORBA.NVList values)
```

This method sets the properties of the current `Context` using the supplied `NVList`, containing one or more name-value pairs. See “NVList” on page 6-22.

Parameter	Description
values	The list of properties for the Context.

## InvalidName

---

```
public class org.omg.CORBA.ORBPackage.InvalidName extends org.omg.CORBA.UserException
```

The exception is raised by the `ORB.resolve_initial_reference` method, described in “abstract public org.omg.CORBA.Object `resolve_initial_references(java.lang.String identifier)` throws `org.omg.CORBA._ORB.InvalidName`” on page 5-20.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

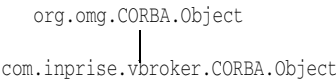
## Object

---

```
public interface Object
```

The `Object` interface is the root of the CORBA inheritance hierarchy. All interfaces defined in IDL inherit from this interface. This interface provides platform independent runtime type information and object reference equivalence testing.

`VisiBroker` extends the standard `org.omg.CORBA.Object` with a few more methods. The hierarchy is:



Both of these classes are described in the following information.

### org.omg.CORBA Object definition

---

```
package org.omg.CORBA;
public interface Object {
    Request _create_request(
        Context ctx,
        String operation,
        NVList arg_list,
        NamedValue result
    );
    Request _create_request(
```



```

    Context ctx,String operation,
    NVList arg_list,
    NamedValue result,
    ExceptionList exclist,
    ContextList ctxlist
);
org.omg.CORBA.Object _duplicate();
DomainManager[] _get_domain_managers();
org.omg.CORBA.Object _get_interface_def();
Policy _get_policy(int policy_type);
int _hash(int maximum);
boolean _is_a(String repositoryIdentifier);
boolean _is_equivalent(org.omg.CORBA.Object other);
boolean _non_existent();
void _release();
Request _request(String operation);
org.omg.CORBA.Object _set_policy_override(
    Policy[] policies,
    SetOverrideType set_add
);
}

```

## org.omg.Object methods

---

public org.omg.CORBA.Request **\_create\_request**(org.omg.CORBA.Context **ctx**,  
java.lang.String **operation**, org.omg.CORBA.NVList **arg\_list**, org.omg.CORBA.NamedValue **result**)

This method creates an dynamic invocation request initialized with the specified parameters. For information on obtaining the default context, see `get_default_context` in “abstract public org.omg.CORBA.Context `get_default_context()`” on page 5-19.

Parameter	Description
ctx	The Context to use for the dynamic invocation request.
operation	The name of the operation to be invoked.
arg_list	A list of NamedValue items. There is one NamedValue for each argument that is to be passed to the operation.
result	The type of the return value.

public org.omg.CORBA.Request **\_create\_request**(org.omg.CORBA.Context **ctx**,  
java.lang.String **operation**, org.omg.CORBA.NVList **arg\_list**, org.omg.CORBA.NamedValue **result**,  
org.omg.CORBA.Typecode[] **exceptions**, java.lang.String[] **contexts**)

This method creates an dynamic invocation request initialized with the specified parameters, including a list of exceptions that the request may raise. For information on obtaining the default context, see

`get_default_context` in “abstract public org.omg.CORBA.Context  
`get_default_context()`” on page 5-19.

Parameter	Description
<code>ctx</code>	The Context to use for the dynamic invocation request.
<code>operation</code>	The name of the operation to be invoked.
<code>arg_list</code>	A list of NamedValue items. There is one NamedValue for each argument that is to be passed to the operation.
<code>exceptions</code>	A list of Typecode objects representing the exceptions that this request might raise.
<code>contexts</code>	A list of Context objects. The list of Context objects supports type checking on context names.

`public int _hash(int maximum)`

This method computes a hash value for this object in the range of [0 - `maximum`]. The value returned is always positive.

Parameter	Description
<code>maximum</code>	The maximum hash value to return.

`public boolean _is_a(java.lang.String repid)`

This method queries an object to see if it implements the specified interface. This method returns `true` if the implementation object supports the interface. Otherwise, `false` is returned.

Parameter	Description
<code>repid</code>	A String containing the repository identifier of the desired interface.

Note

Invoking this method may result in a call to the implementation object, since it is possible for a given server to simultaneously implement multiple interfaces via multiple inheritance, as shown in the following IDL code example.

```
module M {
  interface A {
    void opA();
  };
  interface B {
    void opB();
  };
  interface C : A, B {
  };
};
```

Given an interface A in module M, (that is, M::A) the corresponding repository identifier will generally be “IDL:M/A:1.0” (omitting the quotation marks). The repository identifier can also be set to an arbitrary string, using `#pragmas` in the IDL file.

```
public boolean _is_equivalent(org.omg.CORBA.Object other_object)
```

This method compares this object's IOR (Interoperable Object Reference) with the specified object's IOR and returns `true` if the IOR's are equivalent. Otherwise, `false` is returned. Under certain circumstances, you may get a false negative if the same implementation object can be reached by both (distinct) IOR's.

Parameter	Description
<code>other_object</code>	A reference to the object to be compared with this object.

```
public boolean _non_existent()
```

This method attempts to *ping* the implementation object to determine if it is active. This method returns `false` if the implementation object is currently active (possibly after causing the server to be activated). Otherwise, `true` is returned. This method does not cause a client's proxy object to rebind to another server. In other words, it does force a bind, but it does not force a rebind.

```
public org.omg.CORBA.Request _request(java.lang.String operation)
```

This method creates an empty dynamic invocation request. Both `IN` and `INOUT` parameters must be initialized prior to sending the request. The types must also be initialized for `out` parameters and return values. See "Request" on page 6-25 for more information initializing and sending dynamic invocation requests.

Parameter	Description
<code>operation</code>	The name of the operation to be invoked.

## VisiBroker extension to Object

```
Package com.inprise.vbroker.CORBA;
```

```
public interface Object extends org.omg.CORBA.Object {
    public void _bind();
    public BOA _boa();
    public org.omg.CORBA.Policy _get_client_policy(int policy_type);
    public org.omg.CORBA.Policy[] _get_policy_overrides(int[] types);
    public com.inprise.vbroker.IOP.IOR _ior();
    public com.inprise.vbroker.IOP.IORValue _ior_value();
    public boolean _is_bound();
    public boolean _is_local();
    public boolean _is_persistent();
    public boolean _is_remote();
    public java.lang.String _object_name();
    public org.omg.CORBA.ORB _orb();
    public java.lang.String _repository_id();
    public org.omg.CORBA.Object _resolve_reference(java.lang.String id);
    public boolean _validate_connection()
```

```

        org.omg.CORBA.PolicyListHolder inconsistent_policies
    );
}

```

## VisiBroker extension to Object methods

---

`public org.omg.CORBA.BOA _boa()`

This method returns either the BOA associated with a particular request or the default BOA if you are not in a request. If the BOA has not been initialized, a `CORBA.INITIALIZE` exception is raised.

`public boolean _is_bound()`

This method returns `true` if a TCP connection has been established with the implementation object. Otherwise, `false` is returned.

`public boolean _is_local()`

This method returns `true` if this object refers to an object implemented in the local address space. Otherwise, `false` is returned.

`public boolean _is_persistent()`

This method returns `true` if this object reference is valid beyond the lifetime of the process that implements the object. Otherwise, `false` is returned.

`public boolean _is_remote()`

This method returns `true` if this object refers to an object implemented in a remote address space. Otherwise, `false` is returned.

`public java.lang.String _object_name()`

This method is deprecated, and should only be used with BOA based objects.

This method returns the name of the object implementation. If the object is created by POA and is not named, the method returns `NULL`. Transient objects and foreign objects are not named.

`public java.lang.String _repository_id()`

This method returns the repository identifier of the object implementation's most derived interface.

`public org.omg.CORBA.Object _resolve_reference(java.lang.String id)`

Your client application can invoke this method on an object reference to resolve the server-side interface with the specified service identifier. This method causes the `ORB.resolve_initial_references` method, described in “abstract public org.omg.CORBA.Object `resolve_initial_references(java.lang.String identifier)` throws `org.omg.CORBA._ORB.InvalidName`” on page 5-20, to be invoked on the server-side to resolve the specified service, either the `ServerManager` or

ORManager. An object reference is returned which your client can narrow to the appropriate server type.

Parameter	Description
id	The name of the interface to be resolved on the server-side, either the <code>ServerManager</code> or <code>ORManager</code> .

## ORB

abstract public class **ORB**

This class provides a method for initializing the CORBA infrastructure, as shown in Code sample 5.3. The Object Request Broker, provides a variety of methods used by both clients and servers.

The JDK bundles an `org.omg.CORBA.ORB` class that is slightly older than the ORB specified in the CORBA 2.3 specification. VisiBroker extends the CORBA 2.3 ORB and adds a few more methods to the class. The hierarchy is:

```

org.omg.CORBA.ORB
|
org.omg.CORBA_2_3.ORB
|
com.inprise.vbroker.CORBA.ORB

```

### Code sample 5.3 Example client usage of the ORB class

```

public class SimpleClientProgram {
    public static void main(String args[]) {
        try {
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
            org.omg.CORBA.Object object = orb.string_to_object(args[0]);
            System.out.println("Contacted object: " + object);
        }
        catch(org.omg.CORBA.SystemException se) {
            System.out.println("Failure: " + se);
        }
    }
}

```

## JDK's ORB definition

```

abstract public class ORB {
    public void connect(org.omg.CORBA.Object obj);
    public org.omg.CORBA.TypeCode create_abstract_interface_tc(
        String id,
        String name
    );
    abstract public TypeCode create_alias_tc(
        String id,
        String name,
        TypeCode original_type
    );
}

```

```

);
abstract public Any create_any();
abstract public TypeCode create_array_tc(int length, TypeCode element_type);
abstract public ContextList create_context_list();
abstract public TypeCode create_enum_tc(String id, String name, String[] members);
abstract public Environment create_environment();
abstract public ExceptionList create_exception_list();
abstract public TypeCode create_exception_tc(
    String id,
    String name,
    StructMember[] members
);
public org.omg.CORBA.TypeCode create_fixed_tc(short digits, short scale);
abstract public TypeCode create_interface_tc(String id, String name);
abstract public NVList create_list(int count);
abstract public NamedValue create_named_value(String s, Any any, int flags);
public org.omg.CORBA.TypeCode create_native_tc(
    String id,
    String name
);
public NVList create_operation_list(org.omg.CORBA.Object oper);
abstract public org.omg.CORBA.portable.OutputStream create_output_stream();
public org.omg.CORBA.Policy create_policy(int type, org.omg.CORBA.Any val)
    throws org.omg.CORBA.PolicyError;
abstract public TypeCode create_recursive_sequence_tc(int bound, int offset);
public org.omg.CORBA.TypeCode create_recursive_tc(String id);
abstract public TypeCode create_sequence_tc(int bound, TypeCode element_type);
abstract public TypeCode create_string_tc(int bound);
abstract public TypeCode create_struct_tc(
    String id,
    String name,
    StructMember[] members
);
abstract public TypeCode create_union_tc(
    String id,
    String name,
    TypeCode discriminator_type,
    UnionMember[] members
);
public org.omg.CORBA.TypeCode create_value_box_tc(String id,
    String name,
    TypeCode boxed_type
);
public org.omg.CORBA.TypeCode create_value_tc(String id,
    String name,
    short type_modifier,
    TypeCode concrete_base,
    ValueMember[] members
);
abstract public TypeCode create_wstring_tc(int bound);
public void disconnect(org.omg.CORBA.Object obj);
public org.omg.CORBA.Current get_current();
abstract public Context get_default_context();
abstract public Request get_next_response() throws WrongTransaction;

```

```

abstract public TypeCode get_primitive_tc(TCKind tcKind);
public boolean get_service_information(
    short service_type,
    ServiceInformationHolder service_info
);
public static ORB init(String[] args, Properties props);
public static ORB init(Applet app, Properties props);
abstract public String[] list_initial_services();
abstract public String object_to_string(org.omg.CORBA.Object obj);
public void perform_work();
abstract public boolean poll_next_response();
abstract public org.omg.CORBA.Object resolve_initial_references(String object_name)
    throws InvalidName;
public void run();
abstract public void send_multiple_requests_oneway(Request[] req);
abstract public void send_multiple_requests_deferred(Request[] req);
abstract protected void set_parameters(Applet app, Properties props);
abstract protected void set_parameters(String[] args, Properties props);
public void shutdown(boolean wait_for_completion);
abstract public org.omg.CORBA.Object string_to_object(String str);
public boolean work_pending();
}

```

## JDK ORB methods

---

```

abstract public org.omg.CORBA.TypeCode create_alias_tc(
    java.lang.String repository_id,
    java.lang.String type_name,
    org.omg.CORBA.TypeCode original_type)

```

This method creates and returns a `TypeCode` describing an IDL alias.

Parameter	Description
<code>repository_id</code>	The repository identifier that specifies the type in IDL.
<code>type_name</code>	The unscoped type name of the type.
<code>original_type</code>	The aliased type.

```

abstract public org.omg.CORBA.Any create_any()

```

This method creates an empty `Any` object with a `NULL` type code.

```

public static org.omg.CORBA.TypeCode create_array_tc(CORBA::Ulong bound,
    TypeCode element_type);

```

This static method dynamically creates a `TypeCode` for an array.

Parameter	Description
<code>bound</code>	The maximum number of array elements.
<code>element_type</code>	The type of elements stored in this array.

`abstract public org.omg.CORBA.TypeCode create_array_tc(int length,  
org.omg.CORBA.TypeCode element_type)`

This method creates and returns a `TypeCode` describing an IDL array.

Parameter	Description
length	The length of the array.
element_type	The type of the elements contained in the array.

`public abstract ContextList create_context_list()`

This method creates and returns an empty `ContextList`.

`public org.omg.CORBA.DynAny create_dyn_any(org.omg.CORBA.Any value)`

Creates a `DynAny` object initializing it with the specified value.

Note `DynAny` objects cannot be used as parameters on operation requests or DII requests nor can they be externalized using the `ORB.object_to_string` method. See “`DynAny`” on page 6-6 for complete details.

Parameter	Description
value	An <code>Any</code> object used to initialize this object.

`abstract public org.omg.CORBA.TypeCode create_enum_tc(  
java.lang.String repository_id,  
java.lang.String type_name, java.lang.String members[])`

This method creates and returns a `TypeCode` describing an IDL enumeration.

Parameter	Description
repository_id	The repository identifier that specifies the type in IDL.
type_name	The unscoped type name of the type.
members	An array of strings defining the members of the type.

`abstract public org.omg.CORBA.Environment create_environment()`

This method creates and returns an empty `Environment`.

`abstract public org.omg.CORBA.TypeCode create_exception_tc(  
java.lang.String repository_id,  
java.lang.String type_name, org.omg.CORBA.StructMember members[])`

This method creates and returns a `TypeCode` describing an IDL exception.

Parameter	Description
repository_id	The repository identifier that specifies the type in IDL.
type_name	The unscoped type name of the type.
members	An array of structures defining the members of the type.



```
abstract public org.omg.CORBA.TypeCode create_interface_tc(
    java.lang.String repository_id,
    java.lang.String type_name)
```

This method creates and returns a `TypeCode` describing an IDL interface.

Parameter	Description
<code>repository_id</code>	The repository identifier that specifies the type in IDL.
<code>type_name</code>	The unscoped type name of the type.

```
abstract public org.omg.CORBA.NVList create_list(int length)
```

This method creates and returns an `NVList` of the specified length. For information on this method, see “`NVList`” on page 6-22.

Parameter	Description
<code>length</code>	The length of the list to be created.

```
abstract public org.omg.CORBA.NamedValue create_named_value(
    java.lang.String name
    org.omg.CORBA.Any value, int flags)
```

This method creates and returns a new `NamedValue` for the Dynamic Invocation Interface.

Parameter	Description
<code>name</code>	The name for the <code>NamedValue</code> .
<code>value</code>	The value for the <code>NamedValue</code> .
<code>flags</code>	The flags for the <code>NamedValue</code> : IN, OUT, or INOUT.

```
abstract public org.omg.CORBA.NVList create_operation_list(
    org.omg.CORBA.OperationDef operationDef)
```

This method creates and returns a new `NVList` for use with a Dynamic Invocation Interface request.

Parameter	Description
<code>operationDef</code>	The operation description that must be specified.

```
abstract public org.omg.CORBA.TypeCode create_recursive_sequence_tc(
    int length, int offset)
```

This method creates and returns a `TypeCode` describing an IDL sequence.

Parameter	Description
<code>length</code>	The length of the sequence to be created. A length of zero indicates an unbounded sequence is desired.
<code>offset</code>	The offset into the type code’s (recursive) definition.

```
abstract public org.omg.CORBA.TypeCode create_sequence_tc(
    int length, org.omg.CORBA.TypeCode element_type)
```

This method creates and returns a `TypeCode` describing an IDL sequence.

Parameter	Description
<code>length</code>	The length of the sequence to be created. A length of zero indicates an unbounded sequence is desired.
<code>element_type</code>	The type of the elements contained by the sequence.

```
abstract public org.omg.CORBA.TypeCode create_string_tc(int length)
```

This method creates and returns a `TypeCode` describing an IDL String.

Parameter	Description
<code>length</code>	The length of the String to be created. A length of zero indicates an unbounded string is desired.

```
abstract public org.omg.CORBA.TypeCode create_struct_tc(
    java.lang.String repository_id,
    java.lang.String type_name, org.omg.CORBA.StructMember members[])
```

This method creates and returns a `TypeCode` describing an IDL struct.

Parameter	Description
<code>repository_id</code>	The repository identifier that specifies the type in IDL.
<code>type_name</code>	The unscoped type name of the type.
<code>members</code>	An array of structures defining the members of the type.

```
abstract public org.omg.CORBA.TypeCode create_union_tc(
    java.lang.String repository_id,
    java.lang.String type_name, org.omg.CORBA.TypeCode discriminator_type,
    org.omg.CORBA.UnionMembers members[])
```

This method creates and returns a `TypeCode` describing an IDL union.

Parameter	Description
<code>repository_id</code>	The repository identifier that specifies the type in IDL.
<code>type_name</code>	The unscoped type name of the type.
<code>discriminator_type</code>	The type of the discriminator. The discriminator is the type used in the switch statement.
<code>members</code>	An array of structures defining the members of the type.

`abstract public org.omg.CORBA.TypeCode create_wstring_tc(int length)`

This method creates and returns a `TypeCode` describing an IDL `wString`, or Unicode string.

Parameter	Description
<code>length</code>	The length of the String to be created. A length of zero indicates an unbounded string is desired.

`abstract public org.omg.CORBA.Context get_default_context()`

This method returns the global default `Context`. Because the default context is a shared resource, any updates to it should be synchronized.

`abstract public org.omg.CORBA.Request get_next_response()`

This blocking method waits until a response to a deferred operation request is available. The completed `Request` is returned. See also `send_multiple_requests_deferred` in “`abstract public void send_multiple_requests_oneway(org.omg.CORBA.Request reqs[])`” on page 5-22.

`abstract public org.omg.CORBA.TypeCode get_primitive_tc(TCKind kind)`

This method returns the primitive type code associated with the kind. A `org.omg.CORBA.BAD_PARAM` exception is raised if `kind` is out of range or is not for a primitive data type.

Parameter	Description
<code>kind</code>	The kind of type code <code>kind</code> , as defined in <code>TCKind</code> .

`public static ORB init(Strings[] args, Properties props)`

This method initializes the ORB for use by an application and returns a new instance of the ORB.

Parameter	Description
<code>args</code>	The command-line arguments to pass to the program.
<code>props</code>	The properties you can set to customize the ORB's behavior.

`public static ORB init(Applet app, Properties props)`

This method initializes the ORB for use by an applet and returns a new instance of the ORB.

Parameter	Description
<code>app</code>	The applet to associate with this instance of the ORB.
<code>props</code>	The properties you can set to customize the ORB's behavior.

`abstract public java.lang.String[] list_initial_services()`

This method returns a list of names of any object services initially available to the process. Some of the services include Location Service, Interface Repository, Name Service, or Event Service.

`abstract public java.lang.String object_to_string(org.omg.CORBA.Object obj)`

This method converts an object reference to a `String`, which is returned. The `String` is valid for the lifetime of the server or, if the implementation is registered with the activation daemon, for the lifetime of the registration and activation daemons. This method returns a stringified Internet Object Reference.

Parameter	Description
<code>obj</code>	The object reference to be converted.

`void perform_work();`

This method requests the ORB to perform some work.

`abstract public boolean poll_next_response()`

This method returns `true` if a response to a deferred operation request is available. Otherwise, `false` is returned. See also, `send_multiple_requests_deferred` in “`abstract public void send_multiple_requests_oneway(org.omg.CORBA.Request reqs[])`” on page 5-22.

`abstract public org.omg.CORBA.Object resolve_initial_references(java.lang.String identifier)`  
throws `org.omg.CORBA._ORB.InvalidName`

This method resolves one of the names returned by the `list_initial_services` method to its corresponding implementation object. It returns the resolved object, which can be narrowed to the appropriate server type. If the specified name is not found, an `org.omg.CORBA.InvalidName` exception is raised.

Parameter	Description
<code>identifier</code>	The identifier is the name of the service which is used to resolve an initial object reference; it is not the name of the object (as would be specified in the <code>Helper.bind</code> method).

There are a list of initial services provided by the ORB. These services enable your program to access the ORB’s internal functions. You can use those functions via `resolve_initial_references`. See also, `Object._resolve_reference` in “`public org.omg.CORBA.Object _resolve_reference(java.lang.String id)`” on page 5-13.

The initial services provided by the ORB as add-ons are the Interface Repository, the Handler Registry, three interceptor services, two untyped

object wrapper services, and URL naming (also known as Web Naming). The following table provides information about the add-on services:

#### Value of the string's identifier

ChainUntypedObjectWrapperFactory  
 DistributedService  
 DynAnyFactory  
 InterfaceRepository  
 LocationService  
 NameService  
 ORBPolicyManager  
 PolicyCurrent  
 POACurrent  
 RootPOA  
 URLNamingResolver  
 VBRootPOA  
 VisiBrokerInterceptorControl

abstract public void **run**();

This method orders the ORB to receive requests and dispatch them to start processing work. This call blocks this process until the ORB is shut down.

abstract public void **send\_multiple\_requests\_deferred**(org.omg.CORBA.Request reqs[])

This non-blocking method sends a number of operation requests. Return values may then be obtained using the `poll_next_response` and `get_next_response` methods.

Parameter	Description
reqs	The operation requests.

abstract public void **send\_multiple\_requests\_oneway**(org.omg.CORBA.Request reqs[])

This method sends a number of *oneway* operation requests. Return values are not provided for *oneway* requests.

Parameter	Description
reqs	The server's implementation object to be activated.

abstract public org.omg.CORBA.Object **string\_to\_object**(java.lang.String ior)

This method converts a *String* to an object reference. The *Object* that is returned can be narrowed to a specific interface. If the *ior* parameter refers to an implementation object in the local address space, the resulting object will be a direct pointer reference to the implementation object. An

org.omg.CORBA.INV\_OBJREF exception will be raised if the ior parameter is invalid.

Parameter	Description
ior	An Internet Object Reference that was previously created with the object_to_string method.

public boolean **work\_pending**

This method returns true if the ORB needs the main thread to perform some work. It returns false if the ORB does not need the main thread.

## OMG ORB definition

```
package org.omg.CORBA_2_3;

abstract public class ORB extends org.omg.CORBA.ORB {
    public org.omg.CORBA.portable.ValueFactory lookup_value_factory(String id);
    public org.omg.CORBA.portable.ValueFactory register_value_factory(
        String id,
        org.omg.CORBA.portable.ValueFactory factory
    );
    public void set_delegate(java.lang.Object object);
    public void unregister_value_factory(String id);
}
```

## VlsiBroker ORB extensions

```
package com.inprise.vbroker.CORBA;
public abstract class ORB extends org.omg.CORBA_2_3.ORB {
    public org.omg.CORBA.Object bind(
        String fullPoaName,
        byte[] oid,
        String host_name,
        BindOptions bind_options
    );
    public org.omg.CORBA.Object bind(
        String repository_id,
        String object_name,
        String host_name,
        BindOptions bind_options
    );
    public BOA BOA_init();
    public BOA BOA_init(
        String boaType,
        java.util.Properties properties
    );
    public com.inprise.vbroker.CORBA.portable.InputStream create_input_stream(
        byte[] bytes
    );
    public com.inprise.vbroker.CORBA.portable.OutputStream create_output_stream(
```

```

        byte[] bytes
    );
    public BindOptions default_bind_options();
    public void default_bind_options(BindOptions options);
}

```

## VisiBroker ORB methods

---

```

public org.omg.CORBA.Object bind(java.lang.String repository_id,
    java.lang.String object_name,
    java.lang.String host_name,
    org.omg.CORBA.BindOptions bind_options)

```

This method attempts a bind on the ORB object and obtains a generic object reference.

Parameter	Description
<code>repository_id</code>	String identifying the repository id.
<code>object_name</code>	String identifying the ORB object's name.
<code>host_name</code>	String identifying the host's name where the ORB object is located.
<code>bind_options</code>	Bind options for this object.

---

**Note** This method is deprecated in VisiBroker 4.x.

```

public static org.omg.CORBA.BOA BOA_init()

```

This method initializes a BOA and returns a reference to the BOA. If you use this method rather than

```

    BOA_init(java.lang.String boaType, java.util.Properties properties)

```

the thread policy will be `TPool`.

Like the `init` method, `BOA_init` can be called repeatedly and at anytime to obtain a reference to the BOA.

**Note** This method is deprecated in VisiBroker 4.x.

```

public org.omg.CORBA.BOA BOA_init(java.lang.String boaType, java.util.Properties properties)

```

This method initializes a particular type of BOA with optional properties. It returns the adapter corresponding to the `boaType`. Adapter types include one of the following:

- `TPool`—Thread pooling
- `TSession`—Thread per session
- `SSLTPool`—SSL with thread pooling
- `SSLTSession`—SSL with thread per session

See Appendix A, “Using command-line options,” in the VisiBroker for Java *Reference* for a listing of properties that can be set with `BOA_init` and `ORB_init`.

For more information, see “BOA” on page 5-2.

This method can be called repeatedly and at any time to obtain a reference to the BOA.

Parameter	Description
boaType	A string identifying the type of BOA to be created.
properties	Properties to be passed to the BOA when it is created.

abstract public org.omg.CORBA.InputStream **create\_input\_stream**  
(org.omg.CORBA.OutputStream **ostream**)

This method creates an IIOP input stream from an IIOP output stream. All bytes written to the output stream will be available to be read from the input stream.

Parameter	Description
ostream	The output stream from which the input stream is created.

abstract public org.omg.CORBA.OutputStream **create\_output\_stream**()

This method creates an IIOP output stream. An array of bytes constituting an IIOP buffer can be extracted from the stream.

abstract public org.omg.CORBA.BindOptions **default\_bind\_options**()

This method returns the global default bind options, if set; otherwise it returns NULL. The bind options are stored by reference—be careful when modifying the value.

abstract public void **default\_bind\_options**(org.omg.CORBA.BindOptions **options**)

Caution

This method sets the global, default bind options.  
The global, default bind options are stored by reference. Use care when modifying the value.

Parameter	Description
options	The new bind options that are being specified.

abstract public Principal **default\_principal**()

This method returns the global, default Principal.

abstract public void **default\_principal**(Principal **principal**)

Caution

This method sets the global default Principal. The bind options are stored by reference—be careful when modifying the value.  
The global, default Principal is stored by reference. Use care when modifying the value.

Parameter	Description
principal	The new principal that is being specified.



```
public static ORB init()
```

This method returns the ORB singleton.

```
public static CORBA::ORB_ptr _nil();
```

This static method returns a `NULL` ORB pointer suitable for initialization purposes.

## PortableServer.AdapterActivator

---

Adapter activators are associated with Portable Object Adapters (POAs) which they supply with the ability to create child POAs on demand, as a side-effect of receiving a request which names the child POA (or one of its children), or when the `find_POA` method is called with an `activate` parameter set to `True`.

### PortableServer.AdapterActivator methods

---

```
boolean unknown_adapter(org.omg.PortableServer.POA parent, java.lang.String name);(POA_ptr  
parent, const char* name)
```

This method is called when the ORB receives a request for an object reference which identifies a target POA that does not exist. The ORB invokes this method once for each POA that must be created in order for the POA to exist (starting with the ancestor POA closest to the root POA).

Parameter	Description
<code>parent</code>	The parent POA associated with the adapter activator on which the method is to be invoked.
<code>name</code>	The name of the POA to be created (relative to the parent).

---

## PortableServer.Current

---

```
public class PortableServer.Current extends CORBA.Current
```

This class provides method with access to the identity of the object on which the method was called. The `Current` class is provided to support servants which implement multiple objects but can be used within the context of POA-dispatched method invocations on any servant.

### PortableServer.Current methods

---

```
PortableServer.ObjectId get_object_id();
```

This method returns the `ObjectId` which identifies the object in whose context is called. If called outside the context of a POA-dispatched method, a `NoContext` exception is raised.

PortableServer.POA **get\_POA()**;

This method returns a reference to the POA which implements the object in whose context it is called. If called outside the context of a POA-dispatched method, a `NoContext` exception is raised.

# PortableServer.POA

---

public interface **POA** extends **org.omg.CORBA.Object**, **org.omg.PortableServer.POA**, **org.omg.CORBA.portable.IDLEntity**

Objects of the POA class manage the implementation of a collection of objects. The POA supports a name space for these objects which are identified by Object Ids. A POA also provides a name space for other POAs in that a POA must be created as a child of an existing POA, which then forms a hierarchy starting with the root POA.

A POA object must not be exported to other processes or be stringified. A `MARSHAL` exception is raised if this is attempted.

## PortableServer.POA methods

---

byte[] **activate\_object**(org.omg.PortableServer.Servant **p\_servant**);

This method generates an object id, which is an array of bytes, and returns it. The object id and the specified `p_servant` are entered into the Active Object Map. If the `UNIQUE_ID` policy is present with the POA and the specified `p_servant` is already in the Active Object Map, then a `ServantAlreadyActive` exception is raised.

This method requires that the `SYSTEM_ID` and `RETAIN` policies be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>p_servant</code>	The <code>Servant</code> to be entered into the Active Object Map.

void **activate\_object\_with\_id**(byte[] **id**, org.omg.PortableServer.Servant **p\_servant**);

This method attempts to activate the specified `id` and to associate it with the specified `p_servant` in the Active Object Map. If the `id` already has a servant bound to it in the Active Object Map, then an `ObjectAlreadyActive` exception is raised. If the POA has the `UNIQUE_ID` policy present and the `p_servant` is already in the Active Object map, then a `ServantAlreadyActive` exception is raised.

This method requires that the `RETAIN` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>id</code>	The <code>ObjectId</code> of the object to be activated.
<code>p_servant</code>	The <code>Servant</code> to be entered into the Active Object Map.

```
org.omg.PortableServer.ImplicitActivationPolicy create_implicit_activation_policy(
    org.omg.PortableServer.ImplicitActivationPolicyValue value);
```

This method returns an `ImplicitActivationPolicy` object with the specified value.

If no `ImplicitActivationPolicy` is specified at POA creation, then the default is `NO_IMPLICIT_ACTIVATION`.

Parameter	Description
<code>value</code>	If set to <code>IMPLICIT_ACTIVATION</code> , the POA will support implicit activation of servants; also requires <code>SYSTEM_ID</code> and <code>RETAIN</code> policies. If set to <code>NO_IMPLICIT_ACTIVATION</code> , the POA will not support the implicit activation of servants.

```
org.omg.CORBA.Object create_reference(String intf);
```

This method creates and returns an object reference that encapsulates a POA-generated `ObjectId` and the specified `intf` values. The `intf`, which may be a null string, becomes the `type_id` of the generated object reference. This method will not cause an activation to take place. An `intf` value that does not identify the most derived interface of the object or one of its base interfaces will result in undefined behavior.

This method requires that the `SYSTEM_ID` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>intf</code>	The repository interface id of the class of the object to be created.

```
org.omg.CORBA.Object create_reference_with_id(byte[]oid, String intf);
```

This method creates and returns an object reference that encapsulates the specified `oid` and `intf` values. The `intf`, which may be a null string, becomes the `type_id` of the generated object reference. An `intf` value that does not identify the most derived interface of the object or one of its base interfaces will result in undefined behavior. This method does not cause an activation to take place. The returned object reference may be passed to clients, so that subsequent requests on those references will cause the object to be activated

if necessary, or the default servant used, depending on the applicable policies.

Parameter	Description
oid	The object id for which a reference is to be created.
intf	The repository interface id of the class of the object to be created.

```
org.omg.PortableServer.IdAssignmentPolicy create_id_assignment_policy(  
    org.omg.PortableServer.IdAssignmentPolicyValue value);
```

This method returns an `IdAssignmentPolicy` object with the specified value.

If no `IdAssignmentPolicy` is specified at POA creation, then the default is `SYSTEM_ID`.

Parameter	Description
value	If set to <code>USER_ID</code> , then objects created the POA are assigned object ids only by the application. If set to <code>SYSTEM_ID</code> , then objects created with the POA are assigned object ids only by the POA.

```
org.omg.PortableServer.IdUniquenessPolicy create_id_uniqueness_policy(  
    org.omg.PortableServer.IdUniquenessPolicyValue value);
```

This method returns an `IdUniquenessPolicy` object with the specified value.

If no `IdUniquenessPolicy` is specified at POA creation, then the default is `UNIQUE_ID`.

Parameter	Description
value	If set to <code>UNIQUE_ID</code> , then servants which are activated with the POA support exactly one object id. If set to <code>MULTIPLE_ID</code> , then a servant which is activated with the POA may support one or more object ids.

```
org.omg.PortableServer.LifespanPolicy  
create_lifespan_policy(org.omg.PortableServer.LifespanPolicyValue value);
```

This method returns a `LifespanPolicy` object with the specified value.

If no `LifespanPolicy` is specified at POA creation, then the default is `TRANSIENT`.

Parameter	Description
value	If set to <code>TRANSIENT</code> , then objects implemented in the POA cannot outlive the POA instance in which they were first created. Once a transient POA is deactivated, the use of any object references generated from it will result in an <code>OBJECT_NOT_EXIST</code> exception being raised. If set to <code>PERSISTENT</code> , then the objects implemented in the POA can outlive any process in which they are first created.

```
org.omg.PortableServer.POA create_POA(String adapter_name,
org.omg.PortableServer.POAManager a_POAManager, org.omg.CORBA.PolicyList[] policies);
```

This method creates a new POA with the specified `adapter_name`. The new POA is a child of the POA object on which `create_POA` was called. If a child POA with the same name already exists for the parent POA, a `PortableServer.AdapterAlreadyExists` exception is raised.

The specified `policies` are associated with the new POA and used to control its behavior.

Parameter	Description
<code>adapter_name</code>	The name which specifies the new POA.
<code>a_POAManager</code>	The manager of the new POA.
<code>policies</code>	A list of policies which are to apply to the new POA.

The last two parameters may be NULL. If no `PortableServer.POAManager` is specified, one is created and associated with the new POA. If no policies are specified, the default set of policies is used.

```
org.omg.PortableServer.RequestProcessingPolicy
create_request_processing_policy(org.omg.PortableServer.RequestProcessingPolicyValue
value);
```

This method returns a `RequestProcessingPolicy` object with the specified value.

If no `RequestProcessingPolicy` is specified at POA creation, then the default is `USE_ACTIVE_OBJECT_MAP_ONLY`.

Parameter	Description
<code>value</code>	<p>If set to <code>USE_ACTIVE_OBJECT_MAP_ONLY</code> and the object id is not found in the Active Object Map, then an <code>OBJECT_NOT_EXIST</code> exception is returned to the client. (The <code>RETAIN</code> policy is also required.)</p> <p>If set to <code>USE_DEFAULT_SERVANT</code> and the object id is not found in the Active Object Map or the <code>NON_RETAIN</code> policy is present, and a default servant has been registered with the POA using the <code>set_servant</code> method, then the request is dispatched to the default servant. If no default servant has been registered, then an <code>OBJ_ADAPTER</code> exception is returned to the client. (The <code>MULTIPLE_ID</code> policy is also required.)</p> <p>If set to <code>USE_SERVANT_MANAGER</code> and the object id is not found in the Active Object Map or the <code>NON_RETAIN</code> policy is present, and a servant manager has been registered with the POA using the <code>set_servant_manager</code> method, then the servant manager is given the opportunity to locate a servant or raise an exception. If no servant manager has been registered, then an <code>OBJ_ADAPTER</code> is returned to the client.</p>

```
org.omg.PortableServer.ServantRetentionPolicy create_servant_retention_policy(
org.omg.PortableServer.ServantRetentionPolicyValue value);
```

This method returns a `ServantRetentionPolicy` object with the specified value.

If no `ServantRetentionPolicy` is specified at POA creation, then the default is `RETAIN`.

Parameter	Description
value	If set to <code>RETAIN</code> , then the POA will retain active servants in its Active Object Map. If set to <code>NON_RETAIN</code> , then servants are not retained by the POA.

```
org.omg.PortableServer.ThreadPolicy
create_thread_policy(org.omg.PortableServer.ThreadPolicyValue value);
```

This method returns a `ThreadPolicy` object with the specified value.

If no `ThreadPolicy` is specified at POA creation, then the default is `ORB_CTRL_MODEL`.

Parameter	Description
value	If set to <code>ORB_CTRL_MODEL</code> , the ORB is responsible for assigning requests for an ORB-controlled POA to threads. In a multi-threaded environment, concurrent requests may be delivered using multiple threads. If set to <code>SINGLE_THREAD_MODEL</code> , then requests to the POA are processed sequentially. In a multi-threaded environment, all upcalls made by the POA to servants and servant managers are made in a manner that is safe for code that is multi-thread unaware.

```
void deactivate_object(byte[] oid);
```

This method causes the specified `oid` to be deactivated. An `ObjectId` which has been deactivated continues to process requests until there are no more active requests for that `ObjectId`. An `ObjectId` is removed from the Active Object Map when all requests executing for that `ObjectId` have completed.

If a `ServantManager` is associated with the POA, then the `ServantActivator.etheralize` method is invoked with the `ObjectId` and the associated servant after the `ObjectId` has been removed from the Active Object map. Reactivation for the `ObjectId` blocks until etherealization, if necessary, has completed. However, the method does not wait for requests or etherealization to complete and always returns immediately after deactivating the specified `oid`.

This method requires that the `RETAIN` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
oid	The <code>ObjectId</code> of the object to be deactivated.

void **destroy**(boolean **etherealize\_objects**, boolean **wait\_for\_completion**);

This method destroys this POA object and all of its descendant POAs. First the children are destroyed and finally the current container POA. If desired, later a POA with that same name in the same process can be created.

Parameter	Description
<code>etherealize_objects</code>	If <code>True</code> , the POA has the <code>RETAIN</code> policy, and a servant manager has registered with the POA, then the <code>etherealize</code> method is called on each active object in the Active Object Map. The apparent destruction of the POA occurs before the <code>etherealize</code> method is called, and thus any <code>etherealize</code> method which attempts to invoke methods on the POA raises a <code>OBJECT_NOT_EXIST</code> exception.
<code>wait_for_completion</code>	If <code>True</code> and the current thread is not in an invocation context dispatched from some POA belonging to the same ORB as this POA, the <code>destroy</code> method only returns after all active requests and all invocations of <code>etherealize</code> have completed.  If <code>True</code> and the current thread is in an invocation context dispatched from some POA belonging to the same ORB as this POA, the <code>BAD_INV_ORDER</code> exception is raised and POA destruction does not occur.

org.omg.PortableServer.POA **find\_POA**(String **adapter\_name**, boolean **activate\_it**);

If the POA object on which this method is called is the parent of the POA with the specified `adapter_name`, the child POA is returned.

Parameter	Description
<code>adapter_name</code>	The name of the AdapterActivator associated with the POA.
<code>activate_it</code>	If set to <code>True</code> and no child POA of the POA specified by <code>adapter_name</code> exists, then the POA's AdapterActivator, if not null, is invoked, and, if it successfully activates the child POA, then that POA is returned. Otherwise an <code>AdapterNonExistent</code> exception is raised.

org.omg.PortableServer.Servant **get\_servant**();Servant **get\_servant**();

This method returns the default `Servant` associated with the POA. If no `Servant` has been associated, then a `NoServant` exception is raised.

If neither the `RETAIN` policy nor the `USE_DEFAULT_SERVANT` policy is present, a `WrongPolicy` exception is raised.

org.omg.PortableServer.ServantManager **get\_servant\_manager**();

This method returns the `ServantManager` object associated with the POA. The result is null if no `ServantManager` is associated with the POA.

This method requires that the `USE_SERVANT_MANAGER` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

`org.omg.CORBA.Object id_to_reference(byte[] oid);`

This method returns an object reference if the specified `oid` value is currently active. If the `oid` is not active, then an `ObjectNotActive` exception is raised.

This method requires that the `RETAIN` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>oid</code>	The <code>ObjectId</code> of the object for which a reference is to be returned.

`org.omg.PortableServer.Servant id_to_servant(byte[] oid);`

This method has three behaviors:

- If the POA has the `RETAIN` policy present and the specified `oid` is in the Active Object Map, then it returns the servant associated with that object in the Active Object Map.
- If the POA has the `USE_DEFAULT_SERVANT` policy present and a default servant has been registered with the POA, it returns the default servant.
- Otherwise, an `ObjectNotActive` exception is raised.

This method requires that the `RETAIN` or `USE_DEFAULT_SERVANT` policy be present with the POA; if neither policy is present, a `WrongPolicy` exception is raised.

Parameter	Description
<code>oid</code>	The <code>ObjectId</code> of the object for which a servant is to be returned.

`org.omg.PortableServer.Servant reference_to_servant(org.omg.CORBA.Object reference);`

This method has three behaviors:

- If the POA has the `RETAIN` policy and the specified `reference` is present in the Active Object Map, then it returns the servant associated with that object in the Active Object Map.
- If the POA has the `USE_DEFAULT_SERVANT` policy present and a default servant has been registered with the POA, then it returns the default servant.
- Otherwise, it raises an `ObjectNotActive` exception.

This method requires the `RETAIN` or `USE_DEFAULT_SERVANT` policies to be present; otherwise, a `WrongPolicy` exception is raised.

If the `reference` was not created by the same POA, `reference_to_servant` raises as `WrongAdapter` exception.

Parameter	Description
<code>reference</code>	The object for which a servant is to be returned.



byte[] **reference\_to\_id**(org.omg.CORBA.Object **reference**);

This method returns the `ObjectId` value encapsulated by the specified `reference`. The invocation is valid only if the `reference` was created by the POA on which the method is called. If the `reference` was not created by the POA, a `WrongAdapter` exception is raised. The object denoted by the `reference` parameter does not have to be active for this method to succeed.

Though the IDL specifies that a `WrongPolicy` exception may be raised by this method, it is simply declared for possible future extension.

Parameter	Description
<code>reference</code>	The object for which an <code>ObjectId</code> is to be returned.

byte[] **servant\_to\_id**(org.omg.PortableServer.Servant **p\_servant**);

This method has four possible behaviors:

- If the POA has the `UNIQUE_ID` policy present and the specified `p_servant` is active, then the `ObjectId` associated with the `p_servant` is returned.
- If the POA has the `IMPLICIT_ACTIVATION` policy present and either the POA has the `MULTIPLE_ID` policy present or the specified `p_servant` is not active, then the `p_servant` is activated using the POA-generated `ObjectId` and the repository interface id associated with the `p_servant`, and that `ObjectId` is returned.
- If the POA has the `USE_DEFAULT_SERVANT` policy present, the specified `p_servant` is the default servant, then the `ObjectId` associated with the current invocation is returned.
- Otherwise, a `ServantNotActive` exception is raised.

This method requires that the `USE_DEFAULT_SERVANT` policy or a combination of the `RETAIN` policy and either the `UNIQUE_ID` or `IMPLICIT_ACTIVATION` policies be present; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>p_servant</code>	The Servant for which the <code>ObjectId</code> to be returned is desired.

org.omg.CORBA.Object **servant\_to\_reference**(org.omg.PortableServer.Servant **p\_servant**);

This method has four possible behaviors:

- If the POA has both the `RETAIN` and the `UNIQUE_ID` policies present and the specified `p_servant` is active, then an object reference encapsulating the information used to activate the servant is returned.
- If the POA has both the `RETAIN` and the `IMPLICIT_ACTIVATION` policies present and either the POA has the `MULTIPLE_ID` policy or the specified `p_servant` is not active, then the `p_servant` is activated using a POA-generated `ObjectId` and repository interface id associated with the `p_servant`, and a corresponding object reference is returned.

- If this method was invoked in the context of executing a request on the specified `p_servant`, the reference associated with the current invocation is returned.
- Otherwise, a `ServantNotActive` exception is raised.

This method requires the presence of the `RETAIN` policy and either the `UNIQUE_ID` or `IMPLICIT_ACTIVATION` policies if invoked outside the context of a method dispatched by the POA. If this method is not invoked in the context of executing a request on the specified `p_servant` and the policies specified previously are not present, then a `WrongPolicy` exception is raised.

Parameter	Description
<code>p_servant</code>	The <code>Servant</code> for which a reference is to be returned.

`void set_servant(org.omg.PortableServer.Servant p_servant);`

This method sets the default `Servant` associated with the POA. The specified `Servant` will be used for all requests for which no servant is found in the Active Object Map.

This method requires that the `USE_DEFAULT_SERVANT` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>p_servant</code>	The <code>Servant</code> to be used as the default associated with the POA.

`void set_servant_manager(org.omg.PortableServer.ServantManager imgr)`

This method sets the default `ServantManager` associated with the POA. This method may only be invoked after a POA has been created. Attempting to set the `ServantManager` after one has already been set raises a `BAD_INV_ORDER` exception.

This method requires that the `USE_SERVANT_MANAGER` policy be present with the POA; otherwise, a `WrongPolicy` exception is raised.

Parameter	Description
<code>imgr</code>	The <code>ServantManager</code> to be used as the default used with the POA.

`org.omg.PortableServer.AdapterActivator the_activator();`

This method returns the `AdapterActivator` associated with the POA. Upon creation, a POA does not have an `AdapterActivator` (i.e., the attribute is null). It is system-dependent whether a root POA has an activator and the application can assign one as it wishes.

void **the\_activator**(org.omg.PortableServer.AdapterActivator **the\_activator**);

This method sets the `AdapterActivator` object associated with the POA to the one specified. The application can assign an activator to the rootPOA.

Parameter	Description
<code>the_activator</code>	The <code>ActivatorAdapter</code> to be associated with the POA.

String **the\_name**();

This method returns the read-only attribute which identifies the POA relative to its parent. This attribute is assigned at POA creation. The name of the root POA is system dependent and should not be relied upon by the application.

org.omg.PortableServer.POA **the\_parent**();

This method returns the POA's parent POA. The parent of the root POA is null.

org.omg.PortableServer.POAManager **the\_POAManager**();

This method returns the POAManager associated with the POA.

org.omg.CORBA.Policy[ ] **the\_policies** ()

This method returns an array of the policies active for this POA.

Inprise added this method to its implementation of POA. This method may be included by the OMG to POA in a future version of the CORBA specification.

## PortableServer.POAManager

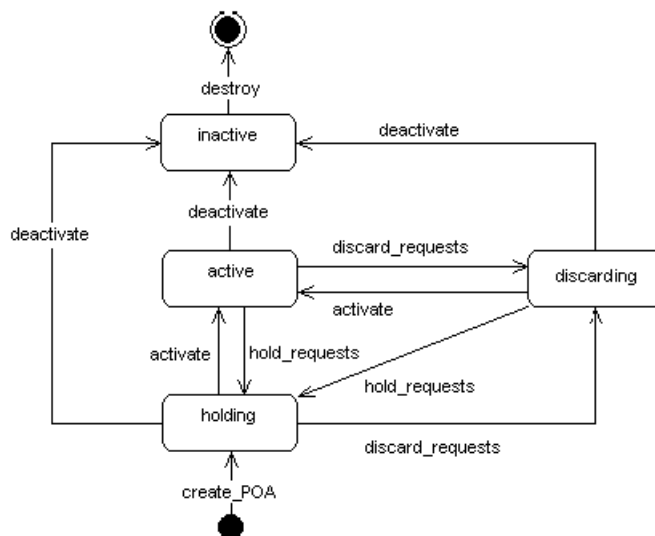
public interface **PortableServer.POAManager**

Each POA has an associated POA manager which in turn may be associated with one or more POA objects. A POA manager encapsulates the processing state of the POAs with which it is associated.

There are four possible states which a POA manager can be in:

- active
- inactive
- holding
- discarding

A POA manager is created in the holding state. The following diagram illustrates the states to which a POA manager transitions based on the method called.



## PortableServer.POAManager methods

**void activate();**

This method changes the state of the POA manager to active, which enables the associated POAs to process requests. If invoked while the POA manager is in the inactive state, the `AdapterInactive` exception is raised.

**void deactivate(**  
     CORBA.boolean etherealize\_objects,  
     CORBA.boolean wait\_for\_completion);

This method changes the state of the POA manager to inactive, which causes the associated POAs to reject requests that have not begun to be executed, as well as any new requests. If invoked while the POA manager is in the inactive state, the `AdapterInactive` exception is raised.

**void discard\_requests(boolean wait\_for\_completion);**

This method changes the state of the POA manager to discarding, which causes the associated POAs to discard incoming requests. In addition, any requests that have been queued but have not started executing are discarded. When a request is discarded, a `TRANSIENT` system exception is returned to the client. If invoked while the POA manager is in the inactive state, the `AdapterInactive` exception is raised.

**void hold\_requests();**

This method changes the state of the POA manager to holding, which causes the associated POAs to queue incoming requests. Any requests that have been queued but are not executing will continue to be queued while in the holding state. If invoked while the POA manager is in the inactive state, the `AdapterInactive` exception is raised.

## PortableServer.ServantActivator

---

public interface **ServantActivator** extends `org.omg.PortableServer.ServantManager`

If the POA has the `RETAIN` policy present, then it uses servant managers that are `PortableServer.ServantActivator` objects.

### PortableServer.ServantActivator methods

---

```
void etherealize(
    byte[] oid,
    org.omg.PortableServer.POA adapter,
    org.omg.PortableServer.Servant serv,
    boolean cleanup_in_progress,
    boolean remaining_activations);
```

This method is called by the specified `adapter` whenever a servant for an object (the specified `oid`) is deactivated, assuming that the `RETAIN` and `USE_SERVANT_MANAGER` policies are present.

Parameter	Description
<code>oid</code>	The object id of the object whose servant is to be deactivated.
<code>adapter</code>	The POA in whose scope the object was active.
<code>serv</code>	The servant which is to be deactivated.
<code>cleanup_in_progress</code>	If set to <code>True</code> , the reason for the invocation of the method is either that the deactivate or destroy method was called with the <code>etherealize_objects</code> parameter set to <code>True</code> ; otherwise, the method was called for other reasons.
<code>remaining_activations</code>	If the specified <code>serv</code> is associated with other objects in the specified <code>adapter</code> it is set to <code>True</code> ; otherwise it is <code>FALSE</code> .

---

```
org.omg.PortableServer.Servant incarnate(
    byte[] oid,
    org.omg.PortableServer.POA adapter);
```

This method is called by the POA whenever the POA receives a request for an inactive object (the specified `oid`) assuming that the `RETAIN` and `USE_SERVANT_MANAGER` policies are present.

The user supplies a servant manager implementation which is responsible for locating and creating an appropriate servant that corresponds to the specified `oid` value. The method returns a servant, which is also entered into

the Active Object map. Any further requests for the active object are passed directly to the servant associated with it without invoking the servant manager.

If this method returns a servant that is already active for a different object id and if the POA also has the `UNIQUE_ID` policy present, then it raises the `OBJ_ADAPTER` exception.

Parameter	Description
<code>oid</code>	The object id of the object whose servant is to be activated.
<code>adapter</code>	The POA in whose scope the object is to be activated.

## PortableServer.ServantLocator

public interface **PortableServer.ServantLocator** extends `org.omg.PortableServer.ServantManager`

When the POA has the `NON_RETAIN` policy present, it uses servant managers which are `PortableServer.ServantLocator` objects. Because the POA knows that the servant returned by the servant manager will be used only for a single request, it can supply extra information for the servant manager's methods and the servant manager's pair of methods may do something different than a `PortableServer.ServantLocator` servant manager.

### PortableServer.ServantLocator methods

`org.omg.PortableServer.Servant preinvoke(`  
    `byte[] oid,`  
    `org.omg.PortableServer.POA adapter,`  
    `String operation,`  
    `org.omg.PortableServer.ServantLocatorPackage.CookieHolder the_cookie);`

This method is called by the POA whenever the POA receives a request for an object that is not currently active, assuming that the `NON_RETAIN` and `USE_SERVANT_MANAGER` policies are present.

The user-supplied implementation of the servant manager is responsible for locating or creating an appropriate servant that corresponds to the specified `oid` value if possible.

Parameter	Description
<code>oid</code>	The <code>ObjectId</code> value that is associated with the incoming request.
<code>adapter</code>	The POA in which the object is to be activated.
<code>operation</code>	The name of the operation which will be called by the POA when the servant is returned.
<code>the_cookie</code>	An opaque value which can be set by the servant manager to be used later in the <code>postinvoke</code> method.

```
void postinvoke(
    byte[] oid,
    org.omg.PortableServer.POA adapter,
    String operation,
    Object the_cookie,
    org.omg.PortableServer.Servant the_servant);
```

This method is called whenever a servant completes a request, assuming that the POA has the `NON_RETAIN` and `USE_SERVANT_MANAGER` policies present. This method is considered to be part of the request on an object, (that is, if the method finishes normally, but `postinvoke` raises a system exception, then the method's normal return is overridden; the request completes with the exception).

Destroying a servant that is known to a POA can lead to undefined results.

Parameter	Description
<code>oid</code>	The <code>ObjectId</code> value that is associated with the incoming request.
<code>adapter</code>	The POA in which the object is to be activated.
<code>operation</code>	The name of the operation which will be called by the POA when the servant is returned.
<code>the_cookie</code>	An opaque value which can be set by the servant manager in the <code>preinvoke</code> method for use in this method.
<code>the_servant</code>	The servant associated with the object.

## PortableServer.ServantManager

```
public class PortableServer.ServantManager
```

Servant managers are associated with Portable Object Adapters (POAs). A servant manager allows a POA to activate objects on demand when the POA receives a request targeted for an inactive object.

The `PortableServer.ServantManager` class has no methods, rather it is the base class for two other classes: the `PortableServer.ServantActivator` and the `PortableServer.ServantLocator` classes. For more details, see “`PortableServer.ServantActivator`” on page 5-37 and “`PortableServer.ServantLocator`” on page 5-38. The use of these two classes depends on the POA's policies: `RETAIN` for the `PortableServer.ServantActivator` and `NON_RETAIN` for the `PortableServer.ServantLocator`.

## Principal

Note This feature is deprecated in VisiBroker 4.x.

```
abstract public class Principal
```

The `Principal` contains a sequence of bytes that a client application may associate with an operation request. Client applications can set a default

principal for a proxy object by using the `orb.default_principal` method described in “abstract public Principal default\_principal()” on page 5-25. The principal may also be retrieved using the `boa.get_principal` method described in “public Principal get\_principal(Object object)” on page 5-4.

## IDL definition

---

```
pseudo interface Principal {  
  attribute sequence<octet> name;  
}
```

## Principal methods

---

```
abstract public void name(byte[] name)
```

This method sets the value of the Principal.

Parameter	Description
name	The name to be set.

```
abstract public byte[] name()
```

This method returns the value of the Principal.







# Dynamic interfaces and classes

This chapter describes the dynamic interfaces and classes most of which are in the `org.omg.CORBA` package. `InputStream` and `OutputStream` are in the `org.omg.CORBA.portable` package. All of these interfaces or classes are used in the creation of client requests and object implementations at runtime.

## Any

---

public interface **Any** extends `org.omg.CORBA.Any`

The **Any** interface is used to store a value of any type in a type-safe manner and is used in the Dynamic Invocation Interface. The type stored in an **Any** is defined by a `TypeCode`. An **Any** can store a `String`, an interface object, or even another **Any**. Methods are provided to set and retrieve the contained value. VisiBroker 4.x does not support the `Fixed` and `Value` types. Using these two types will raise a `NO_IMPLEMENT` exception.

To create an **Any**, use `org.omg.CORBA.ORB.create_any()`. For more information, see the methods provided by “ORB” on page 5-13.

A `Holder` version of this interface is also provided, described in Chapter 4, “Generated interfaces and classes.”

## Any methods

---

public `org.omg.CORBA.portable.InputStream` **create\_input\_stream()**

This method creates an input stream containing the **Any**’s value.

public `org.omg.CORBA.portable.OutputStream` **create\_output\_stream()**

This method creates an empty output stream.

public boolean **equal**(Any **anAny**)

This method returns `true` if the value contained by the `Any` is the same as the value contained by the argument `Any.a`. Otherwise, `false` is returned.

Parameter	Description
<code>anAny</code>	The <code>Any</code> whose value is compared with the value of this <code>Any</code> .

public void **read\_value**(org.omg.CORBA.portable.InputStream **input**, org.omg.CORBA.Typecode **type**)

This method reads an `Any`'s value from an input stream given a type code. Only the `Any`'s value is read. To read the complete `Any` definition, including the type code, use `org.omg.CORBA.portable.InputStream.read_any`.

Parameter	Description
<code>input</code>	A GIOP input stream from which the specified type's value will be read.
<code>type</code>	The type to be read from the input stream. See "TypeCode" on page 6-31 for the possible values for this parameter.

public org.omg.CORBA.Typecode **type**()

This method returns the `TypeCode` representing the type contained in this `Any`.

public void **type**(org.omg.CORBA.Typecode **type**)

This method sets the `TypeCode` representing the type contained in this `Any`.

Parameter	Description
<code>type</code>	The type to be set for this <code>Any</code> object. See "TypeCode" on page 6-31 for the possible values for this parameter.

public void **write\_value**(org.omg.CORBA.portable.OutputStream **output**)

This method writes an `Any`'s value to an output stream. Only the `Any`'s value is written. To write the complete `Any` definition, including the type code, use `org.omg.CORBA.portable.OutputStream.write_any`.

Parameter	Description
<code>output</code>	A GIOP output stream into which the specified <code>Any</code> 's value is written.

## Any extraction methods

A set of methods is provided which return the type contained in this `Any`. Code sample 6.1 shows the name of each of the extraction methods. A `BAD_PARAM` exception is raised if the value contained in this `Any` does not match the expected return type for the extraction method used.

**Code sample 6.1** Extraction methods offered by the Any interface

```

public short      extract_short()
public int        extract_long()
public long       extract_longlong()
public short      extract_ushort()
public int        extract_ulong()
public long       extract_ulonglong()
public float      extract_float()
public double     extract_double()
public boolean    extract_boolean()
public char       extract_char()
public char       extract_wchar()
public byte       extract_octet()
public Any        extract_any()
public org.omg.CORBA.Object extract_Object()
public java.lang.String extract_string()
public java.lang.String extract_wstring()
public org.omg.CORBA.TypeCode extract_TypeCode()

```

## Any Insertion methods

---

A set of methods is provided that copies a particular type of value to this *Any*. Code sample 6.2 shows the list of methods provided for inserting various types. With one exception, all of the methods accept a single parameter that represents the type to be inserted.

The first `insert_Object` method inserts an `Object`.

The second `insert_Object` method inserts an `Object` with a particular `TypeCode`, effectively narrowing the object to a more specialized type. The second method will raise a `BAD_PARAM` exception if the `TypeCode` kind is not `TCKind.tk_objref`.

**Code sample 6.2** Insertion methods offered by the Any interface

```

public void      insert_short(short s);
public void      insert_long(int i);
public void      insert_longlong(long l);
public void      insert_ushort(short s);
public void      insert_ulong(int i);
public void      insert_ulonglong(long l);
public void      insert_float(float f);
public void      insert_fixed(java.math.BigDecimal value)
public void      insert_fixed(java.math.BigDecimal value,
                             org.omg.CORBA.Typecode type)

public void      insert_double(double d);
public void      insert_boolean(boolean b);
public void      insert_char(char c);
public void      insert_wchar(char c);
public void      insert_octet(byte b);
public void      insert_any(Any a);
public void      insert_Object(org.omg.CORBA.Object o);
public void      insert_Object(org.omg.CORBA.Object o,
                             org.omg.CORBA.TypeCode t)

```

## ARG\_IN

```
public void      insert_string(java.lang.String s)
public void      insert_wstring(java.lang.String s)
void             insert_Value(java.io.Serializable v)
void             insert_Value(java.io.Serializable v, org.omg.CORBA.
                    Typecode t)
public void      insert_TypeCode(org.omg.CORBA.TypeCode t);
public void      insert_Streamable(org.omg.CORBA.portable.Streamable s);
```

## ARG\_IN

---

public interface org.omg.CORBA.**ARG\_IN**

**ARG\_IN** is used to designate parameters for dynamic invocation interface requests that are only used for input purposes and will not be modified by the server.

See also    “Request” and “NVList.”

### Variables

---

public static int **value** = (int) 1;

## ARG\_INOUT

---

public interface org.omg.CORBA.**ARG\_INOUT**

**ARG\_INOUT** is used to designate parameters for dynamic invocation interface requests that are used for input and output purposes, but may also be modified by the server upon return to the client.

See also    “Request” and “NVList.”

### Variables

---

final public static int **value** = (int) 3;

## ARG\_OUT

---

public interface org.omg.CORBA.**ARG\_OUT**

**ARG\_OUT** is used to designate parameters for dynamic invocation interface requests that are used for output purposes, but may only be set by the server upon return to the client.

See also    “Request” and “NVList.”

## Variables

---

```
final public static int value = (int) 2;
```

## ContextList

---

public interface **ContextList** extends org.omg.CORBA.Object

A ContextList maintains a modifiable list of context strings used.

To create an instance of ContextList, use the `create_context_list` method provided by org.omg.CORBA.ORB. For more information, see “ORB” on page 5-13.

## IDL definition

---

```
interface ContextList {
    readonly attribute unsigned long count;
    void add(in string ctx);
    string item(in unsigned long index)
        raises(CORBA::Bounds);
    void remove(in unsigned long index)
        raises(CORBA::Bounds);
};
```

## ContextList methods

---

public void **add**(String ctx)

This method adds a string to the context list.

Parameter	Description
ctx	The name of the string to add to the context list.

public int **count**()

This method returns the number of elements in the context list.

public String **item**(int index) throws org.omg.CORBA.Bounds

This method returns an item in the context list. Bounds is raised if index number is not valid.

Parameter	Description
index	The index number of the item.

public void **remove**(int **index**) throws org.omg.CORBA.Bounds

This method deletes an item from the context list. Bounds is raised if index number is not valid.

Parameter	Description
index	The index number of the item.

## DynAny

public interface **DynAny** extends org.omg.CORBA

A `DynAny` object is used by a client application or server to create and interpret data types at runtime which were not defined at compile-time. A `DynAny` may contain a basic type (such as a `boolean`, `int`, or `float`) or a complex type (such as `struct` or `union`). The type contained by a `DynAny` is defined when it is created and may not be changed during the lifetime of the object.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

A `DynAny` object may represent a data type as one or more components, each with its own value. The `next`, `rewind`, and `current_component` methods are provided to help you navigate through the components.

The following interfaces are derived from `DynamicAny.DynAny` and provide support for constructed types that are dynamically managed.

Constructed type	Interface
Array	“ <code>DynArray</code> ” on page 6-9
Enumeration	“ <code>DynEnum</code> ” on page 6-11
Sequence	“ <code>DynSequence</code> ” on page 6-12
Structure	“ <code>DynStruct</code> ” on page 6-13
Union	“ <code>DynUnion</code> ” on page 6-14

## Important usage restrictions

`DynAny` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynAny` object to an `Any`, which can be used as a parameter. We do not support insert `extract_val` and fixed types. Those operations raise `no_implement` exceptions.



## DynAny methods

---

public void **assign**(org.omg.DynamicAny.DynAny **dyn\_any**) throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

Initializes the current component in this object from the specified DynAny.

An org.omg.DynamicAny.DynAnyPackage.Invalid exception is raised if the type contained in the Any does not match the type contained by this object.

public org.omg.DynamicAny.DynAny **copy**()

Returns a copy of this object.

public org.omg.DynamicAny.DynAny **current\_component**()

Returns the current component in this object.

public void **destroy**()

Destroys this object.

public void **from\_any**(any **value**) throws org.omg.DynamicAny.DynAnyPackage.Invalid.TypeMismatch,  
org.omg.DynanmicAny.DynAnyPackage.InvalidValue

Initializes the current component of this object from the specified Any object.

An org.omg.DynamicAny.DynAnyPackage.Invalid exception is raised if the TypeCode of value contained in the Any does not match the TypeCode defined for this object when it was created.

Parameter	Description
value	An Any object containing the value to set for this object.

public boolean **next**()

Advances to the next component, if one exists, and returns true. If there are no more components, false is returned.

public void **rewind**()

Returns to the first component contained in this object's sequence. A subsequent invocation of the `current_component` method will return the first component in the sequence.

If this object contains only one component, invoking this method will have no effect.

public boolean **seek**(int **index**)

If this object contains multiple components, this method advances to the component with the specified index and returns true. A subsequent invocation of the `current_component` method will return the component with the specified index.

If there is no component at the specified index, `false` is returned.

Parameter	Description
index	The zero-base index of the desired component.

```
public org.omg.CORBA.Any to_any( )
```

Returns an Any object containing the value of the current component.

```
public org.omg.CORBA.TypeCode type();
```

Returns the TypeCode for the value stored in the current component of this object.

## DynAny Extraction methods

A set of methods is provided which return the type contained in this `DynAny` object's current component. Code sample 6.3 shows the name of each of the extraction methods.

An `org.omg.DynamicAny.DynAnyPackage.Invalid` exception is raised if the value contained in this `DynAny` does not match the expected return type for the extraction method used.

**Code sample 6.3** Extraction methods offered by the `DynAny` interface

```
public org.omg.CORBA.Any get_any()
public org.omg.DynamicAny.DynAny get_dyn_any()
public boolean get_boolean()
public char get_char()
public double get_double()
public float get_float()
public int get_long()
public long get_longlong()
public byte get_octet()
public org.omg.CORBA.Object get_reference()
public short get_short()
public java.lang.String get_string()
public org.omg.CORBA.TypeCode get_typecode()
public int get_ulong()
public long get_ulonglong()
public short get_ushort()
public java.io.Serializable get_val()
public char get_wchar()
public java.lang.String get_wstring()
```

## DynAny Insertion methods

A set of methods is provided that copies a particular type of value to this `DynAny` object's current component. Code sample 6.4 shows the list of methods provided for inserting various types.

These methods will raise a `org.omg.DynamicAny.DynAnyPackage.InvalidValue` exception if the inserted object's type does not match the `DynAny` object's type.

#### Code sample 6.4 Insertion methods offered by the `DynAny` interface

```
public void insert_any(org.omg.CORBA.Any value)
public void insert_dyn_any(org.omg.DynamicAny.DynAny value)
public void insert_boolean(boolean value)
public void insert_char(char value)
public void insert_double(double value)
public void insert_float(float value)
public void insert_long(int value)
public void insert_longlong(long value)
public void insert_octet(byte value)
public void insert_reference(org.omg.CORBA.Object value)
public void insert_short(short value)
public void insert_string(java.lang.String value)
public void insert_typecode(org.omg.CORBA.TypeCode value)
public void insert_ulong(int value)
public void insert_ulonglong(long value)
public void insert_ushort(short value)
public void insert_val(java.io.Serializable value)
public void insert_wchar(char value)
public void insert_wstring(java.lang.String value)
```

## DynArray

---

public interface **DynArray** extends `org.omg.DynamicAny.DynAny`

This interface is used by a client application or server to create and interpret array data types at runtime which were not defined at compile-time. A `DynArray` may contain a sequence of basic type (such as a `boolean`, `int`, or `float`) or a constructed type (such as `struct` or `union`). The type contained by a `DynArray` is defined when it is created and may not be changed during the lifetime of the object.

The `next`, `rewind`, `seek`, and `current_component` methods, inherited from `DynamicAny`, may be used to navigate through the components.

Helper and Holder versions of this interface are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these interfaces and the methods they offer.

### Important usage restrictions

---

`DynArray` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynArray` object to a sequence of `Any` objects, which can be used as a parameter.

## DynArray methods

---

public org.omg.CORBA.Any[] **get\_elements()**

Returns a sequence of *Any* objects containing the values stored in this object.

public void **set\_elements**(org.omg.CORBA.Any[] **value**) throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch  
org.omg.DynamicAny.DynAnyPackage.InvalidValue

Sets the contained in this object from the specified sequence of *Any* objects.

An `org.omg.CORBA.DynAnyPackage.InvalidSeq` exception will be raised if the number of elements in *value* is not equal to the number of elements in this *DynArray*.

Parameter	Description
value	An array of <i>Any</i> objects whose values will be set in this <i>DynArray</i> .

---

## DynAnyFactory

---

public interface **DynAnyFactory** extends org.omg.CORBA.Object

A *DynAnyFactory* object is used to create a new *DynAny* object from an *any* value by invoking an operation on this object.

### Important usage restrictions

---

*DynAnyFactory* objects are intended to be local to the process in which they are created and used. Consequently, references to *DynAnyFactory* objects cannot be exported to other processes or externalized.

## DynAnyFactory methods

---

public org.omg.DynamicAny.DynAny **create\_dyn\_any** (org.omg.CORBA.Any value) throws  
org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode;

Creates a *DynAny* object. of the specified value

Parameter	Description
value	A new <i>DynAny</i> object of a specified value.

---

`public org.omg.DynamicAny.DynAny create_dyn_any_from_type_code (org.omg.CORBA.Typecode type) throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode;`

Creates a `DynAny` object of the specified type.

Parameter	Description
<code>type</code>	A new <code>DynAny</code> object of a specified type.

## DynEnum

`public interface DynEnum` extends `org.omg.DynamicAny.DynAny`

This interface is used by a client application or server to create and interpret enumeration values at runtime which were not defined at compile-time.

Since this type contains a single component, invoking the `DynAny.rewind` and `DynAny.next` methods on a `DynEnum` object will always return `false`.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

### Important usage restrictions

`DynEnum` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynEnum` object to an `Any`, which can be used as a parameter.

### DynEnum methods

`public java.lang.String get_as_string();`

Returns the `DynEnum` object’s value as a string.

`public void set_as_string(java.lang.String value)`

Sets the value contained in this `DynEnum` from the specified string.

Parameter	Description
<code>value</code>	A string that will be used to set the value in this <code>DynEnum</code> .

`public int get_as_ulong()`

Returns a `int` containing the `DynEnum` object’s value.

public void **set\_as\_ulong**(int value)

Sets the value contained in this `DynEnum` from the specified `int`.

Parameter	Description
value	An integer that will be used to set the value in this <code>DynEnum</code> .

# DynFixed

---

public interface **DynFixed** extends org.omg.DynamicAny.DynAny

This interface is used by a client application or server to create objects with values of the IDL fixed type.

## DynFixed methods

---

public java.lang.String **get\_value**()

Returns a value of a `DynFixed` object.

public boolean **set\_value** (java.lang.String val)

Sets the value of a `DynFixed` object.

# DynSequence

---

public interface **DynSequence** extends org.omg.DynamicAny.DynAny

This interface is used by a client application or server to create and interpret array data types at runtime which were not defined at compile-time. A `DynSequence` may contain a sequence of basic type (such as a `boolean`, `int`, or `float`) or a constructed type (such as a `struct` or `union`). The type contained by a `DynSequence` is defined when it is created and may not be changed during the lifetime of the object.

The `next`, `rewind`, `seek`, and `current_component` methods, inherited from `DynAny`, may be used to navigate through the components.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

## Important usage restrictions

---

`DynSequence` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynSequence` object to a sequence of `Any` objects, which can be used as a parameter.

## DynSequence methods

---

public org.omg.CORBA.Any[] **get\_elements()**

Returns a sequence of Any objects containing the values stored in this object.

public int **get\_length()**

Returns the number of components contained in this DynSequence.

public void **set\_length(int len)**

Sets the number of components contained in this DynSequence.

If you specify a length that is less than the current number of components, the sequence will be truncated and are the extra components.

Parameter	Description
length	The number of components to be contained in this DynSequence.

public void **set\_elements**(org.omg.CORBA.Any[] **value**) throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

Sets the contained in this object from the specified sequence of Any objects.

An org.omg.CORBA.DynAnyPackage.InvalidSeq exception will be raised if the number of elements in value is not equal to the number of elements in this DynSequence.

Parameter	Description
value	An array of Any objects whose values will be set in this DynArray.

## DynStruct

---

public interface **DynStruct** extends org.omg.DynamicAny.DynAny

This interface is used by a client application or server to create and interpret structures at runtime which were not defined at compile-time.

The next, rewind, seek, and current\_component methods, inherited from DynAny, may be used to navigate through the structure members.

You create an DynStruct object by invoking the ORB.create\_dyn\_struct method.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

## Important usage restrictions

---

`DynStruct` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynStruct` object to an `Any` objects, which can be used as a parameter.

## DynStruct methods

---

```
public java.lang.String current_member_name() throws
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
```

Returns a string containing the member name of the current component.

```
public org.omg.CORBA.TCKind current_member_kind() throws
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
```

Returns the `TypeCode` associated with the current component.

```
public org.omg.DynamicAny.NameValuePair[] get_members()
```

Returns the members of the structure as an array of `NameValuePair` objects.

```
public void set_members(org.omg.DynamicAny.NameValuePair[] value) throws
    org.omg.DynamicAny.DynAnyAckage.TypeMismatch,
    org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
```

Sets the structure members from the array of `NameValuePair` objects.

```
public org.omg.DynamicAny.NameDynAnyPair[] get_members_as_dyn_any();
```

Returns `DynAny` objects containing the values stored in this object.

```
public void set_members as_dyn_any (org.omg.DynamicAny.NameDynAnyPair[] throws
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
    org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
```

Sets the contained in this object of the `DynAny` objects.

An `org.omg.DynamicAny.DynAnyPackage.InvalidValue` exception will be raised if the order of the of elements in `value` is not identical to the order of the members in this `DynStruct`.

## DynUnion

---

```
public interface DynUnion extends org.omg.DynamicAny.DynAny
```

This interface is used by a client application or server to create and interpret unions at runtime which were not defined at compile-time. The `DynUnion` contains a sequence of two elements; the union discriminator and the actual member.



The `next`, `rewind`, `seek`, and `current_component` methods, inherited from `DynAny`, may be used to navigate through the components.

You create a `DynUnion` object by invoking the `ORB.create_dyn_union` method.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

## Important usage restrictions

---

`DynUnion` objects cannot be used as parameters on operation requests or DII requests, nor can they be externalized using the `ORB.object_to_string` method. However, you may use the `DynAny.to_any` method to convert a `DynUnion` object to an `Any` object, which can be used as a parameter.

## DynUnion methods

---

```
public org.omg.DynamicAny.DynAny get_discriminator()
```

Returns a `DynAny` object containing the discriminator for the union.

```
public void set_discriminator (org.omg.DynamicAny.DynAny d) throws  
    org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

Sets the discriminator of the `DynUnion` to the specified value.

```
public org.omg.CORBA.TCKind discriminator_kind()
```

Returns the type code of the discriminator kind for the union.

```
public org.omg.DynamicAny.DynAny member() throws  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns a `DynAny` object for the current component which represents a member in the union.

```
public org.omg.CORBA.TCKind member_kind() throws  
    org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the type code for the current component, which represents a member in the union.

```
public java.lang.String member_name() throws org.omg.DynamicAny.DynAnyPackage.InvalidValue;
```

Returns the member name of the current component.

```
public void set_to_default_member( throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch;
```

Sets the default member name of the current component to the specified name.

Parameter	Description
<code>member_name</code>	The member name to set for the current component.

---

public boolean **has\_no\_active\_member()**

Returns true if the current component is the default member.

public void **set\_to\_no\_active\_member()** throws org.omg.DynamicAny.DynAnyPackage.TypeMismatch;

Enables or disables the current component as the default member, based on the value of the `set_to_default_member` parameter.

Parameter	Description
set_to_no_active_member	If set to true, the current component will become the default member.

# DynValue

public interface DynValue extends org.omg.DynamicAny.DynAny

This interface is used by a client application or server to associate DynValue objects with value types.

## DynValue methods

public java.lang.String **current\_member\_name()** throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

Returns a string containing the member name of the current component.

public org.omg.CORBA.TCKind **current\_member\_kind()** throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

Returns the `TypeCode` associated with the current component.

public org.omg.DynamicAny.NameValuePair[] **get\_members()**

Returns the members of the structure as an array of `NameValuePair` objects.

public void **set\_members**(org.omg.DynamicAny.NameValuePair[] **value**) raises  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

Sets the structure members from the array of `NameValuePair` objects.

public org.omg.DynamicAny.NameDynAnyPair[] **get\_members\_as\_dyn\_any();**

Returns `DynAny` objects containing the values stored in this object.

public void **set\_members as\_dyn\_any** (org.omg.DynamicAny.NameDynAnyPair[] throws  
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,  
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

Sets the contained in this object of the `DynAny` objects.

An `org.omg.DynamicAny.DynAnyPackage.InvalidValue` exception will be raised if the order of the of elements in `value` is not identical to the order of the members in this `DynValue`.

## DynamicImplementation

---

public interface **DynamicImplementation** extends `org.omg.CORBA.portable.ObjectImpl`

The `DynamicImplementation` is an interface that provides a way to deliver requests from an ORB to any object implementation—even object implementations that do not have compile-time knowledge of the type of the objects they are implementing. This differs with the type-specific, IDL-based skeletons; however, they both serve the same function. The `DynamicImplementation` implements all requests on a particular object by having the ORB invoke an upcall to the implementation via the `invoke` method.

The ORB upcalls to the `DynamicImplementation`, passing a `ServerRequest` object. The `ServerRequest` pseudo object captures the explicit state of a request for the `DynamicImplementation`. For more information, see “`ServerRequest`” on page 6-29.

For more information about using dynamic skeletons, see Chapter 23, “Using the Dynamic Skeleton Interface,” in the *VisiBroker for Java Programmer’s Guide*.

## Constructors

---

protected **DynamicImplementation**(String `object_name`, String `repository_id`)

This constructor assumes that the interface has no other derived interfaces. If the interface has base interfaces, use the other constructor.

Parameter	Description
<code>object_name</code>	The name of the instance. If null, the instance is transient (anonymous).
<code>repository_id</code>	The interface’s repository identifier.

---

## DynamicImplementation methods

---

public void **invoke**(`org.omg.CORBA.ServerRequest request`)

This method provides the functionality of the server.

Parameter	Description
<code>request</code>	A description of the request which the server is to perform.

---

# Environment

---

public interface org.omg.CORBA.**Environment**

The Environment interface encapsulates an exception. It is used in conjunction with the dynamic invocation interface to provide a place for exceptions raised by asynchronous DII requests.

To create an instance of Environment, use `create_environment()` provided by org.omg.CORBA.ORB. For more information, see “ORB” on page 5-13.

## Environment methods

---

public void **clear**()

This method clears any `Exception` that may have been raised in the current Environment. This is the same as setting the exception to null.

public void **exception**(java.lang.Exception **except**)

This method sets the current exception. When setting, any previously stored exception will be lost.

Parameter	Description
exception	The exception to be set for the current Environment.

public java.lang.Exception **exception**()

This method returns the current `Exception` set for this Environment. If no `Exception` has been set, `NULL` is returned.

# ExceptionList

---

public interface **ExceptionList**

An `ExceptionList` is used in the DII (Dynamic Invocation Interface) to describe exceptions that can be raised by IDL operations. It maintains a modifiable list of type codes.

To create an instance of `ExceptionList`, use `create_exception_list()` provided by org.omg.CORBA.ORB. For more information, see “ORB” on page 5-13.

## IDL definition

---

```
interface ExceptionList {
    readonly attribute unsigned long count;
    void add(in CORBA::TypeCode exc);
    CORBA::TypeCode item(in unsigned long index) raises(CORBA::Bounds);
    void remove(in unsigned long index) raises(CORBA::Bounds);
};
```

## ExceptionList methods

---

public void **add**(TypeCode **exc**)

This method adds a type code to the exception list.

Parameter	Description
exc	The exception to add to the list.

public int **count**()

This method returns the number of items in the exception list.

public TypeCode **item**(int **index**) throws org.omg.CORBA.Bounds

This method returns an item from the list. Bounds is raised if the index number is not valid.

Parameter	Description
index	The index number of the item to be returned.

public void **remove**(int **index**) throws org.omg.CORBA.Bounds

This method removes an item from the exception list. Bounds is raised if the index number specified is not valid.

Parameter	Description
index	The index number of the item to be removed from the list.

## InputStream

---

public interface **InputStream**

The interface represents a General Inter-ORB Protocol (GIOP) input stream. Objects of this type are created with the `ORB.create_input_stream` method. All bytes written to an output stream can be read using the input stream methods. Several methods are provided for reading various data types.

See also `ORB.create_input_stream` and `ORB.create_output_stream` in the Methods section of ORB in Chapter 5, “Core interfaces and classes.”

## InputStream methods

---

The methods shown below are provided for reading data from an `InputStream`. Each method returns a particular data type.

**Code sample 6.5** Methods provided for reading data from an `InputStream`

```
public boolean read_boolean();
public char read_char();
```

```

public char read_wchar();
public byte read_octet();
public short read_short();
public short read_ushort();
public int read_long();
public int read_ulong();
public long read_longlong();
public long read_ulonglong();
public float read_float();
public double read_double();
public String read_string();
public String read_wstring();
public void read_boolean_array(boolean[] value,
                                int offset, int length);
public void read_char_array(char[] value,
                             int offset, int length);
public void read_wchar_array(char[] value,
                              int offset, int length);
public void read_octet_array(byte[] value,
                              int offset, int length);
public void read_short_array(short[] value,
                              int offset, int length);
public void read_ushort_array(short[] value,
                               int offset, int length);
public void read_long_array(int[] value,
                             int offset, int length);
public void read_ulong_array(int[] value,
                              int offset, int length);
public void read_longlong_array(long[] value,
                                 int offset, int length);
public void read_ulonglong_array(long[] value,
                                  int offset, int length);
public void read_float_array(float[] value,
                              int offset, int length);
public void read_double_array(double[] value,
                               int offset, int length);
public Object read_estruct(String expected_type)
public org.omg.CORBA.Object read_Object();
public org.omg.CORBA.TypeCode read_TypeCode();
public org.omg.CORBA.Any read_any();
public org.omg.CORBA.Principal read_Principal();

```

## Invalid

---

public interface **Invalid** extends org.omg.CORBA.UserException

An object of this interface is thrown if you attempt to assign an `Any` object to a `DynAny` object with an incompatible type.

# InvalidSeq

---

public interface **InvalidSeq** extends org.omg.CORBA.UserException

An object of this interface is thrown if you attempt to use an inconsistent value. For example, attempting to set a `DynSequence` with a sequence of `Any` objects that contains more elements than the `DynSequence`.

## NamedValue

---

public interface **NamedValue**

The `NamedValue` interface is used by a client to specify parameters and return values for a Dynamic Invocation Interface request. It includes a name, a value (an `Any`), and an integer representing a set of flags.

To create an instance of `NamedValue`, use `create_named_value(String name, Any value, int flags)` provided by `org.omg.CORBA.ORB`. For more information, see “ORB” on page 5-13 or the “NVList” on page 6-22.

## IDL definition

---

```
interface NamedValue {
    readonly attribute CORBA::Identifier name;
    readonly attribute any value;
    readonly attribute CORBA::Flags flags;
};
```

## NameValue methods

---

public int **flags()**

This method returns the flags for this `NamedValue`. See `ARG_IN`, `ARG_INOUT` and `ARG_OUT` in this chapter for more information.

public String **name()**

This method returns the name of this `NamedValue`. If no name has been set, `NULL` is returned.

public org.omg.CORBA.Any **value()**

This method returns an `Any` representing the current value set for this `NamedValue`. The value may be modified in place.

# NameValuePair

---

public interface **NameValuePair**

This interface is used to represent a structure member contained in a `DynStruct` object.

## NameValuePair variables

---

public java.lang.String **id**

Represents the name of the structure member.

public org.omg.CORBA.Any **value**

Represents the value and type of the structure member.

## NameValuePair constructors

---

public **NameValuePair**()

Creates an empty `NameValuePair`.

public **NameValuePair**(java.lang.String **id**, org.omg.CORBA.Any **value**)

Creates an `NameValuePair` initialized with the specified member name and value.

Parameter	Description
id	The name for the member.
value	The value for the member.

---

# NVList

---

public interface **NVList**

The `NVList` interface contains a set of `NamedValue` objects. It is used by client applications to pass the parameters associated with a Dynamic Invocation Interface request and in the context routines to describe context values. It maintains a modifiable list of `NamedValues`.

To create an instance of `NVList`, use `create_list` provided by `org.omg.CORBA.ORB`. For more information, see “ORB” on page 5-13.

## IDL definition

---

```
interface NVList {
    unsigned long count();
    void add(in CORBA::Flags flags);
    void add_item(in CORBA::Identifier name,in CORBA::Flags flags);
```



```

void add_value(in CORBA::Identifier name,in any value,
               in CORBA::Flags flags);
CORBA::NamedValue item(in unsigned long index);
void remove(in unsigned long index);
};

```

## NVList methods

---

public org.omg.CORBA.NamedValue **add**(int **flags**)

This method adds a `NamedValue` item to this list without initializing the name or the value associated with the item.

Parameter	Description
flags	The mode of the parameter to be added. Allowed values are org.omg.CORBA.ARG_IN.value, org.omg.CORBA.ARG_OUT.value, org.omg.CORBA.ARG_INOUT.value.

---

public org.omg.CORBA.NamedValue **add\_item**(String **item\_name**, int **flags**)

This method adds a `NamedValue` item to this list without initializing the value associated with the item.

Parameter	Description
item_name	The name of the item to be added.
flags	The mode of the parameter to be added. Allowed values are org.omg.CORBA.ARG_IN.value, org.omg.CORBA.ARG_OUT.value, org.omg.CORBA.ARG_INOUT.value.

---

public org.omg.CORBA.NamedValue **add\_value**(String **item\_name**,  
org.omg.CORBA.Any **value**, int **flags**)

This method adds a `NamedValue` item to this NVList that has the specified name, value and flags.

Parameter	Description
item_name	The name of the <code>NamedValue</code> to be added.
value	The value of the <code>NamedValue</code> , represented as an Any. The Any interface is described on page 6-1.
flags	The mode of the parameter to be added.

---

public int **count**()

This method returns the number of `NamedValue` items in this NVList.

public org.omg.CORBA.NamedValue **item**(int **index**) throws org.omg.CORBA.Bounds

This method returns the `NamedValue` item from this list that has the specified index. If the index is out of range, `Bounds` is raised.

Parameter	Description
index	The index of the <code>NamedValue</code> to be returned from this list.

public void **remove**(int **index**) throws org.omg.CORBA.Bounds

This method removes the `NamedValue` with the specified index from this list. If the index is out of range, `Bounds` is raised.

Parameter	Description
index	The index of the <code>NamedValue</code> item to be removed from this list.

## OutputStream

public interface **OutputStream**

The interface represents a General Inter-ORB Protocol (GIOP) output stream. Objects of this type are created by using the `ORB.create_output_stream` method. All bytes written to an output stream will be available to be read using the input stream. Several methods are provided for writing various data types.

See also `ORB.create_input_stream` and `ORB.create_output_stream`.

### OutputStream methods

The methods shown below are provided to write a particular type to this `OutputStream`. Each of these methods accept a single parameter that represents the type to be written.

**Code sample 6.6** Methods provided for writing a particular type to `OutputStream`

```
public inputStream create_input_stream();
public void write_boolean (boolean value);
public void write_char (char value);
public void write_wchar (char value);
public void write_octet (byte value);
public void write_short (short value);
public void write_ushort (short value);
public void write_long (int value);
public void write_ulong (int value);
public void write_longlong (long value);
public void write_ulonglong (long value);
public void write_float (float value);
public void write_double (double value);
public void write_string (String value);
public void write_wstring (String value);
```

```

public void write_boolean_array(boolean[] value,
                                int offset, int length);
public void write_char_array(char[] value,
                              int offset, int length);
public void write_wchar_array(char[] value,
                              int offset, int length);
public void write_octet_array(byte[] value,
                              int offset, int length);
public void write_short_array(short[] value,
                              int offset, int length);
public void write_ushort_array(short[] value,
                              int offset, int length);
public void write_long_array(int[] value,
                              int offset, int length);
public void write_ulong_array(int[] value,
                              int offset, int length);
public void write_longlong_array(long[] value,
                              int offset, int length);
public void write_ulonglong_array(long[] value,
                              int offset, int length);
public void write_float_array(float[] value,
                              int offset, int length);
public void write_double_array(double[] value,
                              int offset, int length);
public org.omg.CORBA.Object write_estruct(org.omg.CORBA.Object value,
                                           String expected_type)
public void write_Object(org.omg.CORBA.Object value);
public void write_TypeCode(org.omg.CORBA.TypeCode value);
public void write_any (org.omg.CORBA.Any value);
public void write_Principal(org.omg.CORBA.Principal value);

```

## Request

---

public interface **Request**

The **Request** interface represents a dynamic invocation request and provides methods for initializing and sending the request as well as receiving the response. An operation request may be sent synchronously, asynchronously, or as a *oneway* request for which no response is expected. Replies to invocations can be polled or obtained synchronously. The ORB interface can also be used to perform multiple simultaneous invocations, allowing for higher parallelism and reduced latency.

This object includes the following state information:

- Target object
- Operation name
- Argument types and values
- Return type and value
- Environment, described in “Environment” on page 6-18
- Context, described in “Context” on page 5-6

See the Object methods `_create_request` and `_request` for information on creating a Request.

## IDL definition

---

```
interface Request {
    readonly attribute CORBA::Object target;
    readonly attribute CORBA::Identifier operation;
    readonly attribute CORBA::NVList arguments;
    readonly attribute CORBA::NamedValue result;
    readonly attribute CORBA::Environment env;
    readonly attribute CORBA::ExceptionList exceptions;
    readonly attribute CORBA::ContextList contexts;

    attribute CORBA::Context ctx;

    any add_in_arg();
    any add_named_in_arg(in string name);
    any add_inout_arg();
    any add_named_inout_arg(in string name);

    any add_out_arg();
    any add_named_out_arg(in string name);
    void set_return_type(in ::CORBA::TypeCode tc);
    any return_value();

    void invoke();
    void send_oneway();
    void send_deferred();
    void get_response();
    boolean poll_response();
};
```

## Request methods

---

public Any **add\_in\_arg()**

Adds an IN argument to the request.

public Any **add\_inout\_arg()**

Adds an INOUT argument to the request.

public Any **add\_named\_in\_arg(String name)**

Adds a named IN argument to the request.

Parameter	Description
name	The name of the argument associated with this request.

public Any **add\_named\_inout\_arg**(String name)

Adds a named INOUT argument to the request.

Parameter	Description
name	The name of the argument associated with this request.

public Any **add\_named\_out\_arg**(String name)

Adds a named OUT argument to the request.

Parameter	Description
name	The name of the argument associated with this request.

public Any **add\_out\_arg**()

Adds an OUT argument to the request.

public org.omg.CORBA.NVList **arguments**()

This method returns the list of arguments for this request. These arguments must be initialized prior to sending the request.

public String[] **contexts**()

This method returns the context list. The list will be empty if the operation does not specify any contexts.

public org.omg.CORBA.Context **ctx**()

This method returns the context list associated with this request. See “ContextList” on page 6-5 for more information.

public void **ctx**(org.omg.CORBA.Context ctx)

This method sets the context for this request.

See also `ORB.get_default_context`

Parameter	Description
ctx	The Context.

public org.omg.CORBA.Environment **env**()

This method returns the `Environment` in which the request is invoked. Exceptions raised by the server will be put into the request’s `Environment`. For more information, see “Environment” on page 6-18.

public org.omg.CORBA.ExceptionList **exceptions**()

This method returns the list of user exception type codes. The list will be empty if no user exceptions can be raised by the operation.

public void **get\_response()**

This blocking method waits for the results of a dynamic invocation request that was sent with the `send_deferred` method. All `inout`, `out`, and `return` values are updated by this method.

The non-blocking `poll_response` method can be used to determine if a response is available prior to invoking this method.

public void **invoke()**

This method sends the request and blocks waiting for a response. If the client does not wish to block waiting for a response, the `send_deferred` method may be used instead.

public String **operation()**

This method returns the name of the operation, or method name, associated with this request.

public boolean **poll\_response()**

This method returns `true` if a response for an request is currently available. Otherwise, `false` is returned. This method is used after the `send_deferred` method has been invoked and prior to invoking the `get_response` method, which actually reads in the result values.

See also `ORB.poll_next_response`

public org.omg.CORBA.NamedValue **result()**

This method returns the results, or return value, of the request. If the type of the result is not specified, the type defaults to `void`. If the return type is not `void`, the type must be initialized prior to sending the request.

public Any **return\_value()**

This method gets the return value of the operation as an `Any`.

public void **send\_deferred()**

This method sends this request, but does not block waiting for a response. The `poll_response` and `get_response` methods are then used to determine when a response is available and to receive the results.

See also `send_multiple_requests_deferred` in “public void `send_deferred()`” on page 6-28.

public void **send\_oneway()**

This non-blocking method sends this request as a *oneway* request. Oneway requests do not result in a response from the server to which they are sent.

See also `ORB.send_multiple_requests_oneway`

```
public void set_return_type(TypeCode tc)
```

This method sets the type expected to be returned prior to invoking the operation.

Parameter	Description
tc	The type code to set.

```
public org.omg.CORBA.Object target()
```

This method returns the `target Object` to which this request will be sent. The `target Object` is specified when the Request is created, using the `Object methods _create_request`.

## ServerRequest

---

```
public interface ServerRequest
```

The `ServerRequest` interface, an important element when using dynamic skeletons, represents a request received by a server from a client. It provides methods for obtaining the `Context`, operation name, and operation parameters as well as a method for reflecting any `Exception` that may be raised during the processing of the request. This interface works with `DynamicImplementation` to provide dynamic skeletons. For more information about `DynamicImplementation`, see “`DynamicImplementation`” on page 6-17.

For more information about using dynamic skeletons, see Chapter 23, “Using the Dynamic Skeleton Interface,” in the *VisiBroker for Java Programmer’s Guide*.

### IDL definition

---

```
interface ServerRequest {
    readonly attribute ::CORBA::Identifier op_name;
    readonly attribute ::CORBA::Context ctx;
    void params(in CORBA::NVList params);
    void result(in any result);
    void except(in any except);
};
```

### ServerRequest methods

---

```
public org.omg.CORBA.Context ctx()
```

This method returns the current `Context` associated with this `ServerRequest`.

public void **except**(org.omg.CORBA.Any **except**)

This method is used by the server to set an `Exception` that occurred during the processing of the request so that it may be reflected to the client.

Parameter	Description
exception	The Exception that was raised.

public String **op\_name**()

This method returns the operation name associated with this request.

public void **params**(org.omg.CORBA.NVList **params**)

This method sets the parameters for this operation request. For more information about using this method, see Chapter 23, “Using the Dynamic Skeleton Interface,” in the VisiBroker for Java *Programmer’s Guide*.

Parameter	Description
params	The NVList where the parameters are to be stored.

public void **result**(org.omg.CORBA.Any **result**)

This method sets the result for this operation request.

Parameter	Description
result	The result to be set for the operation request.

# TCKind

public interface **TCKind** extends org.omg.CORBA.Object

The `TCKind` class contains the constants used in conjunction with `TypeCode` objects, which define the `TypeCode`. There are a set of integer constants, prefixed with `tk_`, that correspond to all the possible type codes. For example, the type code for float is `TCKind.tk_float`.

Helper and Holder versions of this interface are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these interfaces and the methods they offer.

## IDL definition

```
enum TCKind {
    tk_null, tk_void,
    tk_short, tk_long, tk_ushort, tk_ulong,
    tk_float, tk_double, tk_boolean, tk_char,
    tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
    tk_struct, tk_union, tk_enum, tk_string,
    tk_sequence, tk_array, tk_alias, tk_except,
```



```

tk_longlong, tk_ulonglong, tk_longdouble,
tk_wchar, tk_wstring, tk_fixed
tk_value, tk_value_box,
tk_native
tk, _interface
};

```

## TCKind methods

---

public int **value**()

This method returns an integral value representing the constant.

public static TCKind **from\_int**(int **value**)

This method returns an enum instance for the value you specify. For more information about enum mapping, see the “Enum” section.

Parameter	Description
value	The enum value.

---

## TypeCode

---

public interface **TypeCode**

The **TypeCode** interface describes the various types that are defined in IDL and allows them to be created and examined at run time. Type codes are most often used to describe the type of value being stored in an Any object, described in the “Any” section. Type codes may also be passed as parameters to method invocations.

Type codes are created using the various `ORB.create_<type>_tc` methods, described in Chapter 5, “Core interfaces and classes.” Type codes for all built-in types are provided by the **TCKind** interface, also described on Chapter 5.

A Holder version of this interface is also provided, described on Chapter 4, “Generated interfaces and classes.”

## IDL definition

---

```

interface TypeCode {
    exception Bounds {
    };
    exception BadKind {
    };
    boolean equal(in CORBA::TypeCode tc);
    CORBA::TCKind kind();
    CORBA::RepositoryId id() raises(CORBA::TypeCode::BadKind);
    CORBA::Identifier name() raises(CORBA::TypeCode::BadKind);
    unsigned long member_count() raises(CORBA::TypeCode::BadKind);
};

```

```
CORBA::Identifier member_name(in unsigned long index)
    raises(CORBA::TypeCode::BadKind, CORBA::TypeCode::Bounds);
CORBA::TypeCode member_type(in unsigned long index)
    raises(CORBA::TypeCode::BadKind, CORBA::TypeCode::Bounds);
any member_label(in unsigned long index)
    raises(CORBA::TypeCode::BadKind, CORBA::TypeCode::Bounds);
CORBA::TypeCode discriminator_type() raises(CORBA::TypeCode::BadKind);
long default_index() raises(CORBA::TypeCode::BadKind);
unsigned long length() raises(CORBA::TypeCode::BadKind);
CORBA::TypeCode content_type() raises(CORBA::TypeCode::BadKind);
long param_count();
any parameter(in long index) raises(CORBA::TypeCode::Bounds);
};
```

## TypeCode methods

---

```
public org.omg.CORBA.TypeCode content_type() throws
    org.omg.CORBA._TypeCodePackage.BadKind
```

This method returns the type code of the element contained in container types or an aliased type. This method is only valid for the following type codes:

- tk\_sequence
- tk\_array
- tk\_alias

A `BAD_PARAM` exception will be raised if the type code is not one of these types.

```
public int default_index() throws org.omg.CORBA._TypeCodePackage.BadKind
```

This method returns the default index of a union. This method is only valid for type codes `tk_union`, otherwise a `BAD_PARAM` exception will be raised.

```
public TypeCode discriminator_type() throws org.omg.CORBA._TypeCodePackage.BadKind
```

This method returns the type code of the discriminator of a union. This method is only valid when invoked on object with the type code of `tk_union`, otherwise a `BAD_PARAM` exception will be raised.

```
public boolean equal(org.omg.CORBA.TypeCode tc)
```

This method returns `true` if this object is equivalent to `tc`. Otherwise, `false` is returned. Type equivalence is determined by the structure of the types, not by their names. Two structures with the same fields, declared in the same order, are considered type equivalent.

Parameter	Description
tc	The TypeCode to be compared to this object's type.

```
public String id() throws org.omg.CORBA._TypeCodePackage.BadKind
```

This method returns the repository identifier of the type code. This string is used by IDL to define the type.

public TCKind **kind**()

This method returns the kind of type associated with this type code. Type codes kind constants are defined by TCKind, described in the “TCKind” section.

public int **length**() throws org.omg.CORBA.\_TypeCodePackage.BadKind

This method returns the number of elements contained by the type. Zero is returned if the number of elements is unbounded, such as for strings and sequences. This method is only valid for the following type codes:

- tk\_string
- tk\_sequence
- tk\_array

A `BAD_PARAM` exception will be raised if the type code is not one of these types.

public int **member\_count**() throws org.omg.CORBA.\_TypeCodePackage.BadKind

This method returns the number of members contained by the type. This method is only valid for the following type codes:

- tk\_struct
- tk\_union
- tk\_enum
- tk\_except

A `BAD_PARAM` exception will be raised if the type code is not one of these types.

public Any **member\_label**(int **index**) throws org.omg.CORBA.\_TypeCodePackage.BadKind,  
org.omg.CORBA.\_TypeCodePackage.Bounds

This method returns the label of the case statement associated with the member that has the specified index. This method is only valid for the type code `tk_union`, otherwise a `BAD_PARAM` exception will be raised. If the index is out of the bounds of the `String`, a `Bounds` exception will be raised.

Parameter	Description
index	The index of the member whose label is to be returned.

public String **member\_name**(int **index**) throws org.omg.CORBA.\_TypeCodePackage.BadKind,  
org.omg.CORBA.\_TypeCodePackage.Bounds

This method returns the name of the member that has the specified index. This method is only valid for the following type codes:

- tk\_struct
- tk\_union
- tk\_enum
- tk\_except

A `BAD_PARAM` exception will be raised if the type code is not one of these types. If the index is out of the bounds of the `String`, a `Bounds` exception will be raised.

Parameter	Description
index	The index of the member whose name is to be returned.

```
public org.omg.CORBA.TypeCode member_type(int index) throws
    org.omg.CORBA._TypeCodePackage.BadKind, org.omg.CORBA._TypeCodePackage.Bounds

This method returns the type code of the member that has the specified
index. This method is only valid for the following type codes:
```

- `tk_struct`
- `tk_union`
- `tk_except`

A `BAD_PARAM` exception will be raised if the type code is not one of these types. If the index is out of the bounds of the `String`, a `Bounds` exception will be raised.

Parameter	Description
index	The index of the member whose type code is to be returned.

```
public String name() throws org.omg.CORBA._TypeCodePackage.BadKind

This method returns the unscoped type name. This method is only valid for
the following type codes:
```

- `tk_objref`
- `tk_struct`
- `tk_union`
- `tk_enum`
- `tk_alais`
- `tk_except`

A `BAD_PARAM` exception will be raised if the type code is not one of these types.

# UnknownUserException

```
public interface UnknownUserException extends org.omg.CORBA.UserException

When a client issues a DII request and a user exception is raised, the specific
exception cannot be reflected to the client. This exception is used instead.
```

# Interface repository interfaces and classes

This chapter describes the interfaces and classes in the `org.omg.CORBA` package that are used with the interface repository.

## AbstractInterfaceDef

---

public interface **AbstractInterfaceDef** extends `org.omg.CORBA.AbstractInterfaceDefOperations`,  
`org.omg.CORBA.Container`, `org.omg.CORBA.Contained`,  
`org.omg.CORBA.IDLType`, `org.omg.CORBA.portable.IDLEntity`

The `AbstractInterfaceDef` is similar to the `InterfaceDef` interface, except that abstract interfaces can only inherit from abstract interfaces; they cannot use concrete interfaces as a base. It is used to represent an IDL abstract interface that is stored in the Interface Repository. This interface provides methods for setting and retrieving the base interfaces as well as creating attributes, operations and an interface description.

For more information about the `InterfaceDef` interface, see “InterfaceDef” on page 7-26.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## AbstractInterfaceDef methods

---

public `org.omg.CORBA.AbstractInterfaceDef[]` **base\_interfaces()**

This method returns the list of base interfaces of this object.

```
public void base_interfaces(org.omg.CORBA.InterfaceDef[] base_interfaces)
```

This method sets the base interface list for this object.

Parameter	Description
<code>base_interfaces</code>	The list of base interfaces to be set.

```
public org.omg.CORBA.InterfaceDef create_abstract_interface(java.lang.String id,  
java.lang.String name,  
java.lang.String version, org.omg.CORBA.AbstractInterfaceDef[] base_interfaces)
```

This method creates an `AbstractInterfaceDef` object in this `Container` with the specified attributes and returns a reference to the newly created object. Unlike a concrete `InterfaceDef`, this interface *cannot* contain definitions of both abstract and concrete interfaces. It can inherit only abstract interfaces.

Parameter	Description
<code>id</code>	The interface's repository id.
<code>name</code>	The interface's name.
<code>version</code>	The interface's version.
<code>base_interfaces</code>	A list of all interfaces from which this interface inherits.

```
public org.omg.CORBA.AttributeDef create_attribute(java.lang.String id, java.lang.String name,  
java.lang.String version, org.omg.CORBA.IDLType type,  
org.omg.CORBA.AttributeMode mode)
```

This method adds an attribute to an interface definition.

Parameter	Description
<code>id</code>	The attribute's identifier.
<code>name</code>	The attribute's name.
<code>version</code>	The attribute's version.
<code>type</code>	The attribute's IDL type.
<code>mode</code>	The attribute's mode. See <code>AttributeMode</code> in the section, "AttributeMode" on page 7-7, for possible values.

```
public org.omg.CORBA.OperationDef create_operation(java.lang.String id, java.lang.String name,  
java.lang.String version, org.omg.CORBA.IDLType result,  
org.omg.CORBA.OperationMode mode, org.omg.CORBA.ParameterDescription[] params,  
org.omg.CORBA.ExceptionDef[] exceptions, java.lang.String[] contexts)
```

This method adds an operation to an interface definition.

Parameter	Description
<code>id</code>	The operation's identifier.
<code>name</code>	The operation's name.
<code>version</code>	The operation's version.

Parameter	Description
result	The operation's IDL result type.
mode	The operation's mode. See "OperationMode" on page 7-35.
params	The list of parameters for this operation.
exceptions	The list of exceptions that can be raised by this operation.
contexts	The list of contexts.

public org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription **describe\_interface**()

This method returns an interface description for this object.

public boolean **is\_a**(java.lang.String **interface\_id**)

This method returns `true` if this object represents an interface definition that is compatible with a given `interface_id`.

Parameter	Description
interface_id	The interface identifier to compare with this object.

public boolean **is\_abstract** ();

If set to true, this method specifies that the represented interface is abstract.

Parameter	Description
is_abstract	Specifies the creation of an abstract interface.

public void **is\_abstract** (boolean **is\_abstract**);

This method sets the interface to abstract.

Parameter	Description
boolean is_abstract	Sets the object to abstract.

## AliasDef

public interface **AliasDef** extends org.omg.CORBA.AliasDefOperations  
 org.omg.CORBA.IDLType  
 org.omg.CORBA.portable.IDLEntity

This interface is used to represent a `typedef` that is stored in the Interface Repository. This interface provides methods for setting and obtaining the `IDLType` of the original `typedef`.

For more information on the `TypedefDef` interface, see "IDLType" on page 7-25.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## AliasDef methods

---

public void **original\_type\_def**(org.omg.CORBA.IDLType **original\_type\_def**)

This method sets the IDLType of this object.

Parameter	Description
original_type_def	The IDLType of this object.

public org.omg.CORBA.IDLType **original\_type\_def**()

This method returns the IDLType of the original typedef for which this object is an alias.

## ArrayDef

---

public interface **ArrayDef** extends org.omg.ArrayDefOperations, org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity

This interface is used to represent an array that is stored in the Interface Repository. This interface provides methods for setting and obtaining the type of elements in the array as well as the length of the array.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## ArrayDef methods

---

public int **length**()

This method returns the number of elements in the array.

public void **length**(int **length**)

This method sets the number of elements in the array.

Parameter	Description
length	The number of elements in the array.

public org.omg.CORBA.TypeCode **element\_type**()

This method returns the TypeCode of the array’s elements.

public void **element\_type\_def**(org.omg.CORBA.IDLType **element\_type\_def**)

This method sets the IDLType of the elements stored in the array.

Parameter	Description
element_type_def	The IDLType of the elements in the array.



```
public org.omg.CORBA.IDLType element_type_def()
```

This method returns the `IDLType` of the elements stored in this array.

## AttributeDef

---

```
public interface AttributeDef extends org.omg.CORBA.AttributeDefOperations,  
    org.omg.CORBA.Contained, org.omg.CORBA.portable.IDLEntity
```

This interface is used to represent an interface attribute that is stored in the Interface Repository. This interface provides methods for setting and obtaining the attribute's mode, and type.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### AttributeDef methods

---

```
public org.omg.CORBA.TypeCode type()
```

This method returns the `TypeCode` representing the attribute's type.

```
public void type_def(org.omg.CORBA.IDLType type_def)
```

This method sets the `IDLType` of the attribute.

Parameter	Description
<code>type_def</code>	The <code>IDLType</code> of this object.

```
public org.omg.CORBA.IDLType type_def()
```

This method returns the attribute's `IDLType`.

```
public org.omg.CORBA.AttributeMode mode()
```

This method returns the mode of the attribute. It might be either `AttributeMode ATTR_READONLY` for read-only attributes or `AttributeMode ATTR_NORMAL` for read-write attributes. See "AttributeMode" on page 7-7 for more information.

```
org.omg.CORBA.AttributeDef mode()
```

This method returns a value for the mode attribute.

```
public void mode (org.omg.CORBA.AttributeMode mode)
```

This method sets a value for mode attribute.

# AttributeDescription

---

public final class **AttributeDescription**

The `AttributeDescription` class describes an attribute that is stored in the interface repository. The `AttributeDescription` struct is used to fully describe Interfaces and Values. These are the only IDL types that can hold attributes.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## AttributeDescription variables

---

public java.lang.String **name**;

This variable represents the name of the attribute.

public java.name.String **id**;

This variable represents the repository id for the attribute.

public java.lang.String **defined\_in**;

This variable represents the repository id or value type of the interface in which the attribute is defined.

public java.lang.String **version**

This variable represents the attribute’s version.

public org.omg.CORBA.Typecode **type**;

This variable represents the attributes IDL type.

public org.omg.CORBA.AttributeMode **mode**;

This variable represents the mode of the attribute.

## AttributeDescription methods

---

public **AttributeDescription**()

This method is the default constructor for an `AttributeDescription`.

```
public AttributeDescription(java.lang.String name, java.lang.String id, java.lang.String defined_in,
    java.lang.String version, org.omg.CORBA.TypeCode type,
    org.omg.CORBA.AttributeMode mode)
```

This method constructs an `AttributeDescription`, using the supplied parameters.

Parameter	Description
<code>name</code>	The name of this attribute.
<code>id</code>	The repository id for this attribute.
<code>defined_in</code>	The interface or value type in which this attribute is defined.
<code>version</code>	The object's version.
<code>type</code>	The attribute's IDL type code.
<code>mode</code>	The mode of this attribute; read-only or read-write. See "AttributeMode" on page 7-7.

## AttributeMode

```
public final class AttributeMode
```

This class (the IDL's enumeration) is used to represent the *mode* of an attribute; either read-only or normal (read-write). Parameters may be used in one of the following two ways:

- **NORMAL** This mode specifies read and write access for this attribute.
- **READONLY** The mode specifies read only access for this attribute.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### AttributeMode enumeration elements

```
org.omg.CORBA.AttributeMode.ATTR_NORMAL
```

This variable is an attribute definition as Normal.

```
org.omg.CORBA.AttributeMode.ATTR_READONLY
```

This variable is an attribute definition as read-only.

# ConstantDef

---

public interface **ConstantDef** extends org.omg.CORBA.ConstantDefOperations, org.omg.Contained, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a constant definition that is stored in the interface repository. This interface provides methods for setting and obtaining the constant's type and value.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## ConstantDef methods

---

public org.omg.CORBA.TypeCode **type**()

This method returns the `TypeCode` representing the object's type.

public org.omg.CORBA.IDLType **type\_def**()

This method returns the constant's `IDLType`.

public void **type\_def**(org.omg.CORBA.IDLType **type\_def**)

This method sets the `IDLType` of the constant.

Parameter	Description
<code>type_def</code>	The <code>IDLType</code> of this constant.

---

public org.omg.CORBA.Any **value**()

This method returns an `Any` object representing the constant's value.

public void **value**(org.omg.CORBA.Any **value**)

This method sets the value for this constant.

Parameter	Description
<code>value</code>	An <code>Any</code> object that represents this object's value.

---

# ConstantDescription

---

public final class **ConstantDescription**

The `ConstantDescription` class describes a constant that is stored in the interface repository.

## ConstantDescription variables

---

public java.lang.String **name**

This variable represents the name of the constant.

public java.lang.String **id**

This variable represents the repository id of the constant.

public java.lang.String **defined\_in**

This variable represents the repository id of the module or interface in which this constant is defined.

public org.omg.CORBA.VersionSpec **version**

This variable represents the constant's version.

public org.omg.CORBA.TypeCode **type**

This variable represents the constant's IDL type.

public org.omg.CORBA.Any **value**

This variable represents the value of this constant.

## Constant Description methods

---

public **ConstantDescription**()

This method is the default constructor for a `ConstantDescription`.

public **ConstantDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, org.omg.CORBA.VersionSpec **version**, org.omg.CORBA.TypeCode **type**, org.omg.CORBA.Any **value**)

This method constructs an `ConstantDescription`, using the supplied parameters.

Parameter	Description
name	The name of this constant.
id	The repository id for this constant.
defined_in	The module or interface in which this constant is defined.
version	The object's version.
type	The constant's IDL type code.
value	The value of this constant.

---

# Contained

---

public interface **Contained** extends org.omg.CORBA.ContainedOperations, org.omg.CORBA.IRObject, org.omg.CORBA.portable.IDLEntity;

The interface is used to represent Interface Repository objects that are, themselves, contained within another Interface Repository object. This interface provides methods for:

- Setting and retrieving the object's name and version.
- Determining the **Container** that contains this object.
- Obtaining the object's absolute name, containing repository, and description.
- Moving an object from one container to another.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer. All interfaces that correspond to the IDL's constructs are inherited from **Contained**.

## IDL definition

---

```
interface Contained: IRObject {
    attribute RepositoryId id;
    attribute Identifier name;
    attribute VersionSpec version;

    readonly attribute Container defined_in;
    readonly attribute ScopedName absolute_name;
    readonly attribute Repository containing_repository;

    struct Description {
        DefinitionKind kind;
        any value;
    };

    Description describe();

    void move(in Container new_container,
             in Identifier new_name,
             in VersionSpec new_version);
};
```

## Contained methods

---

public java.lang.String **absolute\_name()**

This method returns this object's absolute name.

public org.omg.CORBA.Repository **containing\_repository()**

This method returns the repository that contains this object.

```
public org.omg.CORBA.Container defined_in()
```

This method returns the `Container` where this object is defined.

```
public org.omg.CORBA.ContainedPackage.Description describe()
```

This method returns this object's description.

```
public java.lang.String id()
```

This method returns this object's repository identifier.

```
public void id(java.lang.String id)
```

This method sets the repository identifier that uniquely identifies this object.

Parameter	Description
id	The repository identifier for this object.

```
public java.lang.string name()
```

This method returns this object's name.

```
public void name(java.lang.String name)
```

This method sets the name for this object.

Parameter	Description
name	The object's name.

```
public java.lang.String version()
```

This method returns this object's version.

```
public void version(java.lang.String version)
```

This method sets the version for this object.

Parameter	Description
version	The object's version.

```
public void move(org.omg.CORBA.Container new_container, String new_name,  
java.lang.String new_version)
```

This method moves this object to another container.

Parameter	Description
new_container	The Container to which the object is to be moved.
new_name	The new name for the object.
new_version	The new version specification for the object.

# ContainedPackage.Description

---

public class **Description**

This class provides a generic description for items in the interface repository that are derived from the Contained interface.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## ContainedPackage.Description variables

---

public org.omg.CORBA.DefinitionKind **kind**

This variable represents kind of the item.

public org.omg.CORBA.Any **value**

This variable represents value of the item.

## ContainedPackage.Description methods

---

public **Description()**

This method is the default constructor for a `Description`.

public **Description**(org.omg.CORBA.DefinitionKind **kind**, org.omg.CORBA.Any **value**)

This method constructs a `Description`, using the supplied parameters.

Parameter	Description
kind	This item’s kind. See “ContainerPackage.Description” on page 7-19 for more information.
value	An Any object that represents the value for this item.

# Container

---

public interface **Container** extends org.omg.CORBA.ContainerOperations, org.omg.CORBA.IRObject, org.omg.CORBA.portable.IDLEntity;

The `Container` interface is used to create a containment hierarchy in the Interface Repository. A `Container` object holds object definitions derived from the `Contained` class. All object definitions derived from the `Container` class, with the exception of the `Repository` class, also inherit from the `Contained` class.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.



## IDL definition

---

```

interface Container: IRObjct {
    Contained lookup(in ScopedName search_name);
    ContainedSeq contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited);
    ContainedSeq lookup_name(
        in Identifier search_name,
        in long levels_to_search,
        in DefinitionKind limit_type,
        in boolean exclude_inherited
    );
    struct Description {
        Contained contained_object;
        DefinitionKind kind;
        any value;
    };
    typedef sequence<Description> DescriptionSeq;

    DescriptionSeq describe_contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited,
        in long max_returned_objs);
    ModuleDef create_module,
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version);

    ConstantDef create_constant(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType type,
        in any value);
    StructDef create_struct(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in StructMemberSeq members);

    NativeDef create_native(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version);

    UnionDef create_union(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType discriminator_type,
        in UnionMemberSeq members);

```

```
EnumDef create_enum(  
    in RepositoryId id,  
    in Identifier name,  
    in VersionSpec version,  
    in EnumMemberSeq members);  
  
AliasDef create_alias(  
    in RepositoryId id,  
    in Identifier name,  
    in VersionSpec version,  
    in IDLType original_type);  
  
ExceptionDef create_exception(  
    in RepositoryId id,  
    in Identifier name,  
    in VersionSpec version,  
    in StructMemberSeq members);  
  
InterfaceDef create_interface(  
    in RepositoryId id,  
    in Identifier name,  
    in VersionSpec version,  
    in InterfaceDefSeq base_interfaces  
    in boolean is_abstract);  
  
ValueDef create_value(  
    in RepositoryId id,  
    in Identifier name,  
    in VersionSpec version  
    in boolean is_custom,  
    in boolean is_abstract,  
    in ValueDef base_value,  
    in boolean is_truncatable,  
    in ValueDefSeq abstract_base_values,  
    in InterfaceDefSeq supported_interfaces,  
    in InitializerSeq initializers);  
};
```

## Container methods

---

```
public org.omg.CORBA.Contained[] contents(org.omg.CORBA.DefinitionKind limit_type,  
boolean exclude_inherited)
```

This method sets the repository identification that uniquely identifies this object.

Parameter	Description
limit_type	The interface object types to be returned.
exclude_inherited	If set to true, inherited objects will not be returned.

```
public org.omg.CORBA.InterfaceDef create_abstract_interface(java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.AbstractionInterfaceDef[] base_interfaces)
```

This method creates an `AbstractInterfaceDef` object in this `Container` with the specified attributes and returns a reference to the newly created object. The interface that is created can only definitions of abstract interfaces.

Parameter	Description
<code>id</code>	The interface's repository id.
<code>name</code>	The interface's name.
<code>version</code>	The interface's version.
<code>base_interfaces</code>	A list of all interfaces from which this interface inherits.

```
public org.omg.CORBA.AliasDef create_alias(java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.IDLType original_type)
```

This method creates an `AliasDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The alias' repository id.
<code>name</code>	The alias' name.
<code>version</code>	The alias' version.
<code>original_type</code>	The IDL type of the original object for which this is an alias.

```
public org.omg.CORBA.ConstantDef create_constant(java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.IDLType type, org.omg.CORBA.Any value)
```

This method creates a `ConstantDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The constant's repository id.
<code>name</code>	The constant's name.
<code>version</code>	The constant's version.
<code>type</code>	The constant's IDL type.
<code>value</code>	The constant's value, represented by an <code>Any</code> object.

```
public org.omg.CORBA.EnumDef create_enum(java.lang.String id, java.lang.String name, java.lang.String version, java.lang.String members[])
```

This method creates an `EnumDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The enumeration's repository id.
<code>name</code>	The enumeration's name.

Parameter	Description
version	The enumeration's version.
members	A list of the enumeration's values.

```
public org.omg.CORBA.ExceptionDef create_exception(java.lang.String id, java.lang.String name,
java.lang.String version, org.omg.CORBA.StructMember[] members)
```

This method creates an `ExceptionDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
id	The exception's repository id.
name	The exception's name.
version	The exception's version.
members	A list of all the types of the members of the exception, if any.

```
public org.omg.CORBA.InterfaceDef create_interface(java.lang.String id, java.lang.String name,
java.lang.String version, org.omg.CORBA.InterfaceDef[] base_interfaces)
```

This method creates a concrete `InterfaceDef` object in this `Container` with the specified attributes and returns a reference to the newly created object. Unlike an `AbstractInterfaceDef`, this interface can contain definitions of both abstract and concrete interfaces.

Parameter	Description
id	The interface's repository id.
name	The interface's name.
version	The interface's version.
base_interfaces	A list of all interfaces from which this interface inherits.

```
public org.omg.CORBA.ModuleDef create_module(java.lang.String id, java.lang.String name,
java.lang.String version)
```

This method creates a `ModuleDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
id	The module's repository id.
name	The module's name.
version	The module's version.

```
public org.omg.CORBA.NativeDef create_native(java.lang.String id, java.lang.String name,
java.lang.String version)
```

This method creates a `NativeDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The structure's repository id.
<code>name</code>	The structure's name.
<code>version</code>	The structure's version.

```
public org.omg.CORBA.StructDef create_struct(java.lang.String id, java.lang.String name,
java.lang.String version, org.omg.CORBA.StructMember members[])
```

This method creates a `StructDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The structure's repository id.
<code>name</code>	The structure's name.
<code>version</code>	The structure's version.
<code>members</code>	The values for the structure's fields.

```
public org.omg.CORBA.UnionDef create_union(java.lang.String id, java.lang.String name,
java.lang.String version, org.omg.CORBA.IDLType discriminator_type,
org.omg.CORBA.UnionMember[] members)
```

This method creates a `UnionDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The union's repository id.
<code>name</code>	The union's name.
<code>version</code>	The union's version.
<code>discriminator_type</code>	The IDL type of the union's discriminant value.
<code>members</code>	A list of the types of each of the union's fields.

```
public org.omg.CORBA.ContainerPackage.Description[] describe_contents(
org.omg.CORBA.DefinitionKind limit_type, boolean exclude_inherited, int max_returned_objs)
```

This method returns a description for all definitions directly contained by, or inherited into this container.

Parameter	Description
<code>limit_type</code>	The interface object types to be returned.

Parameter	Description
<code>exclude_inherited</code>	If set to true, inherited objects will not be returned.
<code>max_returned_objs</code>	The maximum number of object to be returned. Setting this parameter to -1 will return all objects.

```
public org.omg.CORBA.Contained lookup(java.lang.String search_name)
```

This method locates a definition relative to this container, given a scoped name. An absolute scoped name, one beginning with “::”, may be specified to locate a definition within the enclosing repository. If no object is found, a `NULL` value is returned.

Parameter	Description
<code>search_name</code>	The name of the object to be located.

```
public org.omg.CORBA.Contained[] lookup_name(java.lang.String search_name, int levels_to_search, org.omg.CORBA.DefinitionKind limit_type, boolean exclude_inherited)
```

This method locates an object by name within a particular object. The search can be constrained by the number of levels in the hierarchy to be searched, the type of object, and whether or not inherited objects should be returned.

Parameter	Description
<code>search_name</code>	The name of the object or objects to be located.
<code>levels_to_search</code>	The number of levels in the hierarchy to search. Setting this parameter to a value of -1 will cause all levels to be searched. Setting this parameter to 1 will search only this object.
<code>limit_type</code>	The interface object types to be returned.
<code>exclude_inherited</code>	If set to true, inherited objects will not be returned.

```
public org.omg.CORBA.ValueDef create_value(java.lang.String id, java.lang.String name, java.lang.String version, boolean is_custom, boolean is_abstract, org.omg.CORBA.ValueDef base_value, boolean is_truncatable, org.omg.CORBA.ValueDef[] abstract_base_values, org.omg.CORBA.InterfaceDef supported_interfaces, org.omg.CORBA.Initializer)
```

This method creates a `ValueDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
<code>id</code>	The structure’s repository id.
<code>name</code>	The structure’s name.
<code>version</code>	The structure’s version.
<code>is_custom</code>	If set to true, creates a custom value type.
<code>is_abstract</code>	If set to true, creates and abstract value type.
<code>base_value</code>	The base value definition.
<code>is_truncatable</code>	If set to true, creates a truncatable interface.
<code>abstract_base_values</code>	The array of the abstract base definitions.

Parameter	Description
supported_interfaces	The array of the supported interface definitions.
initializer	The list of initializers this value type supports

public org.omg.CORBA.ValueBoxDef **create\_value\_box**(java.lang.String **id**, java.lang.String **name**, java.lang.String **version**, org.omg.CORBA.IDLType **original\_type**)

This method creates a `ValueBoxDef` object in this `Container` with the specified attributes and returns a reference to the newly created object.

Parameter	Description
id	The structure's repository id.
name	The structure's name.
version	The structure's version.
original_type	The IDL type of the original object for which this is an alias.

## ContainerPackage.Description

public final class **Description**

This class provides a generic description for items in the interface repository that are derived from the `Contained` interface.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### ContainerPackage.Description variables

public org.omg.CORBA.Contained **contained\_object**

The contained item.

public org.omg.CORBA.DefinitionKind **kind**

The kind of this item.

public org.omg.CORBA.Any **value**

The value of the item.

### ContainerPackage.Description methods

public **Description**()

This method is the default constructor for a `Description`.

```
public Description(org.omg.CORBA.Contained contained_object,  
org.omg.CORBA.DefinitionKind kind, org.omg.CORBA.Any value)
```

This method constructs a `Description` using the supplied parameters.

Parameter	Description
contained_object	The contained item.
kind	This item’s kind.
value	An Any object that represents the value for this item.

## DefinitionKind

```
public class DefinitionKind
```

The `DefinitionKind` class enumerates the types of objects a Repository can contain. Each value in the enumeration mirrors a similar data type in IDL.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

### DefinitionKind methods

```
public int value()
```

This method returns the integer value of the `DefinitionKind`.

### DefinitionKind enumeration values

**Table 7.1** DefinitionKind constant values

Java Object	Integer Constant	Value
<code>dk_AbstractInterface</code>	<code>_dk_AbstractInterface</code>	Abstract Interface
<code>dk_all</code>	<code>_dk_all</code>	All possible types (used in repository lookup methods)
<code>dk_Alias</code>	<code>_dk_Alias</code>	Alias
<code>dk_Array</code>	<code>_dk_Array</code>	Array
<code>dk_Attribute</code>	<code>_dk_Attribute</code>	Attribute
<code>dk_Constant</code>	<code>_dk_Constant</code>	Constant
<code>dk_Enum</code>	<code>_dk_Enum</code>	Enum
<code>dk_Exception</code>	<code>_dk_Exception</code>	Exception
<code>dk_Fixed</code>	<code>_dk_Fixed</code>	Fixed
<code>dk_Interface</code>	<code>_dk_Interface</code>	Concrete Interface
<code>dk_Module</code>	<code>_dk_Module</code>	Module
<code>dk_Native</code>	<code>_dk_Native</code>	Native
<code>dk_none</code>	<code>_dk_none</code>	Exclude all types (used in repository lookup methods)
<code>dk_Operation</code>	<code>_dk_Operation</code>	Interface Operation



**Table 7.1** DefinitionKind constant values (continued)

Java Object	Integer Constant	Value
dk_Primitive	_dk_Primitive	Primitive type (such as int or long)
dk_Repository	_dk_Repository	Repository
dk_Sequence	_dk_Sequence	Sequence
dk_String	_dk_String	String
dk_Struct	_dk_Struct	Struct
dk_Typedef	_dk_Typedef	Typedef
dk_Union	_dk_Union	Union
dk_Value	_dk_Value	Value
dk_ValueBox	_dk_ValueBox	ValueBox
dk_ValueMember	_dk_ValueMember	ValueMember
dk_Wstring	_dk_Wstring	Unicode string

## EnumDef

public interface **EnumDef** extends org.omg.CORBA.EnumDefOperations, org.omg.IDLType, org.omg.CORBA.portable.IDLEntity

The interface is used to represent an enumeration that is stored in the Interface Repository. This interface provides methods for setting and retrieving the enumeration's list of members.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### EnumDef methods

public java.lang.String[] **members**()

This method returns the enumeration's list of members.

public void **members**(java.lang.String **members**[])

This method sets the enumeration's list of members.

Parameter	Description
members	The list of members.

## ExceptionDef

public interface **ExceptionDef** extends org.omg.CORBA.ExceptionDefOperations, org.omg.CORBAContained, org.omg.CORBA.Container, org.omg.CORBA.portable.IDLEntity

The interface is used to represent an exception that is stored in the Interface Repository. This interface provides methods for setting and retrieving the

exception's list of members as well as a method for retrieving the exception's `TypeCode`.  
Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## ExceptionDef methods

---

public org.omg.CORBA.StructMember[] **members**()

This method returns this exception's list of members.

public void **members**(org.omg.CORBA.StructMember **members**[])

This method sets the exception's list of members.

Parameter	Description
members	The list of members.

public org.omg.CORBA.TypeCode **type**()

This method returns the `TypeCode` that represents this exception's type.

## ExceptionDescription

---

public final class **ExceptionDescription**

The `ExceptionDescription` class describes an exception that is stored in the interface repository.  
Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## ExceptionDescription variables

---

public java.lang.String **name**

The name of the exception.

public java.lang.String **id**

The repository id of the exception.

public java.lang.String **defined\_in**

The repository id of the module or interface in which this exception is defined.

public org.omg.CORBA.VersionSpec **version**

The exception's version.

public org.omg.CORBA.TypeCode **type**

The exception's IDL type.

## ExceptionDescription methods

---

public **ExceptionDescription**()

This method is the default constructor for an `ExceptionDescription`.

public **ExceptionDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, org.omg.CORBA.VersionSpec **version**, org.omg.CORBA.TypeCode **type**, org.omg.CORBA.Any **value**)

This method constructs an `ExceptionDescription`, using the supplied parameters.

Parameter	Description
<code>name</code>	The name of this exception.
<code>id</code>	The repository id for this exception.
<code>defined_in</code>	The module or interface in which this exception is defined.
<code>version</code>	The object's version.
<code>type</code>	The exception's IDL type code.

---

## FixedDef

---

public interface **FixedDef** extends org.omg.CORBA.FixedDefOperations, org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity

This interface is used to represent a fixed definition that is stored in the Interface Repository.

## FixedDef methods

---

public **short digits**();

This method returns the number of digits in the fixed type.

public void **digits** (short digits);

This method sets the number of digits in this fixed type.

public **short scale** ();

The *scale* is the number of digits to the right of the decimal point in the fixed type. This method returns the scale of the fixed type.

public void **scale** (short scale);

This method sets the scale of the fixed type.

# FullValueDescription

---

public final class **FullValueDescription**

This class is used to represent a full value definition that is stored in the Interface Repository. See “ValueDef” on page 7-46 for additional information.

## FullValueDescription variables

---

public java.lang.String **name**

The name of the value type.

public java.lang.String **id**

The repository id of the value type.

public boolean **is\_abstract**

If `true`, this value type is abstract. If `false`, this is a concrete value type.

public boolean **is\_custom**

If `true`, there is custom marshalling for the value type.

public java.lang.String **defined\_in**

The repository id of the module in which this value type is defined.

public java.lang.String **version**

The value type’s version.

public org.omg.CORBA.OperationDescription[] **operations**

The list of operations offered by the value type.

public org.omg.CORBA.AttributeDescription[] **attributes**

The value type’s list of attributes.

public org.omg.CORBA.ValueMember[] **members**

The array of the value type’s members.

public org.omg.CORBA.Initializer[] **initializers**

The array of initializers.

public java.lang.String[] **supported\_interfaces**

The list of interfaces supported by this value type.

public java.lang.String[] **abstract\_base\_values**

The list of reporting IDs of all the abstract base values from which this value type inherits.

public boolean **is\_truncatable**

If `true`, the value can safely be truncated to its base value type.

public java.lang.String **base\_value**

The reporting ID for the concrete base value if one exists for this type.

public org.omg.CORBA.TypeCode **type**

The value type's IDL typecode.

## FullValueDescription methods

---

```
public org.omg.CORBA.FullValueDescription(java.lang.String name, java.lang.String id,
    boolean is_abstract, boolean is_custom, java.lang.String defined_in,
    java.lang.String version, org.omg.CORBA.OperationDescription[] operations,
    org.omg.CORBA.AttributeDescription[] attributes, org.omg.CORBA.ValueMember[] members,
    org.omg.CORBA.Initializer[] initializers, java.lang.String[] supported_interface,
    java.lang.String[] abstract_base_values, boolean is_truncatable, java.lang.String base_value,
    org.omg.CORBA.TypeCode type)
```

This method constructs the FullValueDescription.

Parameter	Description
<b>id</b>	The value type's identifier.
<b>name</b>	The value type's name.
<b>is_abstract</b>	The value type is abstract.
<b>is_custom</b>	The value type is custom.
<b>version</b>	The value type's version.
<b>operations</b>	The list of operations.
<b>attributes</b>	The list of attributes.
<b>members</b>	The list of value member descriptors.
<b>initializers</b>	The list of initializers.
<b>supported_interfaces</b>	The list of supported interfaces.
<b>abstract_base_values</b>	The list of abstract base value types.
<b>is_truncatable</b>	The value type is truncatable.
<b>base_value</b>	The real (non-abstract) base value, if any.
<b>type</b>	The value type's typecode.

---

## IDLType

---

public interface **IDLType** extends org.omg.CORBA.IRObject

The **IDLType** interface is defined as a marker interface used to mark those Repository Objects that form IDL types. For example, **EnumDef** inherits from **IDLType** because it represents the IDL type construct, but **OperationDef** does not, because it is the regular Repository object and not IDL type. Every **IDLType** object has an associated **TypeCode**. The **TypeCode** uniquely identifies the object's type and is described in the section, "TypeCode" on page 6-31.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
interface IDLType : CORBA::IObject {
    readonly attribute TypeCode type;
};
```

## IDLType methods

---

```
public org.omg.CORBA.TypeCode type()
```

This method returns the TypeCode object associated with the IObject that represents an IDL type definition in the Interface Repository.

## InterfaceDef

---

```
public interface InterfaceDef extends org.omg.CORBA.InterfaceDefOperations,
    org.omg.CORBA.Container, org.omg.CORBA.Contained,
    org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity
```

The interface is used to represent the concrete interface that is stored in the Interface Repository. This interface provides methods for setting and retrieving the base interfaces as well as creating attributes, operations and an interface description.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
interface InterfaceDef: Container, Contained, IDLType {
    attribute InterfaceDefSeq base_interfaces;
    boolean is_a(in RepositoryId interface_id);

    struct FullInterfaceDescription {
        Identifier name;
        RepositoryId id;
        RepositoryId defined_in;
        VersionSpec version;
        OpDescriptionSeq operations;
        AttrDescriptionSeq attributes;
        RepositoryIdSeq base_interfaces;
        TypeCode type;
    };
    FullInterfaceDescription describe_interface();
```

```

AttributeDef create_attribute(
    in RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType type,
    in AttributeMode mode);
OperationDef create_operation(
    in CORBA::RepositoryId id,
    in Identifier name,
    in VersionSpec version,
    in IDLType result,
    in OperationMode mode,
    in ParDescriptionSeq params,
    in ExceptionDefSeq exceptions,
    in ContextIdSeq contexts);
};

```

## InterfaceDef methods

---

public org.omg.CORBA.InterfaceDef[] **base\_interfaces**()

This method returns the base interface list for this object.

public void **base\_interfaces**(org.omg.CORBA.InterfaceDef[] **base\_interfaces**)

This method sets the base interface list for this object.

Parameter	Description
base_interfaces	The list of base interfaces to be set.

---

public org.omg.CORBA.AttributeDef **create\_attribute**(java.lang.String **id**, java.lang.String **name**, java.lang.String **version**, org.omg.CORBA.IDLType **type**, org.omg.CORBA.AttributeMode **mode**)

This method adds an attribute to an interface definition.

Parameter	Description
id	The attribute's identifier.
name	The attribute's name.
version	The attribute's version.
type	The attribute's IDL type.
mode	The attribute's mode. See AttributeMode in the section, "AttributeMode" on page 7-7, for possible values.

---

public org.omg.CORBA.OperationDef **create\_operation**(java.lang.String **id**, java.lang.String **name**, java.lang.String **version**, org.omg.CORBA.IDLType **result**,

org.omg.CORBA.OperationMode **mode**, org.omg.CORBA.ParameterDescription[] **params**,  
org.omg.CORBA.ExceptionDef[] **exceptions**, java.lang.String[] **contexts**)

This method adds an operation to an interface definition.

Parameter	Description
id	The operation’s identifier.
name	The operation’s name.
version	The operation’s version.
result	The operation’s IDL result type.
mode	The operation’s mode. See “OperationMode” on page 7-35.
params	The list of parameters for this operation.
exceptions	The list of exceptions that can be raised by this operation.
contexts	The list of contexts.

public org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription **describe\_interface()**

This method returns an interface description for this object.

public boolean **is\_a**(java.lang.String **interface\_id**)

This method returns **true** if this object represents an interface definition that is compatible with a given **interface\_id**.

Parameter	Description
interface_id	The interface identifier to compare with this object.

## InterfaceDefPackage.FullInterfaceDescription

public final class **FullInterfaceDescription**

This class provides a full description of an interface that is stored in the interface repository.

### InterfaceDefPackage.FullInterfaceDescription variables

public java.lang.String **name**

The name of the interface.

public java.lang.String **id**

The repository id of the interface.

public java.lang.String **defined\_in**

The repository id of the module in which this interface is defined.

public java.lang.String **version**

The interface’s version.



public org.omg.CORBA.OperationDescription[] **operations**

The list of operations supported by the interface.

public org.omg.CORBA.AttributeDescription[] **attributes**

The interface's list of attributes.

public java.lang.String[] **base\_interfaces**

The list of base interfaces from which this interface inherits.

public org.omg.CORBA.TypeCode **type**

This variable represents the interface's IDL typecode.

## InterfaceDefPackage.FullInterfaceDescription methods

---

public **FullInterfaceDescription**()

This method is the default constructor for a `FullInterfaceDescription`.

public **FullInterfaceDescription**(final java.lang.String **name**, final java.lang.String **id**,  
final java.lang.String **defined\_in**, final java.lang.String **version**,  
final org.omg.CORBA.OperationDescription[] **operations**,  
final org.omg.CORBA.AttributeDescription[] **attributes**, final java.lang.String[] **base\_interfaces**,  
final org.omg.CORBA.TypeCode **type**)

This method constructs an `FullInterfaceDescription`, using the supplied parameters.

Parameter	Description
<code>name</code>	The name of this interface.
<code>id</code>	The repository id for this interface.
<code>defined_in</code>	The module's repository id in which this attribute is defined.
<code>version</code>	The interface's version.
<code>operations</code>	The list of operation offered by this interface.
<code>attributes</code>	The list of this interface's attributes.
<code>base_interfaces</code>	This variable represents a list of base interfaces for this interface.
<code>type</code>	The interface's IDL type code.

---

## InterfaceDescription

---

public final class **InterfaceDescription**

The `InterfaceDescription` class provides a description of the interface that is stored in the interface repository.

Helper and Holder versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## InterfaceDescription variables

---

public java.lang.String **Name**

The name of the interface.

public java.lang.String **id**

The repository id of the interface.

public java.lang.String **defined\_in**

The repository id of the module in which this interface is defined.

public java.lang.String **version**

The interface’s version.

public String[] **base\_interfaces**

The list of base interfaces for this interface.

## InterfaceDescription methods

---

public **InterfaceDescription**()

This method is the default constructor for an `InterfaceDescription`.

public **InterfaceDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, java.lang.String **version**, java.lang.String[] **base\_interfaces**)

This method constructs an `InterfaceDescription`, using the supplied parameters.

Parameter	Description
name	The name of this interface.
id	The repository id for this interface.
defined_in	The module in which this interface is defined.
version	The interface’s version.
base_interfaces	The interface’s list of base interfaces.

## IObject

---

public interface **IObject**

The `IObject` interface offers a generic interface to any object stored in the Interface Repository.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
interface IRObjct {
    readonly attribute CORBA::DefinitionKind def_kind;
    void destroy();
};
```

## IRObjct methods

---

public org.omg.CORBA.DefinitionKind **def\_kind()**

This method returns the kind of IDL definition that this `IRObjct` represents. For a list of defined types, see “DefinitionKind” on page 7-20.

public void **destroy()**

This method deletes this `IRObjct` from the Interface Repository.

## ModuleDef

---

public interface **ModuleDef** extends org.omg.ModuleDefOperations, org.omg.CORBA.Container, org.omg.CORBA.Contained, org.omg.CORBA.portable.IDLEntity

The interface is used to represent an IDL module in the interface repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## ModuleDescription

---

public final class **ModuleDescription**

The `ModuleDescription` class describes a module that is stored in the interface repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## ModuleDescription variables

---

public java.lang.String **name**

The name of the module.

public java.lang.String **id**

The repository id of the module.

public java.lang.String **defined\_in**

The repository id of the module in which this module is defined.

public java.lang.String **version**

The module's version.

## ModuleDescription methods

---

public **ModuleDescription**()

This method is the default constructor for the `ModuleDescription`.

public **ModuleDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, java.lang.String **version**)

This method constructs a `ModuleDescription` using the supplied parameters.

Parameter	Description
name	The name of this interface.
id	The repository id for this interface.
defined_in	The module id in which this module is defined.
version	The object's version.

---

## NativeDef

---

public interface **NativeDef** extends org.omg.CORBA.NativeDefOperations, org.omg.CORBA.TypedDef, org.omg.CORBA.portable.IDLEntity

This interface is used to represent a native definition that is stored in the Interface Repository. The `Container` interface provides operations to create `NativeDef` as a contained object.

## OperationDef

---

public interface **OperationDef** extends org.omg.CORBA.OperationDefOperations, org.omg.CORBA.Contained, org.omg.CORBA.portable.IDLEntity

The interface is used to represent an interface operation that is stored in the Interface Repository. This interface provides methods for setting and retrieving the operation's contexts, mode, parameters, and result value. A method is also provided for retrieving a list of exceptions that may be raised by this operation.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

## OperationDef methods

---

public java.lang.String[] **contexts**()

This method returns the contexts associated with this operation.

public void **contexts**(java.lang.String[] **contexts**)

This method sets this operation's context list.

Parameter	Description
contexts	The list of contexts.

---

public org.omg.CORBA.ExceptionDef[] **exceptions**()

This method returns a list of exceptions that may be raised by this operation.

public void **exceptions**(org.omg.CORBA.ExceptionDef[] **exceptions**)

This method sets the list of exceptions that may be raised by this operation.

Parameter	Description
exceptions	The list of exceptions.

---

public org.omg.CORBA.OperationMode **mode**()

This method returns the mode of this operation.

public void **mode**(org.omg.CORBA.OperationMode **mode**)

This method sets the mode of this operation.

Parameter	Description
mode	The mode to be set. See "OperationMode" on page 7-35 for more details on the parameters.

---

public org.omg.CORBA.ParameterDescription[] **params**()

This method returns a description of the parameters for this operation.

public void **params**(org.omg.CORBA.ParameterDescription[] **params**)

This method sets the parameter description for this operation.

Parameter	Description
params	The description of the parameters.

---

public org.omg.CORBA.TypeCode **result**()

This method returns the `TypeCode` of the result returned by this operation.

public org.omg.CORBA.IDLType **result\_def**()

This method returns the IDL type of this operation's return value.

```
public void result_def(org.omg.CORBA.IDLType result_def)
```

This method sets the IDL type for this operation’s return value.

Parameter	Description
result_def	The IDL type to set for the return value.

# OperationDescription

```
public final class OperationDescription
```

The `OperationDescription` class is stored in the interface repository.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## OperationDescription variables

```
public java.lang.String name
```

The name of the operation.

```
public java.lang.String id
```

The repository id of the operation.

```
public java.lang.String defined_in
```

The repository id of the interface or value type in which this operation is defined.

```
java.lang.String version
```

The operation’s version.

```
public org.omg.CORBA.TypeCode result
```

The operation’s result `TypeCode`.

```
public org.omg.CORBA.OperationMode mode
```

The operation’s mode.

```
public java.lang.String[] contexts
```

The operation’s associated context list.

```
public org.omg.CORBA.ParameterDescription[] parameters
```

The operation’s parameters.

```
public org.omg.CORBA.ExceptionDescription[] exceptions
```

The exceptions that this operation may throw.

## OperationDescription methods

---

public **OperationDescription**()

This method is the default constructor for an `OperationDescription`.

public **OperationDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, java.lang.String **version**, org.omg.CORBA.TypeCode **result**, org.omg.CORBA.OperationMode **mode**, java.lang.String[] **contexts**, org.omg.CORBA.ParameterDescriptions **parameters**, org.omg.CORBA.ExceptionDescription[] **exceptions**)

This method constructs an `OperationDescription`, using the supplied parameters.

Parameter	Description
name	The name of this operation.
id	The repository id for this operation.
defined_in	The interface or value type id in which this operation is defined.
version	The object's version.
result	The IDL typecode of the result of the operation.
mode	The operation's mode.
contexts	A list of context strings for this operation.
parameters	The list of parameters for this operation.
exceptions	The list of exceptions that this operation may throw.

---

## OperationMode

---

public final class **OperationMode**

This class enumerates the *modes* of operation. The two modes are:

- org.omg.CORBA.OperationMode.OP\_ONEWAY    Oneway operations are those for which the client application does not expect a response.
- org.omg.CORBA.OperationMode.OP\_NORMAL    Normal requests involve a response being sent to the client by the object implementation that contains the results of the request.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

# ParameterDescription

---

public final class **ParameterDescription**

The `ParameterDescription` class describes a parameter for an operation that is stored in the interface repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## ParameterDescription variables

---

public java.lang.String **name**

The name of the parameter.

public org.omg.CORBA.TypeCode **type**

The parameter’s `TypeCode`.

public org.omg.CORBA.IDLType **type\_def**

The parameter’s IDL type.

public org.omg.CORBA.ParameterMode **mode**

The parameter’s mode. See “`ParameterMode`” on page 7-37 for more details on the parameters.

## ParameterDescription methods

---

public **ParameterDescription**()

This method is the default constructor for a `ParameterDescription`.

public **ParameterDescription**(java.lang.String **name**, org.omg.CORBA.TypeCode **type**, org.omg.CORBA.IDLType **type\_def**, org.omg.CORBA.ParameterMode **mode**)

This method constructs a `ParameterDescription`, using the supplied parameters.

Parameter	Description
name	The name of the parameter.
type	The type of the parameter.
type_def	The IDL type of the parameter.
mode	The mode of the parameter.

---



## ParameterMode

---

public final class **ParameterMode**

This class enumerates the three *modes* of parameters:

PARAM\_IN      Used for input from the client to the server.

PARAM\_OUT     Used for output of results from the server to the client.

PARAM\_INOUT   Used both for input from the client and output from the server.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## PrimitiveDef

---

public interface **PrimitiveDef** extends org.omg.CORBA.PrimitiveDefOperations,  
org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a primitive type (such as an `int` or a `long`) that is stored in the Interface Repository. This interface provides a method for retrieving the kind of primitive that is being represented.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

### PrimitiveDef method

---

public org.omg.CORBA.PrimitiveKind **kind()**

This method returns the kind of primitive represented by this object.

## PrimitiveKind

---

public class **PrimitiveKind**

The `PrimitiveKind` class enumerates the primitive types present in the IDL. It provides both an enumeration (the `PrimitiveKind` enumerations begin with the prefix `_pk_`) and a set of Java objects (the `PrimitiveKind` Java objects begin with the prefix `pk_`).

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## PrimitiveKind methods

---

public int **value()**

This method returns an integer that represents the value of the constant.

## PrimitiveKind constants

---

Java Object	Integer Constant	Represents
pk_any	_pk_any	Any object.
pk_boolean	_pk_boolean	Boolean.
pk_char	_pk_char	Character.
pk_double	_pk_double	Double.
pk_float	_pk_float	Float.
pk_long	_pk_long	Long.
pk_longdouble	_pk_longdouble	Long double.
pk_longlong	_pk_longlong	Long long.
pk_null	_pk_null	Null.
pk_octet	_pk_octet	Octet string.
pk_octet	_pk_octet	Octet string.
pk_objref	_pk_objref	Object reference.
pk_short	_pk_short	Short.
pk_string	_pk_string	String.
pk_TypeCode	_pk_TypeCode	TypeCode object.
pk_ulong	_pk_ulong	Unsigned long.
pk_ulonglong	_pk_ulonglong	Unsigned long.
pk_ushort	_pk_ushort	Unsigned short.
pk_void	_pk_void	Void.
pk_wchar	_pk_wchar	Unicode character.
pk_wstring	_pk_wstring	Unicode string.

## Repository

---

public interface **Repository** extends org.omg.CORBA.RepositoryOperations,  
org.omg.CORBA.Container, org.omg.CORBA.portable.IDLEntity

The **Repository** provides an interface to the Interface Repository itself, which is used to contain the definitions of objects that are available to clients. The **Repository** interface provides methods for storing and retrieving definitions. The **Repository** is the only root object in the Interface Repository containment hierarchy.

**Helper** and **Holder** versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## Repository methods

---

```
public org.omg.CORBA.ArrayDef create_array(int length, org.omg.CORBA.IDLType element_type)
```

This method creates an array definition in the repository with the specified length and element type. A reference to the `ArrayDef` that is created is returned.

Parameter	Description
<code>length</code>	The number of elements in the array. This value must be greater than zero.
<code>element_type</code>	The IDL type of the elements contained in the array.

---

```
public org.omg.CORBA.SequenceDef create_sequence(int bound,  
org.omg.CORBA.IDLType element_type)
```

This method creates an array definition in the repository with the specified number of elements (`bound`) and element type. It returns a reference to the `SequenceDef` that is created.

Parameter	Description
<code>bound</code>	The maximum length of the sequence. This value must be zero or greater.
<code>element_type</code>	The IDL type of the elements contained in the sequence.

---

```
public org.omg.CORBA.StringDef create_string(int bound)
```

This method creates a string definition in the repository with the specified number of characters (`bound`). Returns a reference to the `StringDef` that is created.

Parameter	Description
<code>bound</code>	The maximum bounds of the string. This value must be zero or greater.

---

```
public org.omg.CORBA.WstringDef create_wstring(int bound)
```

This method creates a Unicode string definition in the repository with the specified number of characters (`bound`). A reference to the `WstringDef` that is created is returned.

Parameter	Description
<code>bound</code>	The maximum bounds of the string. This value must be zero or greater.

---

public org.omg.CORBA.PrimitiveDef **get\_primitive**(org.omg.CORBA.PrimitiveKind **kind**)

This method returns a `PrimitiveDef` object for the specified `PrimitiveKind`.

Parameter	Description
kind	The primitive type kind.

public org.omg.CORBA.Contained **lookup\_id**(java.lang.String **search\_id**)

This method searches for an object in the interface repository that matches the specified search id. If a match is not found, a null value is returned.

Parameter	Description
search_id	The identifier to use for the search.

public org.omg.CORBA.FixedDef **create\_fixed**(short **digits**, short **scale**)

This method sets the number of digits and the scale for the fixed type.

Parameter	Description
digits	The number of digits for the fixed type.
scale	The scale of the fixed type.

## SequenceDef

public interface **SequenceDef** extends org.omg.CORBA.SequenceDefOperations,  
org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a sequence that is stored in the Interface Repository. This interface provides methods for setting and retrieving the sequence's bound and element type.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### SequenceDef methods

public int **bound**()

This method returns the bounds of the sequence.

public void **bound**(int **bound**)

This method sets the bound of the sequence.

Parameter	Description
members	The list of members.

```
public org.omg.CORBA.TypeCode element_type()
```

This method returns a `TypeCode` representing the type of elements in this sequence.

```
public org.omg.CORBA.IDLType element_type_def()
```

This method returns the IDL type of the elements stored in this sequence.

```
public void element_type_def(org.omg.CORBA.IDLType element_type_def)
```

This method sets the IDL type for the elements stored in this sequence.

Parameter	Description
<code>element_type_def</code>	The IDL type to set.

## StringDef

---

```
public interface StringDef extends org.omg.CORBA.StringDefOpeartions, org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

The interface is used to represent a `String` that is stored in the Interface Repository. This interface provides methods for setting and retrieving the bounds of the string.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

### StringDef methods

---

```
public int bound()
```

This method returns the bounds (maximum length) of the `String`. If the `String` is unbounded, it returns 0.

```
public void bound(int bound)
```

This method sets the bounds of the `String`. Pass 0 to set the bounds to unbounded.

Parameter	Description
<code>bound</code>	The new <code>String</code> bounds.

# StructDef

---

public interface **StructDef** extends org.omg.CORBA.StructDefOperations, org.omg.CORBA.TypeDefDef, org.omg.CORBA.Container, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a structure that is stored in the Interface Repository. This interface provides methods for setting and retrieving the structure’s list of members.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## StructDef methods

---

public org.omg.CORBA.StructMember[] **members**()

This method returns the structures’s list of members.

public void **members**(org.omg.CORBA.StructMember[] **members**)

This method sets the structure’s list of members.

Parameter	Description
members	The list of members.

---

# StructMember

---

public final class **StructMember**

This interface is used to define each field of a struct.

## StructMember variables

---

public java.lang.String **name**

The name of the StructMember.

public org.omg.CORBA.TypeCode **type**

The StructMember’s IDL type.

public org.omg.CORBA.IDLType **type\_def**

The StructMember’s IDL type definition.

## StructMember methods

---

```
public StructMember final java.lang.String name, final org.omg.CORBA.TypeCode type, final
    org.omg.CORBA.IDLType type_def
```

This method constructs a `TypeDescription`, using the supplied parameters.

Parameter	Description
<code>name</code>	The name of this <code>StructMember</code> .
<code>type</code>	The <code>StructMember</code> 's IDL type code.
<code>type_def</code>	The <code>StructMember</code> 's type definition.

---

## TypedefDef

---

```
public interface TypedefDef extends org.omg.CORBA.TypedefDefOperations,
    org.omg.CORBA.Contained, org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity
```

This abstract interface represents a user-defined structure that is stored in the Interface Repository. The following interfaces all inherit from this interface:

- `AliasDef`
- `EnumDef`
- `NativeDef`
- `StructDef`
- `UnionDef`
- `WstringDef`

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## TypeDescription

---

```
public final class TypeDescription
```

The `TypeDescription` class is stored in the interface repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## TypeDescription variables

---

```
public java.lang.String name
```

The name of the type.

```
public java.lang.String id
```

The repository id of the type.

TypeDescription methods

public java.lang.String **defined\_in**

The name of the module or interface in which this type is defined.

public java.lang.String **version**

The type’s version.

public org.omg.CORBA.TypeCode **type**

The type’s IDL type.

TypeDescription methods

---

public **TypeDescription**()

This method is the default constructor for a `TypeDescription`.

public **TypeDescription**(java.lang.String **name**, java.lang.String **id**, java.lang.String **defined\_in**, java.lang.String **version**, org.omg.CORBA.TypeCode **type**)

This method constructs a `TypeDescription` using the supplied parameters.

Parameter	Description
name	The name of this type.
id	The repository id for this type.
defined_in	The module or interface in which this type is defined.
version	The object’s version.
type	The type’s IDL type code.

---

UnionDef

---

public interface **UnionDef** extends org.omg.CORBA.UnionDefOperations, org.omg.CORBA.IDLType, org.omg.CORBA.Container, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a `Union` that is stored in the Interface Repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

UnionDef methods

---

public org.omg.CORBA.TypeCode **discriminator\_type**()

This method returns the `TypeCode` of the discriminator for the `Union`.

public org.omg.CORBA.IDLType **discriminator\_type\_def**()

This method returns the IDL type of the union’s discriminator.



```
public void discriminator_type_def(org.omg.CORBA.IDLType discriminator_type_def)
```

This method sets the IDL type of the union's discriminator.

Parameter	Description
<code>discriminator_type_def</code>	The list of members.

```
public org.omg.CORBA.UnionMember[] members()
```

This method returns the union's list of members.

```
public void members(org.omg.CORBA.UnionMembers[] members)
```

This method sets the union's list of members.

Parameter	Description
<code>members</code>	The list of members.

## UnionMember

---

```
public final class UnionMember
```

The `UnionMember` class describes a member of the union that is stored in the interface repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, "Generated interfaces and classes," for more information on these classes and the methods they offer.

### UnionMember variables

---

```
public java.lang.String name
```

The name of the union member.

```
public org.omg.CORBA.Any label
```

The label that is associated with this member.

```
public org.omg.CORBA.TypeCode type
```

The union's typecode.

```
public org.omg.CORBA.IDLType type_def
```

The union's IDL type.

### UnionMember methods

---

```
public UnionMember()
```

This method is the default constructor for an `UnionMember`.

```
public UnionMember(java.lang.String name, org.omg.CORBA.Any label, org.omg.CORBA.TypeCode
type, org.omg.CORBA.IDLType type_def)
```

This method constructs a `UnionMember`, using the supplied parameters.

Parameter	Description
name	The name of this union member.
label	The label that is associated with this member.
type	The union's typecode.
type_def	The union's IDL type.

## ValueBoxDef

---

```
public interface ValueBoxDef extends org.omg.CORBA.ValueBoxDefOperations,
org.omg.CORBA.Contained, org.omg.CORBA.portable.IDLEntity
```

This interface is used as a simple value type that contains a single public member of any IDL type. `ValueBoxDef` is a simplified version of value type:

```
value type name
public <IDLType> value;
```

This declaration is almost equal to value type boxed type `<IDLType>` but `ValueBoxDef` is not the same as simple `ValueDef`.

### ValueBoxDef methods

---

```
public org.omg.CORBA.IDLType original_type_def ();
```

This method identifies the type being boxed.

```
public void original_type_def (org.omg.CORBA.IDLType original_type_def);
```

This method sets the type being boxed.

## ValueDef

---

```
public interface ValueDef extends org.omg.CORBA.ValueDefOperations, org.omg.CORBA.Container,
org.omg.CORBA.Contained, org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity
```

This interface represents a value definition that is stored in the Interface Repository. It can contain constants, typedefs, exceptions, operations, and attributes. This interface is very close to a class type. See "FullValueDescription" on page 7-24 for additional information.

## ValueDef methods

---

```
public org.omg.CORBA.Interface[] supported_interfaces ();
```

This method lists the interfaces which this value type supports.

```
public void supported_interfaces (org.omg.CORBA.interfaceDef[] supported_interfaces);
```

This method sets the supported interfaces.

```
public org.omg.CORBA.Initializer[] initializers;
```

This method lists the initializers

```
public void initializers (org.omg.CORBA.Initializer[] initializers);
```

This method sets the initializers.

```
public org.omg.CORBA.ValueDef base_value;
```

This method describes the value types from which this value inherits.

```
public void base_value (org.omg.CORBA.ValueDef base_value;
```

This method sets a base valuetype of the valuetype which this valuetype inherits from.

```
public org.omg.CORBA.ValueDef[] abstract_base_values ();
```

This method lists the abstract value types from which this value inherits.

```
public void abstract_base_values (org.omg.CORBA.ValueDef[] abstract_base_values);
```

This method defines the list of base abstract value types.

```
public boolean is_abstract();
```

If set to true, this value returns an abstract value type.

```
public void is_abstract (boolean is_abstract);
```

This method sets the value type to an abstract value type.

```
public boolean is_custom ();
```

If set to true, this value uses custom marshalling.

```
public void is_custom (boolean is_custom);
```

This method sets the custom marshalling for the value.

```
public boolean is_truncatable ();
```

If set to true, this value can be safely truncated from its base value.

```
public void is_truncatable (boolean is_truncatable);
```

This method sets the truncation attribute for this value.

```
public boolean is_a (java.lang.String value_id);
```

This method returns true if the value on which it is invoked either is identical to or inherits, directly or indirectly from the interface or value defined by its ID parameter. Otherwise it returns false.

```
public org.omg.CORBA.ValueDefPackage.FullValueDescription describe_value ();
```

This method returns a `FullValueDescription` describing the value including its operations and attributes.

```
public org.omg.CORBA.ValueMemberDef create_value_member (java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.IDLtype type_def, short access);
```

This method returns a new `ValueMemberDef` contained in the `ValueDef` on which it is invoked.

Parameter	Description
id	The repository id for this type.
name	The name of this type.
version	The object's version.
type_def	The value's IDL type. See
short access	The access value.

```
public org.omg.CORBA.AttributeDef create_attribute (java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.IDLtype type, org.omg.CORBA.AttributeMode mode);
```

This method creates a new attribute definition for this value type and returns an `AttributeDef` for it.

Parameter	Description
id	The repository id for this attribute.
name	The name of this type.
version	The object's version.
type	The type's IDL type.
mode	The object's mode.

```
public org.omg.CORBA.OperationDef create_operation (java.lang.String id, java.lang.String name, java.lang.String version, org.omg.CORBA.IDLtype result, org.omg.CORBA.OperationMode mode, org.omg.CORBA.ParameterDescription[] params, org.omg.CORBA.ExceptionDef[] exceptions, java.lang.String[] contexts);
```

This method creates a a new operation for this value type and returns an `OperationDef` for it.

Parameter	Description
id	The repository id of this operation.
name	The name of this type.
version	The object's version.

Parameter	Description
result	The IDLType for the operation.
mode	The object's mode.
params	The list of the operation's parameters.
exceptions	The list of the operation's exceptions.
contexts	The list of the operation's contexts.

## ValueDescription

---

public final interface **ValueDescription**

This interface is used to represent a description of the value type that is stored in the Interface Repository.

### ValueDescription variables

---

public java.lang.String **name**

The name of the valuedescription.

public java.lang.String **id**

The repository id of the valuedescription.

public boolean **is\_abstract**

If set to true, the valuedescription is an abstract value type.

public boolean **is\_custom**

If set to true, the valuedescription is custom marshalled.

public java.lang.String **defined\_in**

The repository id of the module or interface in which this valuedescription is defined.

public java.lang.String **version**

The valuedescription's version.

public java.lang.String[] **supported\_interfaces**

The list of interfaces which this valuedescription supports.

public java.lang.String[] **abstract\_base\_values**

The list of abstract value types from which this valuedescription inherits.

public boolean **is\_truncatable**

The value type's setting for whether or not this valuedescription can safely truncated to its base value types.

```
public java.lang.String base_value
```

The value types from which this valuedescription inherits.

## ValueDescription methods

---

```
public ValueDescription(java.lang.String name, java.lang.String id, boolean is_abstract, boolean is_custom, java.lang.String defined_in, java.lang.String version, java.lang.String supported_interfaces, java.lang.String abstract_base_values, boolean is_truncatable, java.lang.String base_values)
```

This method constructs an `AttributeDescription`, using the supplied parameters.

Parameter	Description
<code>name</code>	The name of this valuedescription.
<code>id</code>	The repository id for this valuedescription.
<code>is_abstract</code>	If the value is true, the valuedescription is abstract.
<code>is_custom</code>	If the value is true, if the valuedescription uses custom marshalling.
<code>defined_in</code>	The module in which this valuedescription is defined.
<code>version</code>	The valuedescription’s version.
<code>supported_interfaces</code>	The supported interfaces.
<code>abstract_base_values</code>	The supported abstract base values.
<code>is_truncatable</code>	If the value is true, if the valuedescription can be safely truncated to its base values.
<code>base_values</code>	The base values.

---

## ValueMemberDef

---

```
public final class ValueMemberDef extends org.omg.CORBA.ValueMemberDefOperations,  
org.omg.CORBA.Contained, org.omg.CORBA.portable.IDLEntity
```

This interface is used to represent a value member definition that is stored in the Interface Repository.

## ValueMemberDef methods

---

```
public org.omg.CORBA.TypeCode type ();
```

This method returns the value member’s IDL type.

```
public org.omg.CORBA.IDLType type_def ();
```

This method represents the definition of the IDL type.

```
public void type_def (org.omg.CORBA.IDLType type_def);
```

This method sets the IDL type for value member.

public short **access**

This method defines the access value for the object.

public void **access** (short access);

This method sets the access value for value member.

## WStringDef

---

public interface **WStringDef** extends org.omg.CORBA.WStringDefOperations, org.omg.CORBA.IDLType, org.omg.CORBA.portable.IDLEntity

The interface is used to represent a Unicode string that is stored in the Interface Repository.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## WStringDef methods

---

public int **bound**()

This method returns the bounds of the `WString`.

public void **bound**(int **bound**)

This method sets the bounds of the `WString`.

Parameter	Description
members	The new <code>wstring</code> bounds.

---





# Activation interfaces and classes

This chapter describes the interfaces and classes in the Activation package including, `ActivationImplDef`, `Activator`, `CreationImplDef`, `ImplementationDef`, and `OAD`—dynamic interfaces and classes in the Activation package. They are used with the Object Activation Daemon (OAD).

## ActivationImplDef

---

valuetype **ActivationImplDef**

The `ActivationImplDef` valuetype provides a set of attributes for an `Activator`.

```
valuetype ActivationImplDef : ImplementationDef {  
    attribute string service_name;  
    attribute ::CORBA::ReferenceData id;  
    attribute ::extension::Activator activator_obj;  
};
```

## ActivationImplDef methods

---

public abstract `Activator` **activator\_obj()**

This method retrieves the object reference of the object implementation under the control of the `Activator`.

public abstract `byte[]` **id()**

This method retrieves the reference data identifier for the implementation.

public abstract `String` **service\_name()**

This method retrieves the implementation's service name.

# Activator

---

public interface **Activator**

When you design your object implementation, you may want to defer the activation of ORB objects until a client requests them. By deferring object activation, you improve performance. If you have thousands of objects on a server, you can save system resources by only activating objects when clients request them. You can defer activation of multiple object implementations with a single `Activator`.

```
interface Activator {
    Object activate(in CORBA::ImplementationDef impl);
    void deactivate(in Object obj, in CORBA::ImplementationDef impl);
};
```

## Activator methods

---

public org.omg.Object **activate**(org.omg.CORBA.ImplementationDef **impl**)

This method is used to activate an object implementation under the control of an `Activator`. When the ORB receives a client request for an object for which the `Activator` is responsible, the ORB invokes the `activate()` method on the `Activator`. In this method, the ORB uniquely identifies the activated object implementation by passing the `Activator` an `ImplementationDef` parameter—from which the implementation can obtain `ref_data`, the unique identifier.

Parameter	Description
<code>impl</code>	The instance of <code>ImplementationDef</code> .

public void **deactivate**(org.omg.CORBA.Object **obj**, org.omg.CORBA.ImplementationDef **impl**);

This method is used to deactivate an object implementation under the control of an `Activator`. In this method, the ORB uniquely identifies the object implementation to deactivate by passing the `Activator` an object reference and an `ImplementationDef` parameter—from which the implementation can obtain `ref_data`, the unique identifier. For an implementation with a large number of objects, you might use `deactivate()` to clean up state data when you have a loaded cache of objects.

Parameter	Description
<code>obj</code>	The object reference of the object to deactivate.
<code>impl</code>	The instance of <code>ImplementationDef</code> .

# CreationImplDef

---

struct **CreationImplDef**

The `CreationImplDef` interface is an IDL struct that provides a set of attributes for a specific object implementation. Methods for querying and setting the values for these attributes are provided in this interface. The attributes are `_args`, `_env`, `id` (for reference data), `object_name`, `_path_name`, `_policy`, and `repository_id`.

The Object Activation Daemon uses this interface to list, register, and unregister object implementations. The command line arguments, specified when you use `oadutil`, are used to set attributes defined in this interface.

## IDL definition

---

```
struct CreationImplDef
{
    CORBA::RepositoryId repository_id;
    string object_name;
    CORBA::ReferenceData id;
    string path_name;
    CORBA::Policy activation_policy;
    CORBA::StringSequence args;
    CORBA::StringSequence env;
};
```

## Activation policy

---

The following is a discussion of how the values of a `CreationImplDef` are used by the OAD when activating servers in response to client requests.

`CreationImplDef` provides methods that set the server's activation policy. These activation policies only apply to persistent objects, not transient objects.

Policy	Description
SHARED_SERVER	Multiple clients of a given object share the same implementation. Only one client is activated by an OAD at a particular time.
UNSHARED_SERVER	Only one client of a given implementation will ever be bound to the activated server. If multiple clients wish to bind to the same object implementation, a separate server is activated for each client. A server exits when its client application disconnects or exits.
SERVER_PER_METHOD	Each method invocation results in a new server being activated. The server exits when the method invocation completes.

---

## Examples

---

The following examples show how the OAD converts `CreationImplDef` attributes into executed commands.

### Java example

To activate the `VisiBroker` for Java application called `com.mycompany.Server` with the argument `CreditUnion` and the System Property `DEBUG` set to 1, fill out a `CreationImplDef` with the following attributes:

```
path_name = "vbj"
args = ["com.mycompany.Server", "CreditUnion"]
env = ["DEBUG=1"]
```

This would correspond to the OAD spawning the following command:

```
"vbj -DOAoad_uid=<uid> -DOAactivateIOR=<OAD's ior> -DDEBUG=1 \
com.mycompany.Server CreditUnion"
```

In addition, the following environment variables would be propagated from the OAD's environment into that of the spawned "vbj" execution.

- PATH
- CLASSPATH
- OSAGENT\_PORT
- OSAGENT\_ADDR
- VBROKER\_ADM

## Environment variables

---

When the registered Java class is activated by executing the `vbj` command, the OAD's environment is not automatically passed to the spawned process. If set, the environment variables listed in "Environment variables that are propagated or passed explicitly" on page 8-5, will be passed explicitly by the OAD.

All other environment variables must be registered using the `env` attribute in `CreationImplDef`.

For activated Java implementations, the environment settings, as recorded in `CreationImplDef`'s `env` attribute, are propagated in two ways:

- In the spawned `vbj` command's environment
- As System Properties to the class (that is, the `-D` arguments to the Java virtual machine)

Therefore, for spawned Java applications, the registration maps to the following executed command:

```
vbj -DOAoad_uid=<uid> -DOAactivateIOR=<oad's ior>
{ -Denv1 ... -DenvN }
className { args1 ... argsN }
```

Consequently, the spawned environment contains all the specified environment variables from the implementation definition as well as definitions for `PATH`, `CLASSPATH`, `OSAGENT_PORT`, and `OSAGENT_ADDR` which are taken from the OAD's own environment at startup. As with any OA parameter, those added by the OAD are stripped off during `BOA_init` and not seen by the client program.

## Environment variables that are propagated or passed explicitly

---

These are the environment variables that would be propagated from the OAD's environment into that of the spawned server or, if set, passed explicitly by the OAD.

- `PATH`
- `CLASSPATH`
- `OSAGENT_PORT`
- `OSAGENT_ADDR`
- `VBROKER_ADM`
- `LD_LIBRARY_PATH` as set in the `CreationImplDef`

## CreationImplDef methods

---

```
public abstract org.omg.CORBA.Policy activation_policy()
```

This method retrieves the server's activation policy.

```
public abstract void activation_policy(org.omg.CORBA.Policy activation_policy)
```

This method sets the server's activation policy. The activation policies are shared, unshared, and server-per-method.

Parameter	Description
<code>activation_policy</code>	The server's activation policy.

```
public abstract String[] args()
```

This method retrieves a list of arguments passed to the server.

```
public abstract void args(String args[])
```

This method sets the command-line arguments to be passed to the server. You must specify the class name as the first argument. For more information, see "Examples" on page 8-4.

Parameter	Description
<code>args</code>	An array of strings containing all command-line arguments.

```
public abstract String[] env()
```

This method retrieves a list of environment settings passed to the server.

public abstract void **env**(String **env**())

This method sets the environment settings to be passed to the server. For more information about setting the `env` attribute, see “Environment variables” on page 8-4.

Parameter	Description
<code>env</code>	An array of strings containing a list of environment settings.

public abstract byte[] **id**()

This method retrieves the reference data identifier for the implementation.

public abstract void **id**(byte **id**())

This method sets the reference data identifier for the implementation.

Parameter	Description
<code>id</code>	An array of bytes containing the implementation’s reference data identifier.

public abstract String **object\_name**()

This method retrieves the implementation’s object name.

public abstract void **object\_name**(String **object\_name**)

This method sets the implementation’s object name.

Parameter	Description
<code>object_name</code>	A string containing the implementation’s object name.

public abstract String **path\_name**()

For registrations, this method retrieves the string “`vbj`”.

public abstract void **path\_name**(String **path\_name**)

This method sets the exact path name of the executable program that implements the object. For programs, the path must be “`vbj`”.

Note The environment variable setting for the OAD’s path must be able to find the `vbj` executable. The OAD path is set during installation.

Parameter	Description
<code>path_name</code>	A string containing the implementation’s path name.

public abstract String **repository\_id**()

This method retrieves the implementation’s repository identifier.

```
public abstract void repository_id(String repository_id)
```

This method sets the implementation's repository identifier.

Parameter	Description
repository_id	A string containing the implementation's repository identifier.

## ImplementationDef

The `ImplementationDef` is an empty base class for the types of `ImplementationDefs`: `ActivationImplDef` and `CreationImplDef`. You only use `ImplementationDef` in signatures in the `ActivationImplDef` or the `CreationImplDef` methods.

## OAD

```
public interface OAD extends org.omg.CORBA.Object
```

The `OAD` interface provides access to the `OAD` (Object Activation Daemon). It is used by the administration tools for listing, registering, and unregistering objects. It can also be used by client code for programmatic administration of the `OAD`.

```
interface OAD {
    CreationImplDef create_CreationImplDef();

    Object reg_implementation(in extension::CreationImplDef impl)
        raises(DuplicateEntry, InvalidPath);

    CreationImplDef get_implementation(in CORBA::RepositoryId repId,
                                      in string object_name)
        raises(NotRegistered);

    void change_implementation(in extension::CreationImplDef old_info,
                              in extension::CreationImplDef new_info)
        raises(NotRegistered, InvalidPath, IsActive);

    attribute boolean destroy_on_unregister;

    void unreg_implementation(in CORBA::RepositoryId repId, in string object_name)
        raises(NotRegistered);

    void unreg_interface(in CORBA::RepositoryId repId)
        raises(NotRegistered);

    void unregister_all();

    ImplementationStatus get_status(in CORBA::RepositoryId repId,
                                    in string object_name)
        raises(NotRegistered);
}
```

## ImplementationStatus

```
ImplStatusList get_status_interface(in CORBA::RepositoryId repId)
    raises(NotRegistered);

ImplStatusList get_status_all();

Object lookup_interface(in CORBA::RepositoryId repId,
    in long timeout0
    raises(NotRegistered, FailedToExecute, NotResponding, Busy);

Object lookup_implementation(in CORBA::RepositoryId repId,
    in string object_name, in long timeout0
    raises(NotRegistered, FailedToExecute, NotResponding, Busy);

CreationImplDef boa_activate_obj(in Object obj,
    in string repository_id,
    in long unique_id)
    raises(NotRegistered);

void boa_deactivate3_obj(in Object obj,
    in string repository_id
    in long unique_id)
    raises(NotRegistered);

string generated_command(in extension::CreationImplDef impl);

string generated_environment(inextension::CreationImplDef impl);
};
```

## ImplementationStatus

---

**ImplementationStatus** is a struct which includes **impl** from **CreationImplDef** and the **status** from **ObjectStatusList**. **ObjectStatusList** is a struct specifying a **unique\_id** for a long type and **activation\_state** for **State**. The implementation can have one of the following activation states:

- Active
- Inactive
- Waiting for activation

```
module Activation
{
    . . .
    struct ObjectStatus {
        long    unique_id;
        State   activation_state;
        Object   objRef;
    };
    typedef sequence<ObjectStatus>ObjectStatusList;
    struct ImplementationStatus {
        extension::CreationImplDef   impl;
        ObjectStatusList              status;
    };
};
```



## OAD methods

---

```
public void change_implementation(org.omg.CORBA.CreationImplDef old_info,
    org.omg.CORBA.CreationImplDef new_info)
```

This method dynamically changes an object's implementation. You can use this method to change the registration's activation policy, path name, argument settings, and environment settings.

Parameter	Description
<code>old_info</code>	The information you want to change.
<code>new_info</code>	The information to replace the old info.

This method throws the following exceptions.

Exception	Description
<code>NotRegistered</code>	The object you specify is not registered. You must specify a registered object.
<code>InvalidPath</code>	The Java class or cpp executable is not found.
<code>IsActive</code>	The object implementation is currently running. Deactivate the object and then try to change its information.

### Caution

You cannot change information for a currently active implementation. Be sure to exercise caution when changing an object's implementation name and object name with this method. Doing so will prevent client applications from locating the object with the old name.

```
public abstract CreationImplDef boa_activate_obj (Object obj, string repository_id, long unique_id)
```

This method is invoked implicitly when the spawned process deactivates an object on which the `obj_is_ready` was previously called.

Parameter	Description
<code>obj</code>	A string containing an object name.
<code>repository_id</code>	A string containing a repository identifier.
<code>unique_id</code>	The same identifier used when the <code>boa_activate_obj</code> was invoked.

This method throws the following exception.

Exception	Description
<code>NotRegistered</code>	The object you specify is not registered. You must specify a registered object.

void **boa\_deactivate\_obj** (Object **obj**, string **repository\_id**, long **unique\_id**)

This method returns a string which represents the command-line option that is executed for a given implementation.

Parameter	Description
obj	A string containing an object name.
repository_id	A string containing a repository identifier.
unique_id	The same identifier used when the <code>boa_activate_obj</code> was invoked.

This method throws the following exception.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.

public abstract CreationImplDef **create\_CreationImplDef**()

This method creates an instance of `CreationImplDef`. You can then set its attributes as explained in “`CreationImplDef`” on page 8-3.

Exception	Description
DuplicateEntry	The object you specify is a duplicate entry. You must specify an unregistered object.
InvalidPath	The Java class is not found.

public abstract void **destroy\_on\_unregister**(boolean **destroy\_on\_unregister**)

This method sets the `destroy_on_unregister` attribute for the OAD. If the attribute is set to `true`, any active implementations are shut down when unregistered.

public abstract boolean **destroy\_on\_unregister**()

This method retrieves the setting for the `destroy_on_unregister` attribute for an implementation. If the attribute is set to `true`, any active implementations are shut down when unregistered.

public abstract String **generated\_command** (org.omg.CORBA.CreationImplDef **impl**)

This method returns a string which represents the command-line option that will be executed for a given implementation.

public abstract **String generated\_environment** (com.Inprise.vbroker.extension.CreationImplDef **impl**)

This method returns a string which represents the environment in which the spawned server is executed for a given implementation.

```
public org.omg.CORBA.CreationImplDef get_implementation(String repository_id,
String object_name)
```

This method retrieves information about implementations registered for the specified repository identifier and object name.

Parameter	Description
repository_id	A string containing a repository identifier.
object_name	A string containing an object name.

This method throws the following exceptions.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.
InvalidPath	The Java class or cpp executable is not found.
IsActive	The object implementation is currently running. Deactivate the object and then try to change its information.

```
public com.inprise.vbroker.Activation.ImplementationStatus get_status
(String repository_id, String object_name)
```

This method retrieves the status information about implementations registered for the specified repository identifier and object name.

Parameter	Description
repository_id	A string containing a repository identifier.
object_name	A string containing an object name.

```
public Activation.ImplementationStatus[] get_status_all()
```

This method gets the status information for all implementations.

```
public Activation.ImplementationStatus[] get_status_interface(String repository_id)
```

This method gets the status information about implementations registered for the specified repository identifier.

Parameter	Description
repository_id	A string containing a repository identifier.

```
public Activation ImplStatusList get_status_interface (CORBA::RepositoryId repld)
```

This method returns the Object reference for an implementation of the specified RepositoryId.

Parameter	Description
repository_id	A string containing a repository identifier.

This method throws the following exception.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.
FailedToExecute	The object that you specified failed to execute. An error occurred during the execution.
NotResponding	No response to the method.
Busy	The object that you specified is in use.

public Activation ImplStatusList **get\_status\_all** ()

This method returns status information for all registered implementations.

public org.omg.CORBA.Object **lookup\_interface** (CORBA::RepositoryId **replID**, long **timeout**)

This method looks up a specific implementation.

Parameter	Description
repository_id	A string containing a repository identifier.
timeout	

This method throws the following exceptions.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.
FailedToExecute	The object that you specified failed to execute. An error occurred during the execution.
NotResponding	No response to the method.
Busy	The object that you specified is in use.

public org.omg.CORBA.Object **lookup\_implementation** (CORBA::RepositoryId **replID**, string **object\_name**, long **timeout**)

This method is invoked implicitly when the spawned process calls BOA::obj\_is\_ready. This method does not need to be invoked directly by the client.

Parameter	Description
repository_id	A string containing a repository identifier.
object_name	A string containing an object name.
timeout	

This method throws the following exceptions.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.
FailedToExecute	The object that you specified failed to execute. An error occurred during the execution.
NotResponding	No response to the method.
Busy	The object that you specified is in use.

```
public org.omg.CORBA.Object reg_implementation(org.omg.CORBA.CreationImplDef impl)
```

This method registers an implementation with the OAD and the VisiBroker directory service.

Parameter	Description
<code>impl</code>	The instance of <code>CreationImplDef</code> .

This method throws the following exceptions.

Exception	Description
DuplicateEntry	The object you specify is a duplicate entry. You must specify an unregistered object.
InvalidPath	The Java class is not found.

```
public void unreg_implementation(String repository_id, String object_name)
```

This method unregisters implementations by repository identifier and object name. If the `destroy_on_unregister` attribute is set to `true`, this method terminates all processes currently implementing the repository identifier and object name that is specified.

Parameter	Description
<code>repository_id</code>	A string containing a repository identifier.
<code>object_name</code>	A string containing an object name.

This method throws the following exceptions.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.

public void **unreg\_interface**(String **repository\_id**)

This method unregisters all implementations for a repository identifier. If the `destroy_on_unregister` attribute is set to true, this method terminates all processes currently implementing the repository identifier specified.

Parameter	Description
repository_id	A string containing a repository identifier.

This method throws the following exceptions.

Exception	Description
NotRegistered	The object you specify is not registered. You must specify a registered object.

public void **unregister\_all**()

This method unregisters all implementations. Unless the attribute `destroy_on_unregister` is set to true, all active implementations continue to execute.

# Naming service interfaces and classes

This chapter describes the interfaces and classes for the VisiBroker Naming Service. The VisiBroker Naming Service is a complete implementation of the *Interoperable Naming Specification* document (orbos/98-10-11) from the OMG.

## NamingContext

---

public interface **NamingContext** extends com.inprise.vbroker.CORBA.Object

This object is used to contain and manipulate a list of names that are bound to ORB objects or to other `NamingContext` objects. Client applications use this interface to resolve or list all of the names within that context. Object implementations use this object to bind names to object implementations or to bind a name to a `NamingContext` object. The following IDL sample shows the IDL specification for the `NamingContext`.

## IDL definition

---

```
module CosNaming {

    interface NamingContext {
        void bind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName);
    };
};
```

```
Object resolve(in Name n)
    raises(NotFound, CannotProceed, InvalidName);
void unbind(in Name n)
    raises(NotFound, CannotProceed, InvalidName);
NamingContext new_context();
NamingContext bind_new_context(in Name n)
    raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
void destroy()
    raises(NotEmpty);
void list(in unsigned long how_many,
          out BindingList bl,
          out BindingIterator bi);
};
};
```

## NamingContext methods

---

```
public void bind(CosNaming.NameComponent[] n, org.omg.CORBA.Object obj) throws
    (CosNaming.NamingContextPackage.NotFound,
    CosNaming.NamingContextPackage.CannotProceed,
    CosNaming.NamingContextPackage.InvalidName,
    CosNaming.NamingContextPackage.AlreadyBound);
```

This method attempts to bind the specified `Object` to the specified `Name` by resolving the context associated with the first `NameComponent` and then binding the object to the new context using the following `Name`:

```
Name[NameComponent2, ..., NameComponent(n-1), NameComponentn]
```

This recursive process of resolving and binding continues until the context associated with the `NameComponent` ( $n-1$ ) is resolved and the actual name-to-object binding is stored. If parameter `n` is a simple name, the `obj` will be bound to `n` within this `NamingContext`.

Parameter	Description
<code>n</code>	A <code>Name</code> , initialized with the desired name for the object.
<code>obj</code>	The object to be named.

The following exceptions may be throws by this method.

Exception	Description
<code>NotFound</code>	The <code>Name</code> , or one of its components, could not be found.
<code>CannotProceed</code>	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
<code>InvalidName</code>	The specified <code>Name</code> has zero name components or the <code>id</code> field of one of its name components is an empty string.
<code>AlreadyBound</code>	The <code>Name</code> on a <code>bind</code> or <code>bind_context</code> operation has already been bound to another object within the <code>NamingContext</code> .



public void **rebind**(CosNaming.NameComponent[] **n**, in org.omg.CORBA.Object **obj**) throws  
 (CosNaming.NamingContextPackage.NotFound,  
 CosNaming.NamingContextPackage.CannotProceed,  
 CosNaming.NamingContextPackage.InvalidName);

This method is exactly the same as the `bind` method, except that the `AlreadyBound` exception will never be thrown. If the specified `Name` has already been bound to another object, that binding is replaced by the new binding.

Parameter	Description
<code>n</code>	A <code>Name</code> structure, initialized with the desired name for the object.
<code>obj</code>	The object to be named.

The following exceptions may be thrown by this method.

Exception	Description
<code>NotFound</code>	The <code>Name</code> , or one of its components, could not be found.
<code>CannotProceed</code>	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
<code>InvalidName</code>	The specified <code>Name</code> has zero name components or the <code>id</code> field of one of its name components is an empty string.

public void **bind\_context**(CosNaming.NameComponent[] **n**, CosNaming.NamingContext **nc**) throws  
 (CosNaming.NamingContextPackage.NotFound,  
 CosNaming.NamingContextPackage.CannotProceed, InvalidName,  
 CosNaming.NamingContextPackage.AlreadyBound);

This method is identical to the `bind` method, except that the supplied `Name` will be associated with a `NamingContext`, not an arbitrary ORB object.

Parameter	Description
<code>n</code>	A <code>Name</code> structure initialized with the desired name for the naming context. The first ( $n-1$ ) <code>NameComponent</code> structures in the sequence must resolve to a <code>NamingContext</code> .
<code>nc</code>	The <code>NamingContext</code> object to be bound.

The following exceptions may be thrown by this method.

Exception	Description
<code>NotFound</code>	The <code>Name</code> , or one of its components, could not be found.
<code>CannotProceed</code>	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
<code>InvalidName</code>	The specified <code>Name</code> has zero name components or the <code>id</code> field of one of its name components is an empty string.
<code>AlreadyBound</code>	The <code>Name</code> on a <code>bind</code> or <code>bind_context</code> operation has already been bound to another object within the <code>NamingContext</code> .

public void **rebind\_context**(CosNaming. NameComponent[] **n**, in CosNaming.NamingContext **nc**) throws  
(CosNaming.NamingContextPackage.NotFound,  
CosNaming.NamingContextPackage.CannotProceed,  
CosNaming.NamingContextPackage.InvalidName);

This method is exactly the same as the `bind_context` method, except that the `AlreadyBound` exception will never be thrown. If the specified `Name` has already been bound to another naming context, that binding is replaced by the new binding.

Parameter	Description
<code>n</code>	A <code>Name</code> structure, initialized with the desired name for the object.
<code>nc</code>	The <code>NamingContext</code> object to be rebound.

The following exceptions may be thrown by this method.

Exception	Description
<code>NotFound</code>	The <code>Name</code> , or one of its components, could not be found.
<code>CannotProceed</code>	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
<code>InvalidName</code>	The specified <code>Name</code> has zero name components or the <code>id</code> field of one of its name components is an empty string.

public org.omg.CORBA.Object **resolve**(CosNaming.NameComponent[] **n**) throws  
(CosNaming.NamingContextPackage.NotFound,  
CosNaming.NamingContextPackage.CannotProceed,  
CosNaming.NamingContextPackage.InvalidName);

This method attempts to resolve the specified `Name` and return an object reference. If parameter `n` is a *simple name*, it is resolved relative to this `NamingContext`.

If `n` is a *complex name*, it is resolved using the context associated with the first `NameComponent`. Next, the new context to resolve the following `Name`:

`Name [NameComponent(2), . . . , NameComponent(n-1), NameComponentn]`

This recursive process continues until the object associated with the *n*th `NameComponent` is returned.

Parameter	Description
<code>n</code>	A <code>Name</code> structure, initialized with the name for the desired object.

The following exceptions may be thrown by this method.

Exception	Description
NotFound	The Name, or one of its components, could not be found.
CannotProceed	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
InvalidName	The specified Name has zero name components or the <code>id</code> field of one of its name components is an empty string.

```
public void unbind(CosNaming.NameComponent[] n) throws
(CosNaming.NamingContextPackage.NotFound,
CosNaming.NamingContextPackage.CannotProceed,
CosNaming.NamingContextPackage.InvalidName);
```

This method performs the inverse operation of the `bind` method, removing the binding associated with the specified Name.

Parameter	Description
n	A Name structure, initialized with the desired name to be unbound.

The following exceptions may be thrown by this method.

Exception	Description
NotFound	The Name, or one of its components, could not be found.
CannotProceed	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned naming context.
InvalidName	The specified Name has zero name components or the <code>id</code> field of one of its name components is an empty string.

```
public CosNaming.NamingContext new_context();
```

This method creates a new naming context. The newly created context will be implemented within the same server as this object. The new context is initially not bound to any Name.

```
public CosNaming.NamingContext bind_new_context(in NameComponent[] n) throws
(CosNaming.NamingContextPackage.NotFound,
CosNaming.NamingContextPackage.CannotProceed,
CosNaming.NamingContextPackage.InvalidName,
CosNaming.NamingContextPackage.AlreadyBound);
```

This method creates a new context and binds it to the specified Name within this Context.

Parameter	Description
n	A Name structure, initialized with the desired Name for the newly created <code>NamingContext</code> object.

The following exceptions can be thrown by this method.

Exception	Description
NotFound	The Name, or one of its components, could not be found.
CannotProceed	One of the <code>NameComponent</code> objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned <code>NamingContext</code> .
InvalidName	The specified Name has zero name components or the <code>id</code> field of one of its name components is an empty string.
AlreadyBound	The Name on a <code>bind</code> or <code>bind_context</code> operation has already been bound to another object within the <code>NamingContext</code> .

`public void destroy() throws(CosNaming.NamingContextPackage.NotEmpty);`

This method deactivates this naming context. Any subsequent attempt to invoke operations on this object will throw a `CORBA.OBJECT_NOT_EXIST` runtime exception.

Before using this method, all `Name` objects that have been bound relative to this `NamingContext` should be unbound using the `unbind` method. Any attempt to **destroy** a `NamingContext` that is not empty will cause a `NotEmpty` exception to be thrown.

`public void list(int how_many,  
CosNaming.BindingList bl,  
CosNaming.BindingIterator bi)`

This method returns all of the bindings contained by this context. Up to “`how_many`” Names are returned with the `BindingList`. Any left over bindings will be returned via the `BindingIterator`. The returned `BindingList` and `BindingIterator`, described in detail in “Binding and BindingList” on page 9-8, can be used to navigate the list of names.

Parameter	Description
<code>how_many</code>	The maximum number of Names to be returned in the list.
<code>bl</code>	A list of Names returned to the caller. The number of names in the list will not exceed <code>how_many</code> .
<code>bi</code>	An iterator for use in traversing the rest of the Names.

## NamingContextExt

`public interface NamingContextExt extends CosNaming.NamingContext`

The `NamingContextExt` interface, which extends `NamingContext`, provides the operations required to use stringified names and URLs.

### IDL definition

```
module CosNaming {
```

```

interface NamingContextExt:NamingContext{
    typedef string StringName;
    typedef string Address;
    typedef string URLString;

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

    exception InvalidAddress {};

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

```

## NamingContextExt methods

---

public java.lang.String **to\_string**(CosNaming.NameComponent[] **n**) throws  
(CosNaming.NamingContextPackage.InvalidName);

This operation returns the stringified representation of the specified Name.

Parameter	Description
n	A Name structure initialized with the desired name for object.

The following exceptions can be thrown by this method.

Exception	Description
InvalidName	The specified Name has zero name components or the id field of one of its name components is an empty string.

public CosNaming.NameComponent[] **to\_name**(java.lang.String **sn**) throws  
(CosNaming.NamingContextPackage.InvalidName);

This operation returns a Name object for the specified stringified name.

Parameter	Description
sn	The stringified name of an object.

The following exceptions can be thrown by this method.

Exception	Description
InvalidName	The specified Name has zero name components or the id field of one of its name components is an empty string.

public java.lang.String **to\_url**(java.lang.String **addr**, java.lang.String **sn**) throws  
(CosNaming.NamingContextPackage.InvalidAddress,  
CosNaming.NamingContextPackage.InvalidName);

This operation returns a fully-formed string URL given the specified URL component and the stringified name.

Parameter	Description
addr	A URL component of the form “myhost.inprise.com:800”. If the Address is empty, it is the local host.
sn	A stringified name of an object.

The following exceptions can be thrown by this method.

Exception	Description
InvalidAddress	The specified Address is malformed.
InvalidName	The specified Name has zero name components or the id field of one of its name components is an empty string.

public org.omg.CORBA.Object **resolve\_str**(java.lang.String **n**) throws  
(CosNaming.NamingContextPackage.NotFound,  
CosNaming.NamingContextPackage.CannotProceed, InvalidName,  
CosNaming.NamingContextPackage.AlreadyBound);

This operation returns a Name object for the specified stringified name.

Parameter	Description
n	A stringified name of an object.

The following exceptions can be thrown by this method.

Exception	Description
NotFound	The Name, or one of its components, could not be found.
CannotProceed	One of the NameComponent objects in the sequence could not be resolved. The client may still be able to continue the operation from the returned NamingContext.
InvalidName	The specified Name has zero name components or the id field of one of its name components is an empty string.
AlreadyBound	The Name on a bind or bind_context operation has already been bound to another object within the NamingContext.

## Binding and BindingList

public interface **Binding**

The Binding, BindingList, and BindingIterator interfaces are used to describe the name-object bindings contained in a NamingContext. The Binding struct

encapsulates one such pair. The `binding_name` field represents the `Name` and the `binding_type` indicates whether the `Name` is bound to an ORB object or a `NamingContext` object.

The `BindingList` is a sequence of `Binding` structures contained by a `NamingContext` object.

## IDL definition

---

```
module CosNaming {

    enum BindingType {
        nobject,
        ncontext
    }

    struct Binding {
        Name binding_name;
        BindingType binding_type;
    };
    typedef sequence<Binding> BindingList;
};
```

## BindingIterator

---

public interface **BindingIterator** extends `com.inprise.vbroker.CORBA.Object`

This object allows a client application to walk through the unbounded collection of bindings returned by the `NamingContext` operation `list`, described in “public void `list(int how_many, CosNaming.BindingList bl, CosNaming.BindingIterator bi)`” on page 9-6.

## IDL definition

---

```
module CosNaming {

    interface BindingIterator {
        boolean next_one(out Binding b);
        boolean next_n(in unsigned long how_many, out BindingList b);
        void destroy();
    };
};
```

## BindingIterator methods

---

public boolean **next\_one**(CosNaming.Binding b);

This method returns the next `Binding` from the collection. The values `false` is returned if the list has been exhausted. Otherwise, the value `true` is returned.

Parameter	Description
b	The next Binding object from the list.

public boolean **next\_n**(int how\_many, CosNaming.BindingList b);

This method returns a `BindingList` containing the number of requested `Binding` objects from the list. The number of bindings returned may be less than the requested amount if the list is exhausted. The value `false` is returned when the list has been exhausted. Otherwise, the value `true` is returned.

Parameter	Description
how_many	The maximum number of Binding object desired.
b	A BindList containing no more than the requested number of Binding objects.

public void **destroy**();

This method destroys this object and releases the memory associated with the object. Failure to call this method will result in increased memory usage.

## NamingContextFactory

---

public interface **NamingContextFactory** extends com.inprise.vbroker.CORBA.Object

This interface is provided to instantiate an initial `NamingContext`. A client may bind to an object of this type and use the `create_context` method to create an initial context. Once the initial context has been created, the `new_context` method, described in “public CosNaming.NamingContext new\_context();” on page 9-5, can be used to create other contexts. An instance of this naming context factory is created when the naming service is started, described in Chapter 18, “Using the Naming Service,” of the *VisiBroker for Java Programmer’s Guide*.

To create an initial `NamingContextFactory` that automatically creates a single root context, see “`ExtendedNamingContextFactory`” on page 9-11.

## IDL definition

---

```
module CosNaming {
```



```
interface NamingContextFactory {
    NamingContext create_context();
    oneway void shutdown();
};
```

## NamingContextFactory methods

---

public CosNaming.NamingContextExt **create\_context**();

This method allows a client to create a naming context. Since the specification for naming contexts states that they do not have any notion of a root context, simply instantiating a `NamingContextFactory` will not create a naming context.

public void **shutdown**();

This method allows a client to gracefully shutdown the naming service. If the service is restarted with the same logfile, the factory will be restored to the state it had prior to being shutdown.

public CosNamingExt.ClusterManager **get\_cluster\_manager**();

This method creates a cluster using a specified cluster criterion.

public void **remove\_stale\_contexts** (java.lang.String password);

This method allows the client to remove members from the Cluster throughout its lifetime.

public CosNaming.NamingContext[] **list\_all\_roots** (java.lang.String password);

This method allows you to list all root contexts.

## ExtendedNamingContextFactory

---

public interface **ExtendedNamingContextFactory** extends CosNamingExt.NamingContextFactory

This interface extends the `NamingContextFactory` interface and allows the creation of a default root within a factory when the extended naming service is started, described in Chapter 18, “Using the Naming Service,” of the *VisiBroker for Java Programmer’s Guide*.

### IDL definition

---

```
module CosNaming {

    interface ExtendedNamingContextFactory : NamingContextFactory{
        NamingContext root_context();
    };
};
```

## ExtendedNamingContextFactory methods

---

```
public CosNaming.NamingContextExt root_context();
```

This method returns the root naming context that was automatically created when this object was instantiated.

# Event service interfaces and classes

This chapter describes the interfaces and classes for the VisiBroker Event Service.

## ConsumerAdmin

---

class **org.omg.CosEventChannelAdmin.ConsumerAdmin** extends ConsumerAdminPOA

This interface is used by consumer applications to obtain a reference to a proxy supplier object. This is the second step in connecting a consumer application to an EventChannel.

### IDL definition

---

```
module CosEventChannelAdmin {  
    interface ConsumerAdmin {  
        ProxyPushConsumer obtain_push_supplier();  
        ProxyPullConsumer obtain_pull_supplier();  
    };  
};
```

### Java definition

---

```
class ConsumerAdmin {  
    org.omg.CosEventChannelAdmin.ProxyPushSupplier obtain_push_supplier ();  
    org.omg.CosEventChannelAdmin.ProxyPullSupplier obtain_pull_supplier ();  
}
```

## ConsumerAdmin methods

---

```
public org.omg.CosEventChannelAdmin.ProxyPushSupplier obtain_push_supplier();
```

The `obtain_push_supplier` method is invoked if the calling consumer application is implemented using the push model. If the application is implemented using the pull model, the `obtain_pull_supplier` method should be invoked.

```
public org.omg.CosEventChannelAdmin.ProxyPullSupplier obtain_pull_supplier();
```

The returned reference is used to invoke either the `connect_push_consumer`, described in “ProxyPushSupplier” on page 10-7, or the `connect_pull_consumer` method, described in “ProxyPullSupplier” on page 10-6.

## EventChannel

---

```
class org.omg.CosEventChannelAdmin.EventChannel
```

The `EventChannel` class provides the administrative operations for adding suppliers and consumers to the channel and for destroying the channel.

Suppliers and consumers both use the `resolve_initial_instances` method to obtain an `EventChannel` reference. If the object name is not specified, a suitable `EventChannel` will be located by `VisiBroker`. Once a supplier or consumer is connected to an `EventChannel`, it may invoke any of the `EventChannel` methods.

## Java definition

---

```
class org.omg.CosEventChannelAdmin.EventChannel {
    org.omg.CosEventChannelAdmin.ConsumerAdmin for_consumers ();
    org.omg.CosEventChannelAdmin.SupplierAdmin for_suppliers ();
    void destroy ();
}
```

## EventChannel methods

---

```
org.omg.CosEventChannelAdmin.ConsumerAdmin for_consumers();
```

This method returns an `org.omg.CosEventChannelAdmin.ConsumerAdmin` object that can be used to add consumers to this `EventChannel`.

```
org.omg.CosEventChannelAdmin.SupplierAdmin for_suppliers();
```

This method returns an `org.omg.CosEventChannelAdmin.SupplierAdmin` object that can be used to add suppliers to this `EventChannel`.

```
void destroy();
```

This method destroys this `EventChannel`.

# EventLibrary (Java)

---

class **com.inprise.vbroker.CosEvent.EventLibrary**

The `EventLibrary` class provides several methods for creating an `EventChannel` within an application's process. Using an in-process event channel frees you from having to start a separate event channel or event channel factory process.

## Java definition

---

```
class com.inprise.vbroker.CosEvent.EventLibrary {
    static org.omg.PortableServer.POA getRootPoa();
    static org.omg.PortableServer.POA getTransientPoa();
    static org.omg.PortableServer.POA getPersistentPoa();
    static org.omg.CORBA.ORB orb();
    static org.omg.CORBA.Object activate(
        org.omg.PortableServer.POA poa,
        org.omg.PortableServer.Servant servant, String name);
    static org.omg.CORBA.Object activatePersistent(
        org.omg.PortableServer.Servant servant, String name);
    static org.omg.CORBA.Object activateTransient(
        org.omg.PortableServer.Servant servant);
    static EventChannel create_channel(
        String name, boolean debug, int maxQueueLength);
    static EventChannel create_channel(boolean debug, int maxQueueLength);
    static EventChannel create_channel(String name, boolean debug);
    static EventChannel create_channel(String name);
    static EventChannel create_channel();
    static EventChannelFactory create_factory(
        String name, boolean debug, int maxQueueLength);
    static EventChannelFactory create_factory(String name, boolean debug);
    static EventChannelFactory create_factory(String name);
}
```

## EventLibrary methods

---

static org.omg.CosEventChannelAdmin.EventChannel **create\_channel**(  
 String name, boolean debug, int maxQueueLength);

This method creates an `EventChannel` with the specified `name`, `debug`, and `queue length` settings.

Parameter	Description
<code>name</code>	The name to be used for this channel.
<code>debug</code>	If set to <code>true</code> , debugging output is enabled. If set to <code>false</code> , debugging output is disabled.
<code>maxQueueLength</code>	The maximum number of messages that may be queued for each consumer.

---

```
static org.omg.CosEventChannelAdmin.EventChannel create_channel(  
    String name, boolean debug);
```

This method creates an `EventChannel` with the specified `name`, and `debug` settings. The queue length for each consumer is set to 100.

Parameter	Description
name	The name to be used for this channel.
debug	If set to <code>true</code> , debugging output is enabled. If set to <code>false</code> , debugging output is disabled.

```
static org.omg.CosEventChannelAdmin.EventChannel create_channel(  
    String name);
```

This method creates an `EventChannel` with the specified `name`. The `EventChannel` object's debug flag is set to `false` and the queue length is set to 100.

Parameter	Description
name	The name to be used for this channel.

```
static org.omg.CosEventChannelAdmin.EventChannel create_channel();
```

This method creates an `EventChannel`. The `EventChannel` object is given no name, the debug flag is set to `false`, and the queue length is set to 100.

## ProxyPullConsumer

```
class org.omg.CosEventChannelAdmin.ProxyPullConsumer
```

This class is used by a pull supplier application. It provides the `connect_pull_supplier` method for connecting the supplier's `PullSupplier`-derived object to the `EventChannel`. An `AlreadyConnected` exception is raised if an attempt is made to connect the same proxy more than once.

### IDL definition

```
module CosEventChannelAdmin {  
    exception AlreadyConnected();  
    interface ProxyPullConsumer : CosEventComm::PullConsumer {  
        void connect_pull_supplier(in CosEventComm::PullSupplier pull_supplier)  
            raises (AlreadyConnected);  
    };  
};
```

## Java definition

---

```
class org.omg.CosEventChannelAdmin.ProxyPullConsumer extends
    org.omg.CosEventComm.PullConsumerOperations {
    public void connect_pull_supplier (org.omg.CosEventComm.PullSupplier pull_supplier)
        throws org.omg.CosEventChannelAdmin.AlreadyConnected;
}
```

## ProxyPullConsumer method

---

void **connect\_pull\_supplier** (org.omg.CosEventComm.PullSupplier pull\_supplier)

This method connects a PullSupplier to an EventChannel.

Parameter	Description
pull_supplier	The PullSupplier-derived object to be connected to the EventChannel.

## ProxyPushConsumer

---

public interface org.omg.CosEventChannelAdmin.**ProxyPushConsumer**

This interface is used by a push supplier application and provides the `connect_push_supplier` method, used for connecting the supplier's PushSupplier-derived object to the EventChannel. An `AlreadyConnected` exception is raised if an attempt is made to connect the same proxy more than once.

## IDL definition

---

```
module CosEventChannelAdmin {
    exception AlreadyConnected();
    interface ProxyPushConsumer : CosEventComm::PushConsumer {
        void connect_push_supplier(in CosEventComm::PushSupplier push_supplier)
            raises(AlreadyConnected);
    };
};
```

## Java definition

---

```
public interface ProxyPushConsumer extends
    org.omg.CosEventComm.PushConsumer {
    public void connect_push_supplier
        (org.omg.CosEventComm.PushSupplier push_supplier) throws
        org.omg.CosEventChannelAdmin.AlreadyConnected;
}
```

## ProxyPushConsumer method

---

void **connect\_push\_supplier** (org.omg.CosEventComm.PushSupplier push\_supplier)

This method connects a PushSupplier to an EventChannel.

Parameter	Description
push_supplier	PushSupplier-derived object to be connected to the EventChannel.

## ProxyPullSupplier

---

public interface **org.omg.CosEventChannelAdmin.ProxyPullSupplier**

This interface is used by a pull consumer application and provides the `connect_pull_consumer` method, used for connecting the consumer's PullConsumer-derived object to the EventChannel. An `AlreadyConnected` exception will be raised if an attempt is made to connect the same PullConsumer more than once.

### IDL definition

---

```
module CosEventChannelAdmin {
    exception AlreadyConnected();
    interface ProxyPullSupplier : CosEventComm::PullSupplier {
        void connect_pull_consumer(in CosEventComm::PullConsumer pull_consumer)
            raises(AlreadyConnected);
    };
};
```

### Java definition

---

```
public interface ProxyPullSupplier extends
    org.omg.CosEventComm.PullSupplier {
    void connect_pull_consumer (org.omg.CosEventComm.PullConsumer pull_consumer)
}
```

## ProxyPullSupplier method

---

void **connect\_pull\_consumer** (org.omg.CosEventComm.PullConsumer pull\_consumer)

This method connects a PullConsumer to an EventChannel.

Parameter	Description
pull_consumer	PullConsumer-derived object to be connected to the EventChannel.



# ProxyPushSupplier

---

public interface org.omg.CosEventChannelAdmin.**ProxyPushSupplier**

This interface is used by a push consumer application and provides the `connect_push_consumer` method, used for connecting the consumer's `PushConsumer`-derived object to the `EventChannel`. An `AlreadyConnected` exception will be raised if an attempt is made to connect the same `PushConsumer` more than once.

## IDL definition

---

```
module CosEventChannelAdmin {
    exception AlreadyConnected();
    interface ProxyPushSupplier : CosEventComm::PushSupplier {
        void connect_push_consumer(in CosEventComm::PushConsumer push_consumer)
            raises(AlreadyConnected);
    };
};
```

## Java definition

---

```
public interface org.omg.CosEventChannelAdmin.ProxyPushSupplier extends
    org.omg.CosEventComm.PushSupplier {
    public void connect_push_consumer (org.omg.CosEventComm.PushConsumer
        push_consumer) throws org.omg.CosEventChannelAdmin.AlreadyConnected;
}
```

## ProxyPushSupplier method

---

void **connect\_push\_consumer** (org.omg.CosEventComm.PushConsumer push\_consumer)

This method connects a `PushConsumer` to an `EventChannel`.

Parameter	Description
push_consumer	<code>PushConsumer</code> -derived object to be connected to the <code>EventChannel</code> .

---

# PullConsumer

---

public interface org.omg.CosEventComm.**PullConsumer**

This interface is used to derive consumer objects that use the pull model of communication. The `pull` method is called by a consumer whenever it wants data from the supplier. A `Disconnected` exception will be raised if the supplier has disconnected.

## IDL definition

---

```
module CosEventChannelAdmin {
    exception AlreadyConnected();
    interface PullConsumer {
        void disconnect_pull_consumer ();
    };
}
```

## Java definition

---

```
public interface PullConsumer{
    public void disconnect_pull_consumer ();
}
```

## PullConsumer method

---

void **disconnect\_push\_consumer** ()

The `disconnect_push_consumer` method deactivates this consumer if the channel is destroyed.

## PushConsumer

---

public interface org.omg.CosEventComm.**PushConsumer**

This interface is used to derive consumer objects that use the push model of communication. The `push` method is used by a supplier whenever it has data for the consumer. A `Disconnected` exception will be raised if the consumer has disconnected.

## IDL definition

---

```
module CosEventComm {
    exception Disconnected();
    interface PushConsumer {
        void push(in any data) raises(Disconnected);
        void disconnect_push_consumer();
    };
}
```

## Java definition

---

```
interface org.omg.CosEventComm.PushConsumer {
    public void push (org.omg.CORBA.Any data) throws org.omg.CosEventComm.Disconnected;
    public void disconnect_push_consumer ();
}
```

## PushConsumer methods

---

void **disconnect\_push\_consumer** ()

The `disconnect_push_consumer` method deactivates this consumer if the channel is destroyed.

void **push** (org.omg.CORBA.Any data)

A supplier communicates event data to the consumer by invoking the `push()` method and passing the event data as a parameter. If the event communication has already been disconnected, the `Disconnected` exception is raised.

Parameter	Description
data	Information to be passed to the consumer.

---

## PullSupplier

---

public interface **org.omg.CosEventComm.PullSupplier**

This interface is used to derive supplier objects that use the pull model of communication.

### IDL definition

---

```
module CosEventComm {
    interface PullSupplier {
        any pull() raises(Disconnected);
        any try_pull(out boolean has_event) raises(Disconnected);
        void disconnect_pull_supplier();
    };
};
```

### Java definition

---

```
public interface PullSupplierOperations {
    public org.omg.CORBA.Any pull () throws org.omg.CosEventComm.Disconnected;
    public org.omg.CORBA.Any try_pull (org.omg.CORBA.BooleanHolder has_event)
        throws org.omg.CosEventComm.Disconnected;
    public void disconnect_pull_supplier ();
}
```

## PullSupplier methods

---

any **pull()**;

This method blocks until there is data available from the supplier. The data returned is an **Any** type. If the consumer has disconnected, this method raises a **Disconnected** exception.

any **try\_pull**(org.omg.CORBA.BooleanHolder has\_event);

This non-blocking method attempts to retrieve data from the supplier. When this method returns, **has\_event** is set to the value **true** and the data is returned as an **any** type if there was data available. If **has\_event** is set to the value **false**, then no data was available and the return value will be **NULL**.

Parameter	Description
has_event	When the method returns, has_event contains true if data is being returned. Otherwise, it contains false.

---

void **disconnect\_pull\_supplier()**;

This method deactivates this pull server if the channel is destroyed.

## PushSupplier

---

public interface **org.omg.CosEventComm.PushSupplier**

This interface is used to derive supplier objects that use the push model of communication. The **disconnect\_push\_supplier** method is used by the **EventChannel** to disconnect supplier when it is destroyed.

## IDL definition

---

```
module CosEventComm {
    exception AlreadyConnected();
    interface PushSupplier {
        void disconnect_push_supplier();
    };
};
```

## Java definition

---

```
public interface org.omg.CosEventComm.PushSupplierOperations {
    public void disconnect_push_supplier ();
}
```

## PushSupplier method

---

void **disconnect\_push\_supplier**();

This method deactivates this push supplier if the channel is destroyed.

## SupplierAdmin

---

public interface **org.omg.CosEventChannelAdmin.SupplierAdmin**

This interface is used by supplier applications to obtain a reference to the proxy consumer object. This is the second step in connecting a supplier application to an `EventChannel`.

## IDL definition

---

```
module CosEventChannelAdmin {
    interface SupplierAdmin {
        ProxyPushConsumer obtain_push_consumer();
        ProxyPullConsumer obtain_pull_consumer();
    };
};
```

## Java definition

---

```
public interface SupplierAdmin {
    public org.omg.CosEventChannelAdmin.ProxyPushConsumer obtain_push_consumer ();
    public org.omg.CosEventChannelAdmin.ProxyPullConsumer obtain_pull_consumer ();
}
```

## SupplierAdmin methods

---

You should invoke the `obtain_push_consumer` method if you are implementing the supplier application using the push model. If the application is implemented using the pull model, the `obtain_pull_consumer` method should be invoked.

public ProxyPushConsumer **obtain\_push\_consumer**();

The `obtain_push_consumer` method returns a `ProxyPushConsumer` object, which is then used to connect a push-style supplier.

public ProxyPullConsumer **obtain\_pull\_consumer**();

The returned reference is used to invoke either the `connect_push_supplier`, described on “ProxyPushConsumer” on page 10-5, or the `connect_pull_supplier` method, described in “ProxyPullConsumer” on page 10-4.



# Exceptions classes

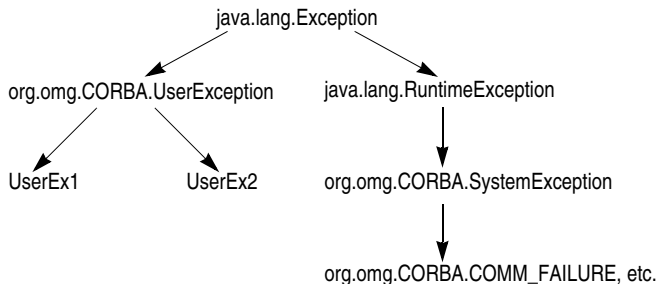
This chapter describes the exception classes used in VisiBroker.

## Introduction

CORBA system exceptions are a subclass of `java.lang.RuntimeException`. This means that CORBA system exceptions do not need to be declared in all method signatures which might raise such exceptions.

The `UserException` is a subclass of `java.lang.Exception`.

**Note** Due to the changes in the inheritance hierarchy, the class `org.omg.CORBA.Exception` no longer exists.



## SystemException

public class **SystemException** extends `java.lang.RuntimeException`

CORBA system exceptions are raised when the runtime encounters problems. They inherit from `java.lang.RuntimeException`. Table 11.1 summarizes all the `SystemException` classes that can be raised and their associated meanings.

The standard IDL system exceptions are mapped to final Java classes that extend `org.omg.CORBA.SystemException` and provide access to the IDL major and minor exception code, as well as a string describing the reason for the exception.

**Note** There are no public constructors for `org.omg.CORBA.SystemException`; only classes that extend it can be instantiated.

Currently, VisiBroker does not support the use of minor codes; consequently, there is no `minor` method to set the minor code.

## SystemException attributes

---

`public CompletionStatus` **completed**

This attribute indicates whether the operation was completed or not.

**Table 11.1** List of system exceptions

Exception class name	Description
BAD_CONTEXT	Error processing context object.
BAD_INV_ORDER	Routine invocations out of order.
BAD_OPERATION	Invalid operation.
BAD_PARAM	An invalid parameter was passed.
BAD_TYPECODE	Invalid typecode.
COMM_FAILURE	Communication failure.
DATA_CONVERSION	Data conversion error.
FREE_MEM	Unable to free memory.
INDIRECTION EXCEPTION	Mismatch in typecode indirection when reading or writing.
IMP_LIMIT	Implementation limit violated.
INITIALIZE	ORB initialization failure.
INTERNAL	ORB internal error.
INTF_REPOS	Error accessing interface repository.
INV_FLAG	Invalid flag was specified.
INV_IDENT	Invalid identifier syntax.
INV_OBJREF	Invalid object reference specified.
INV_POLICY	Invalid policy for invocation.
INVALID_TRANSACTION	Invalid transaction context encountered.
MARSHAL	Error marshalling parameter or result.
NO_IMPLEMENT	Operation implementation not available.
NO_MEMORY	Dynamic memory allocation failure.
NO_PERMISSION	No permission for attempted operation.
NO_RESOURCES	Insufficient resources to process request.
NO_RESPONSE	Response to request not yet available.
OBJ_ADAPTER	Failure detected by object adaptor.



**Table 11.1** List of system exceptions (continued)

Exception class name	Description
OBJECT_NOT_EXIST	Object is not available.
PERSIST_STORE	Persistent storage failure.
REBIND	IOR conflicts with the QoS policies that were set.
TRANSACTION_REQUIRED	Non-null transaction context required.
TRANSACTION_ROLLEDBACK	Transaction has been rolled back.
TRANSIENT	Transient failure.
UNKNOWN	A Java exception of type <code>java.lang.Exception</code> is thrown.

## UserException

---

`public class UserException` extends `java.lang.Exception`

The `UserException` class is an abstract base class used to define exceptions that an object implementation may raise. There is no state information associated with these exception types, but derived classes may add their own state information. The primary use of this class is to simplify the use of `catch` blocks in a client's code, as shown in Code sample 11.1.

**Code sample 11.1** Catching system and user exceptions

```
try {
    proxy.operation();
}
catch(org.omg.CORBA.SystemException se) {
    System.out.println("The runtime failed: " + se);
}
catch(org.omg.CORBA.UserException ue) {
    System.out.println("The implementation failed: " + ue);
}
```

## UserException constructor

---

`protected UserException()`

This method creates a `UserException` exception.



# Interceptor and object wrapper interfaces and classes

This chapter describes the VisiBroker interfaces and classes used with interceptors and object wrappers. For information about how to create and use interceptors and object wrappers, see the VisiBroker for Java *Programmer's Guide*.

## Introduction

---

The ORB provides a set of APIs known as interceptors which provide a way to plug in additional ORB behavior such as support for transactions and security. The three forms of interceptors are listed in the following table.

Interceptor Type	Description
Client Interceptor	System level interceptors which can be used to hook ORB services such as transactions and security into the client ORB processing.
Server Interceptor	System level interceptors which can be used to hook ORB services such as transactions and security into the server ORB processing.
Object Wrappers	User level interceptors which provide a simple mechanism for users to intercept calls to stubs and skeletons allowing for simple tracing and data caching among other things.

## InterceptorManagers

---

Interceptors are installed and managed via interceptor managers. The interface `InterceptorManager` is the global interceptor manager used to manager all global interceptors. Global interceptors may be handed additional interceptor

managers to install localized interceptors, for example, per-POA interceptors use the `POAInterceptorManager`.

An instance of the global interceptor manager, `InterceptorManager`, may be obtained via `ORB.resolve_initial_references` when passing the `String VisiBrokerInterceptorControl` as an argument. This value is only available when the ORB is in administrative mode, that is, during ORB initialization.

## IOR templates

---

In addition to the interceptor, the Interoperable Object Reference (IOR) template may be modified directly on the `POAInterpcptorManager` interface during the call to `POALifeCycleInterceptor.create`. The IOR template is a full IOR value with the `type_id` not set, and all `GIOP ProfileBodyValues` will have incomplete object keys. The POA sets the `type_id` and fills in the object keys of the template before calling the `IORCreationInterceptors`.

## InterceptorManager

---

public interface `InterceptorManager`

This is the base class from which all other interceptor managers are derived. Interceptor mangers are interfaces which are used to manage the installation and removal of interceptors from the system.

## InterceptorManagerControl

---

public interface `InterceptorManagerControl`

This is the interface that is responsible for controlling a set of related interceptor managers.

## Import statement

---

The import statement `import com.inprise.vbroker.interceptor.*;` should be included in your code.

## InterceptorManagerControl method

---

public.com.inprise.vbroker.interceptor.InterceptorManager **get\_manager** (java.lang.string name);

This method is invoked by the ORB to obtain an instance of the an InterceptorManger which returns a string identifying the manager.

Parameter	Description
string name	The name of the inceptor.

---

## BindInterceptor

---

public interface **BindInterceptor**

You can use this interface to derive your own interceptor for handling bind and rebind events for a client or server application. The Bind Interceptors are global interceptors invoked on the client side before and after binds.

If an exception is thrown during a bind, the remaining interceptors in the chain are not called and the chain is truncated to only those interceptors already called. Exceptions thrown during `bind_succeeded` or `bind_failed` are ignored.

## Import statement

---

The import statement **import com.inprise.vbroker.interceptor.\*;** should be included in your code.

## BindInterceptor methods

---

public.com.inprise.vbroker.IOP.IORValue **bind** (com.inprise.vbroker.IOP.IORValue ior, org.omg.CORBA.Object obj, boolean **rebind**, com.inprise.vbroker.interceptor.Closure **closure**);

This method is called during all ORB bind operations.

Parameter	Description
ior	The Interoperable Object Reference (IOR) for the server object to which the client is binding.
obj	The client object which is being bound to the server. The object will not be properly initialized at this time, so do not attempt an operation on it. However, it may be stored in a data structure and used after the bind has completed.
rebind	Specifies whether it is a rebind attempt.
closure	A new closure object for the bind operation. The closure will be used in corresponding calls to either <code>bind_failure</code> or <code>bind_succeeded</code> .

---

```
public com.inprise.vbroker.IOP.IORValue bind_failed (com.inprise.vbroker.IOP.IORValue ior,  
org.omg.CORBA.Object obj,  
com.inprise.vbroker.interceptor.Closure closure);
```

This method is called if a bind operation failed.

Parameter	Description
ior	The IOR of the server object on which the bind operation failed.
obj	The client object which is being bound to the server.
closure	The closure object previously given in the bind call.

```
public void bind_succeeded (com.inprise.vbroker.IOP.IORValue ior, org.omg.CORBA.Object obj,  
int profileIndex,  
com.inprise.vbroker.interceptor.InterceptorManagerContro interceptorControl,  
com.inprise.vbroker.interceptor.Closure closure);
```

This method is called if a bind operation succeeded.

Parameter	Description
ior	The IOR of the server object on which the bind operation succeeded.
obj	The client object which is being bound to the server.
profileIndex	The index of the profile that actually resulted in successful binding.
interceptorControl	This Manager provides a list of the types of Managers.
closure	The closure object previously given in the bind call.

```
public void exception_occurred (com..inprise.vbroker.IOP.IORValue ior,  
org.omg.CORBA.Object obj,  
org.omg.CORBA::Environment env,  
com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked by the ORB when an exception occurs.

Parameter	Description
ior	The IOR of the server object on which the bind operation succeeded.
target	The client object which is being bound to the server.
env	Contains information on the exception that was raised.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

## BindInterceptorManager

```
public interface BindInterceptorManager extends com.inprise.vbroker.interceptor.InceptorManager
```

This interface is used to add BindInterceptor.

## Import statement

---

The import statement **import com.inprise.vbroker.interceptor.\*;** should be included in your code.

## BindInterceptorManager method

---

```
public void add (com.inprise.vbroker.interceptor.BindInterceptor interceptor);
```

This method is used to add a BindInterceptor.

## ClientRequestInterceptor

---

```
public interface ClientRequestInterceptor
```

You use this interface to derive your own client-side interceptor, providing implementations for those methods that you wish to override. The Client Request interceptors may be installed during the `bind_succeeded` call of a bind interceptor and remain active for the duration of the connection. The methods defined in your derived class will be invoked by the ORB during the preparation or sending of an operation request, during the receipt of a reply message, or if an exception is raised.

## Import statement

---

The import statement **import com.inprise.vbroker.interceptor.\*;** should be included in your code.

## ClientRequestInterceptor methods

---

```
public void preinvoke_premarshal (org.omg.CORBA.Object target,
    java.lang.String operation, com.inprise.vbroker.IOP.ServiceContextListHolder service_contexts,
    com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked by the ORB on every request, before the request is marshaled. An exception thrown from this interceptor will result in the request being completed immediately. The chain will be shortened to only those interceptors that have already fired, the request will not be sent, and `exception_occured()` will be called on all interceptors still in the chain.

Parameter	Description
<code>target</code>	The client object which is being bound to the server.
<code>operation</code>	Identifies the name of the operation being invoked.

Parameter	Description
service_context	Identifies the services assigned by the Orb. These services are registered with the OMG.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

```
public void preinvoke_postmarshal (org.omg.CORBA.Object target,
com.inprise.vbroker.CORBA.portable.Outstream payload,
com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked after every request has been marshaled, but before it was sent. Any exception thrown in this method will cause the rest of the chain to not be invoked anymore. The request will not be sent to the server. `exception_occured()` will subsequently be called on the whole interceptor chain.

Parameter	Description
target	The client object which is being bound to the server.
payload	Marshaled buffer.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

```
public void postinvoke (org.omg.CORBA.Object target,
com.inprise.vbroker.IOP.ServiceContext [] Service_contexts,
com.inprise.vbroker.CORBA.portable.InputStream payload,
org.omg.CORBA.Environment env,
com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked after a request has completed.

Parameter	Description
target	The client object which is being bound to the server.
service_context	Identifies the services assigned by the Orb. These services are registered with the OMG.
payload	Marshaled buffer.
env	Contains information on the exception that was raised.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

```
public void exception_occurred (org.omg.CORBA.Object target, org.omg.CORBA.Environment env,
com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked by the ORB when an exception was thrown before the invocation. All exceptions thrown after the invocation will be gathered in the Environment parameter of the `postinvoke` method.



# ClientRequestInterceptorManager

---

public interface **ClientRequestInterceptorManager** extends **com.inprise.vbroker.interceptor.InterceptorManager**

This interface is used to add and remove **ClientRequestInterceptor**.

## Import statement

---

The import statement **import com.inprise.vbroker.interceptor.\*;** should be included in your code.

## ClientRequestInterceptorManager methods

---

public void **add** (com.inprise.vbroker.interceptor.ClientRequestInterceptor interceptor);

This method is invoked by the ORB to add a **ClientRequestInterceptor**.

# POALifeCycleInterceptor

---

public interface **POALifeCycleInterceptor**

The **POALifeCycleInterceptor** is a global interceptor which is invoked every time a POA is created or destroyed. All other Server-side interceptors may be installed either as global interceptors, or for a specific POAs. You install the **POALifeCycleInterceptor** through the **InterceptorManager** interface. The **POALifeCycleInterceptor** is called during POA creation and destruction.

## Import statement

---

The import statements **import com.inprise.vbroker.interceptor.\*** and **com.inprise.vbroker.PortableServerExt.\*;** should be included in your code.

## POALifeCycleInterceptor methods

---

public void **create** (POA poa  
org.omg.CORBA.PolicyListHolder **policies**,  
IORValueHolder **ior**,  
InterceptorManagerControl **poaAdmin**);

This method is invoked when a new POA is created either explicitly through a call to **create\_POA** or via **AdaptorActivator**. With **AdaptorActivator**, the

interceptor is called only after the `unknown_adapter` method successfully returns from the `AdapterActivator`.

Parameter	Description
<code>poa</code>	The POA that is being created.
<code>policies</code>	The list of policies specified for this POA.
<code>iorTemplate</code>	The IOR template is a full IOR value with the <code>type_id</code> not set, and all <code>GIOP.ProfileBodyValues</code> will have incomplete object keys.
<code>poaAdmin</code>	The <code>InterceptorManagerControl</code> used to obtain other interceptor managers.

public void **destroy** (POA `poa`);

This method is called when a POA is just about to be destroyed and all of its objects have been etherialized. It is guaranteed that `destroy` will be called on all interceptors before `create` will be called again for a POA with the same name. If the destroy operation throws a system exception it is ignored, and the remaining interceptors will continue to be called.

Parameter	Description
<code>poa</code>	Portable Object Adaptor (POA) being destroyed.

## POALifeCycleInterceptorManager

public interface **POALifeCycleInterceptorManager**

This interface is used to register `POALifeCycleInterceptor`.

### Import statement

The import statements `import com.inprise.vbroker.interceptor.*` and `com.inprise.vbroker.PortableServerExt.*;` should be included in your code.

### POALifeCycleInterceptorManager method

public void **add** (POALifeCycleInterceptor `interceptor`);

This method is invoked by the ORB to add a `POALifeCycleInterceptor`.

Parameter	Description
<code>interceptor</code>	The interceptor to be added.

# ActiveObjectLifeCycleInterceptor

---

public interface **ActiveObjectLifeCycleInterceptor**

The `ActiveObjectLifeCycleInterceptor` interface is called when objects are added and removed from the active object map. Only used when POA has RETAIN policy. This interface is a per-POA interceptor which may be installed by a `POALifeCycleInterceptor` when the POA is created.

## Import statement

---

The import statements `import com.inprise.vbroker.interceptor.*` and `com.inprise.vbroker.PortableServerExt.*;` should be included in your code.

## ActiveObjectLifeCycleInterceptor methods

---

public void **create** (byte[] **oid**, org.omg.PortableServer.Servant **servant**,  
org.omg.PortableServer.POA **adapter**);

This method is invoked after an object has been added to the Active Object Map, either through explicit or implicit activation, using either direct APIs or a `ServantActivator`. The object reference and the POA of the new active object are passed as parameters.

Parameter	Description
<code>oid</code>	The object ID of the servant being activated.
<code>servant</code>	The servant being activated.
<code>adaptor</code>	The Portable Object Adaptor (POA) on which the servant was activated.

---

public void **destroy** (byte[] **oid**, org.omg.PortableServer.Servant **servant**,  
org.omg.PortableServer.POA **adapter**);

This method is called after an object has been deactivated and etherealized. The object reference and the POA of the object are passed as parameters.

Parameter	Description
<code>oid</code>	The object ID of the servant being destroyed.
<code>servant</code>	The servant being destroyed.
<code>POAadaptor</code>	The Portable Object Adaptor (POA) on which the servant was destroyed.

---

# ActiveObjectLifeCycleInterceptorManager

---

public interface **ActiveObjectLifeCycleInterceptorManager**

This is the interface used to add `ActiveObjectLifeCycleInterceptor`.

## Import statement

---

The import statements `import com.inprise.vbroker.interceptor.*` and `com.inprise.vbroker.PortableServerExt.*`; should be included in your code.

## ActiveObjectLifeCycleInterceptorManager method

---

public void **add** (ActiveObjectLifeCycleInterceptor interceptor);

This method is invoked by the ORB to add a `ActiveObjectLifeCycleInterceptor`.

## ForwardRequestException

---

public interface **ForwardRequestException** extends org.omg.CORBA.UserException,

This exception can be raised by `ServerRequestInterceptor`'s `preinvoke` method. The `preinvoke` method can raise this exception to forward a request to another object.

## Variables

---

public boolean **is\_permanent**

Specifies if the location forwarding is permanent.

public org.omg.CORBA.Object **forward\_reference**

Provides a reference to the object to which the request gets forwarded.

## ServerRequestInterceptor

---

public interface `Interceptor` **ServerRequestInterceptor**

The `ServerRequestInterceptor` interface is a per-POA interceptor which may be installed by a `POALifeCycleInterceptor` at POA creation time. This interface may be used to perform access control, to examine and insert service contexts, and to change the reply status of a request.

## Import statement

---

The import statement `import com.inprise.vbroker.interceptor.*`; should be included in your code.

## ServerRequestInterceptor methods

---

```
public void preinvoke (org.omg.CORBA.Object target,
    java.lang.String operation, com.inprise.vbroker.IOP.ServiceContext [] service_contexts,
    com.inprise.vbroker.CORBA.portable.InputStream payload,
    com.inprise.vbroker.interceptor.Closure closure) raises (ForwardRequestException);
```

This method is invoked by the ORB on every request, as soon as the request arrives on the server side. An exception thrown from this interceptor will result in the request being completed immediately. This method is called before any Servant Locators are invoked. Due to this, the servant may not be available.

Parameter	Description
<code>target</code>	The object on which the request is invoked.
<code>String operation</code>	Identifies the name of the operation being invoked.
<code>service_contexts</code>	Identifies the services assigned by the Orb. These services are registered with the OMG.
<code>payload</code>	Marshaled buffer.
<code>closure</code>	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

---

```
public void postinvoke_premarshal (org.omg.CORBA.Object target,
    com.inprise.vbroker.IOP.ServiceContextListHolder service_contexts,
    org.omg.CORBA.Environment env,
    com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked after an upcall to the servant but before marshalling the reply. An exception here is handled by interrupting the chain and `exception_occured()` is called on all interceptors in the chain.

Parameter	Description
<code>target</code>	The object on which the request is invoked.
<code>service_contexts</code>	Identifies the services assigned by the Orb. These services are registered with the OMG.
<code>env</code>	Contains information on the exception that was raised.
<code>closure</code>	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

---

```
public void postinvoke_postmarshal (org.omg.CORBA.Object target,
    com.inprise.vbroker.CORBA.portable.OutputStream payload,
    com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked after marshalling the reply but before sending the reply to the client. Exceptions thrown here are ignored. The entire chain is guaranteed to be called.

It is called after the ServantLocator has been invoked. Any exception thrown in this method will replace the exception thrown by the application program or, in one-way calls, after the request was successfully sent.

Parameter	Description
target	The object on which the request is invoked.
payload	Marshaled buffer.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

```
public void exception_occurred (org.omg.CORBA.Object target,
    org.omg.CORBA::Environment env,
    com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked if an exception is raised at any point of request processing. An exception thrown during this call should replace the existing exception in the environment.

Parameter	Description
target	The object on which the request is invoked.
env	Contains information on the exception that was raised.
closure	May contain data saved by one interceptor method that can be retrieved later by another interceptor method.

## ServerRequestInterceptorManager

```
public interface ServerRequestInterceptorManager extends com.inprise.vbroker.interceptor.
    InterceptorManager
```

This interface is used to add ServerRequestInterceptors.

### Import statement

The import statement `import com.inprise.vbroker.interceptor.*;` should be included in your code.

### ServerRequestInterceptorManager method

```
public void add (com.inprise.vbroker.interceptor.ServerRequestInterceptor interceptor);
```

This method is invoked by the ORB to add a `ServerRequestInterceptor`.

# IORCreationInterceptor

---

public interface PortableServerExt::**IORInterceptor**

The IORCreationInterceptor is a per-POA interceptor which may be installed by a POALifecycleInterceptor at POA creation time. The interceptor may be used to modify IORs by adding additional profiles or components. This interface is typically used to support services such as transactions or firewall.

In addition to the interceptor, you may also change the per-POA IOR template which controls all IORs for that POA. See “IOR templates” on page 12-2 for more information. You may prefer this IOR template approach if the IOR manipulations are not related to the repository or the OID of the reference being created.

In addition, making radical changes to the IOR is not recommended.

## Import statement

---

The import statements **import com.inprise.vbroker.interceptor.\*** and **com.inprise.vbroker.PortableServerExt.\*;** should be included in your code.

## IORInterceptor method

---

public void **create** (org.omg.PortableServer.POA **poa**,  
com.inprise.vbroker.IOP.IORValueHolder **ior**);

The method is called whenever the POA needs to create an object reference. The interceptor may modify the IORValue by adding additional profiles or components, or changing the existing profiles or components.

Parameter	Description
<code>poa</code>	The POA on which the object reference is being created.
<code>ior</code>	The IOR holder in which you can change the IOR's profiles or components.

---

# IORInterceptorManager

---

public interface PortableServerExt::**IORInterceptorManager** extends **InterceptorManager**

This interface is used to add IORInterceptors.

## Import statement

---

The import statements **import com.inprise.vbroker.interceptor.\*** and **com.inprise.vbroker.PortableServerExt.\*;** should be included in your code.

## IORInterceptorManager method

---

public void **add** (IORCreationInterceptor interceptor);

This method is invoked by the ORB to add an IORInterceptor.

## Location

---

enum **Location**

This enum describes the location, client-side or server-side, where an object wrapper should be registered.

## Closure

---

public interface **Closure** extends Object

Closure objects are created by the ORB at the beginning of certain sequences of interceptor calls. The same Closure object is used for all calls in that particular sequence. The Closure object contains a single public data field, object, of type `java.lang.Object` which may be set by the interceptor to keep state information. The sequences for which Closure objects are created vary depending on the interceptor type.

**Code sample 12.1** Closure class

```
class Closure {
    java.lang.Object object;
};
```

## ExtendedClosure

---

```
public interface ExtendedClosure extends Closure {
    public RequestInfo reqInfo;
    public InputStream payload;
}
```

This interface is a derived class of Closure and contains a RequestInfo for read only attribute.

**IDL sample 12.1** RequestInfo

```
struct RequestInfo {
    boolean response_expected;
    unsigned long request_id;
};
```

You can cast the Closure object passed to the ServerRequestInterceptor and ClientRequestInterceptor to its subclass, ExtendedClosure. ExtendedClosure can be used to extract the RequestInfo, from which you can extract the request\_id and



response\_expected. The request\_id is the unique id assigned to the request. The response\_expected flag indicates whether the request is a one-way call.

```
int my_response_expected = ((ExtendedClosure)closure).reqInfo.response_expected;
int my_request_id = ((ExtendedClosure)closure).reqInfo.request_id;
```

For more information, please see the example in `examples/interceptor/client_server`.

**Note** If you want to modify the `InputStream`, you *must* use the `payload` parameter of the `ExtendedClosure`. The `payload` attribute of the request interceptor is read-only; it does not allow you to change the `InputStream`.

For this reason, `ExtendedClosure` provides a read-write `InputStream` `payload` parameter. The main purpose of the `payload` attribute is to allow a new `InputStream` to be used in place of the old one.

The example `examples/interceptor/encryption` shows how to use `ExtendedClosure`'s `payload` attribute. In this example, when the interceptor tries to decrypt the data in an encrypted `InputStream`, a new `InputStream` containing the decrypted message needs to be created. `ExtendedClosure` serves as a holder for the `InputStream`. When the `payload` is assigned to the newly-created `InputStream`, that `InputStream` becomes the `InputStream` associated with the request.

## ChainUntypedObjectWrapperFactory

---

public interface **ChainUntypedObject WrapperFactory** extends `com.inprise.vbroker.interceptor.UntypedObjectWrapperFactory`;

This interface is used by a client or server application to add or remove an `UntypedObjectWrapperFactory` object. An `UntypedObjectWrapperFactory` object is used to create an `UntypedObjectWrapper` for each object a client application binds to or for each object implementation created by a server application.

```
};

enum Location (CLIENT, SERVER, BOTH );

abstract interface interface ChainUntypedObjectWrapperFactory :
    UntypedObjectWrapperFactory (

    void add(UntypedObjectWrapperFactory owFactory, Location loc);
    void remove(UntypedObjectWrapperFactory owFactory, Location loc);
    long count(Location loc);
};
```

See the *VisiBroker for Java Programmer's Guide* for complete information on using object wrappers.

### Import statement

---

The import statement `import com.inprise.vbroker.interceptor.*;` should be included in your code.

## ChainUntypedObjectWrapperFactory methods

---

public void **add** (com.inprise.vbroker.interceptor.UntypedObjectWrapperFactory **owFactory**,  
com.inprise.vbroker.interceptor.Location **loc**);

This method adds the specified untyped object wrapper factory for a client or server application.

Note On the client side, untyped object wrapper factories must be installed before any objects are bound. On the server side, untyped object wrapper factories must be installed before any implementation objects are created.

Parameter	Description
owFactory	The object wrapper factory to be added.
loc	The location where the object wrapper is to be added.

public void **remove**(com.inprise.vbroker.interceptor.UntypedObjectWrapperFactory **owFactory**,  
com.inprise.vbroker.interceptor.Location **loc**);

This method removes the specified untyped object wrapper factory for a client or server application.

Note Removing one or more object wrapper factories from a client will not affect objects of the class that are already bound by the client. Only subsequently bound objects will be affected. REMoving object wrapper factories from a server will not affect object implementations that have already been created. Only subsequently created object implementations will be affected.

Parameter	Description
owFactory	The object wrapper factory to be added.
loc	The location from which the object wrapper it to be removed.

long **count** (com.inprise.vbroker.interceptor.Location **loc**);

This method is called to add an untyped object wrapper factory.

Parameter	Description
loc	The location for which count is required.

## UntypedObjectWrapper

---

public interface **UntypedObjectWrapper**

You use this interface to derive untyped object wrappers that you wish to use for your client or server applications. When you derive an untyped object wrapper from this interface, you define a `pre_method` method that is invoked before an operation request is issued by a client application or before it is processed by an object implementation on the server-side. You also define a `post_method` method that will be invoked after an operation request is processed

by an object implementation on the server-side or after an reply has been received by a client application.

You must also derive a factory class from the `UntypedObjectWrapperFactory` interface, described in “`UntypedObjectWrapperFactory`” on page 12-18, that will create your `UntypedObjectWrapper` objects.

See the *VisiBroker for Java Programmer’s Guide* for complete information using object wrappers.

```
interface UntypedObjectWrapper {
    void pre_method(
        in string operation,
        in Object target,
        in interceptor Closure closure
    );
    void post_method(
        in string operation,
        in Object target,
        in CORBA Environment env,
        in interceptor Closure closure
    );
};
```

## UntypedObjectWrapper methods

---

```
public void pre_method(java.lang.String operation,
    org.omg.CORBA.Object target,
    com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked before an operation request is sent on the client-side or before it is processed by an object implementation on the server side.

Parameter	Description
<code>operation</code>	The name of the operation being requested.
<code>target</code>	The object that is the target of the request.
<code>closure</code>	The closure object that can be used to pass data between object wrapper method.

---

```
public void post_method(java.lang.String operation,
    org.omg.CORBA.Object target,
```

```
org.omg.CORBA.Environment env,  
com.inprise.vbroker.interceptor.Closure closure);
```

This method is invoked after an operation request has been processed by the object implementation on the server-side or before the reply message is processed by the stub on the client side.

Parameter	Description
operation	The name of the operation being requested.
target	The object that is the target of the request.
env	An <code>Environment</code> object that is used to reflect exceptions that might have occurred in the processing of the operation request.
closure	The closure object that can be used to pass data between object wrapper method.

## UntypedObjectWrapperFactory

---

public interface **UntypedObjectWrapperFactory**

You use this interface to derive your own untyped object wrapper factories. You register your untyped object wrapper factories using the `add` method, offered by the `ChainUntypedObjectWrapperFactory` interface.

Your factory is used to create an instance of your untyped object wrapper for a client or server application whenever a new object is bound or an object implementation is created.

```
interface UntypedObjectWrapperFactory {  
    interceptor UntypedObjectWrapper create(  
        in Object object, location loc(  
    );  
};
```

### Import statement

---

The import statement `import com.inprise.vbroker.interceptor.*;` should be included in your code.

### UntypedObjectWrapperFactory method

---

```
public com.inprise.vbroker.interceptor.UntypedObjectWrapper create (org.omg.CORBA.Object obj,  
com.inprise.vbroker.interceptor.Location loc);
```

This method is called to create an instance of your type of `UntypedObjectWrapper`. Your implementation of this method can examine the

type of bound object or object implementation to determine whether or not it wants to create an object wrapper for that object.

Parameter	Description
obj	The object being bound by a client application for which the untyped object wrapper is being created. If this method is being invoked on the server-side, this represents the object implementation that is being created.
loc	The location where the object wrapper is to be created.



## Quality of Service interfaces and classes

This chapter describes the VisiBroker implementation of the Quality of Service (QoS) APIs. The QoS APIs allow you to use policies to define and manage the connection between your client applications and the servers to which they connect. See “PortableServer.POA” on page 5-26 for information about creating policies.

QoS provides the following classes to manage ORB-, thread-, and Object-level policies:

- **ORB-level policies:** ORB-level policies are handled by a locality-constrained `PolicyManager`, through which you can set Policies and view the current Policy overrides. Policies set at the ORB level override system defaults.
- **Thread-level policies:** Thread-level policies are set through `PolicyCurrent`, which contains operations for viewing and setting Policy overrides at the thread level. Policies set at the thread level override system defaults and values set at the ORB level.
- **Object-level policies:** Object-level policies can be applied by accessing the base Object interface’s quality of service operations. Policies applied at the Object level override system defaults and values set at the ORB or thread level.

### PolicyManager

---

public interface `org.omg.CORBA.PolicyManager`

The `PolicyManager` interface is used to access policy overrides at the ORB level.

## IDL definition

---

```
module CORBA {
    ...
    interface PolicyManager {
        PolicyList get_policy_overrides(in PolicyTypeSeq ts);
        void set_policy_overrides(in Policy[] policies, in SetOverrideType set_add)
            raises (InvalidPolicies);
    };
};
```

## Policy Manager methods

---

`Policy[] get_policy_overrides (int[]);`

This method returns a list of all the policies of the requested types. If the specified sequence is empty (that is, if the length of the list is zero), all Policies at this scope are returned. If none of the requested policy types is set at the target `PolicyManager`, an empty sequence is returned.

`void set_policy_overrides (Policy[] policies, SetOverrideType set_add) throws InvalidPolicy`

This method updates the current set of policy overrides with the requested list. Invoking `set_policy_overrides` with an empty sequence of policies and a mode of `SET_OVERRIDE` removes all overrides from a `PolicyManager`. Only certain policies that pertain to the invocation of an operation at the client end can be overridden using this operation. Attempts to override any other policy will raise the `CORBA.NO_PERMISSION` exception. If the request would put the set of overriding policies for the target `PolicyManager` in an inconsistent state, no policies are changed or added, and the exception `InvalidPolicies` is raised. There is no evaluation of compatibility with policies set within other `PolicyManagers`.

Parameter	Description
policies	A sequence of references to Policy objects.
set_add	A parameter of type <code>org.omg.CORBA.SetOverrideType</code> that indicates whether these policies should be added ( <code>ADD_OVERRIDE</code> ) to any other overrides that already exist in the <code>PolicyManager</code> , or added to a clean <code>PolicyManager</code> free of any other overrides ( <code>SET_OVERRIDE</code> ). If the request would cause the specified <code>PolicyManager</code> to be in an inconsistent state, no policies are changed or added, and an <code>InvalidPolicies</code> exception is raised.

---



# PolicyCurrent

---

**public interface** org.omg.CORBA.PolicyCurrent **extends** org.omg.CORBA.PolicyManager, org.omg.CORBA.Current

The `PolicyCurrent` interface derives from `PolicyManager` and `Current` without adding new methods. Therefore all operations on the `PolicyManager` interface are also available in `PolicyCurrent`. See “`PolicyManager`” on page 13-1 for a description of these methods.

`PolicyCurrent` provides access to the policies overridden at the thread level. A reference to a thread’s `PolicyCurrent` is obtained by invoking `org.omg.CORBA.ORB.resolve_initial_references` and specifying an identifier of `PolicyCurrent`.

## IDL definition

---

```
interface PolicyCurrent : PolicyManager, Current {};
```

# Object

---

**public interface** org.omg.CORBA.**Object**

Because the CORBA 2.3 specification contains only limited support for QoS, VisiBroker extended `org.omg.CORBA.Object` to provide additional QoS support as defined in the OMG Messaging specification. This means that there are two exposed `Object` interfaces. To use this additional functionality, cast `org.omg.CORBA.Object` to `com.inprise.vbroker.CORBA.Object`. Because the additional methods defined in the Messaging specification were not yet available in CORBA 2.3, these methods are added to `com.inprise.vbroker.CORBA.Object` to provide QoS functionality.

For more information, see Chapter 5, “Quality of Service,” of the OMG’s *CORBA Messaging specification (orbos/98-05-05)*, and Chapter 3, “OMG IDL Syntax and Semantics,” of the *CORBA 2.3 specification (orb/98-12-01)*.

## org.omg.CORBA.Object methods

---

**public interface** org.omg.CORBA.**Object**, org.omg.CORBA.Policy **\_get\_policy**(int type);

Returns the effective Policy for the object reference. The effective Policy is the one that would be used if a request were made. This Policy is determined first by obtaining the effective override for the PolicyType as returned by `_get_client_policy`.

The effective override is then compared with the Policy as specified in the IOR. The effective Policy is the intersection of the values allowed by the effective override and the IOR-specified Policy. If the intersection is empty, the system exception `INV_POLICY` is raised. Otherwise, a Policy with a

value legally within the intersection is returned as the effective Policy. The absence of a Policy value in the IOR implies that any legal value may be used. Invoking `non_existent()` or `_validate_connection` on an object reference prior to `_get_policy` ensures the accuracy of the returned effective Policy. If `_get_policy` is invoked prior to the object reference being bound, the returned effective Policy is implementation dependent. In that situation, a compliant implementation may do any of the following: raise the exception `CORBA.BAD_INV_ORDER`, return some value for that `PolicyType` which may be subject to change once a binding is performed, or attempt a binding and then return the effective Policy. Note that if the `RebindPolicy` has a value of `TRANSPARENT`, `VB_TRANSPARENT`, or `VB_NOTIFY_REBIND`, the effective Policy may change from invocation to invocation due to transparent rebinding.

`org.omg.CORBA.Object _set_policy_overrides(const Policy[] _policies, SetOverrideType _set_add)`  
throws `org.omg.CORBA.InvalidPolicy`;

This method returns a new object reference with the requested list of Policy overrides at the object level, and works in a way similar to the `org.omg.CORBA.PolicyManager` method of the same name. However, it updates the current set of policies of an Object, thread, or ORB with the requested list of Policy overrides. In addition, this method returns an `org.omg.CORBA.Object` whereas other methods of the same name return `void`.

## com.inprise.vbroker.CORBA.Object methods

---

public interface `com.inprise.vbroker.CORBA.Object` extends `org.omg.CORBA.Object`,  
`org.omg.CORBA.Policy _get_client_policy(int type)`;

`_get_client_policy` returns the effective overriding Policy for the object reference without doing the intersection with the server-side policies. The effective override is obtained by first checking for an override of the given `PolicyType` at the Object scope, then at the Current scope, and finally at the ORB scope. If no override is present for the requested `PolicyType`, the system-dependent default value for that `PolicyType` is used. Portable applications are expected to set the desired “defaults” at the ORB scope since default Policy values are not specified.

`org.omg.CORBA.Policy[] _get_policy_overrides(int[] types)`;

`_get_policy_overrides` returns a list of Policy overrides of the specified policy types set at the object level. If the specified sequence is empty, all overrides at the object level are returned. If no `PolicyTypes` are overridden at the object level, an empty sequence is returned.

`boolean _validate_connection(org.omg.CORBA.PolicyListHolder inconsistent_policies)`;

`_validate_connection` returns a boolean value based on whether the current effective policies for the object will allow an invocation to be made. It returns the value `true` if the current effective policies for the Object allow an invocation to be made. If the object reference is not yet bound, a binding occurs as part of this operation. If the object reference is already bound, but

current policy overrides have changed or for any other reason the binding is no longer valid, a rebind is attempted regardless of the setting of any RebindPolicy override. The `_validate_connection` operation is the only way to force such a rebind when implicit rebinds are disallowed by the current effective RebindPolicy. The attempt to bind or rebind may involve processing GIOP LocateRequests by the ORB. Returns the value `false` if the current effective policies would cause an invocation to raise the system exception `INV_POLICY`. If the current effective policies are incompatible, the out parameter `inconsistent_policies` contains those policies causing the incompatibility. This returned list of policies is not guaranteed to be exhaustive. If the binding fails due to some reason unrelated to policy overrides, the appropriate system exception is raised.

## RebindPolicy

---

public interface `org.omg.Messaging.RebindPolicy`

The Visibroker implementation of `RebindPolicy` is a complete implementation of the `RebindPolicy` as defined in the CORBA 2.4 Specification with enhancements to support failover.

The `RebindPolicy` determines how the client-side ORB handles closed connections, GIOP location-forward messages and object failures. The ORB handles fail-overs, rebinds, and reconnections by looking at the effective policy at the `org.omg.CORBA.Object` instance.

The OMG-defined Policy values determine whether the ORB may transparently rebind once it is successfully bound to a target server. The extended policy values determine whether the ORB may transparently failover once it is successfully bound to a target Object.

The `RebindPolicy` is a client-side-only policy.

**Note** The `RebindPolicy` is enforced only after being successfully bound to an object. For GIOP-based protocols an object reference is considered bound once it is in a state where a `LocateRequest` message would result in a `LocateReply` message with status `OBJECT_HERE`.

It can have one of six values that determines the behavior in the case of a disconnection, an object-forwarding request, or an object failure. The currently supported values are:

- `org.omg.Messaging.TRANSPARENT` allows the ORB to silently handle object-forwarding and necessary reconnections during the course of making a remote request.
- `org.omg.Messaging.NO_REBIND` allows the ORB to silently handle reopening of closed connections while making a remote request, but prevents any transparent object-forwarding that would cause a change in client-visible effective QoS policies. When `RebindMode` is set to `NO_REBIND`, only explicit rebind is allowed.

- `org.omg.Messaging.NO_RECONNECT` prevents the ORB from silently handling object-forwards or the reopening of closed connections. You must explicitly rebind and reconnect when `RebindMode` is set to `NO_RECONNECT`.
- `com.inprise.vbroker.QoSExt.VB_TRANSPARENT` is the default value of this Policy. It extends the functionality of `TRANSPARENT` by allowing transparent rebinding with both implicit and explicit binding. `VB_TRANSPARENT` is designed to be compatible with the object failover implementation in VisiBroker 3.x.
- `com.inprise.vbroker.QoSExt.VB_NOTIFY_REBIND` throws an exception if a rebind is necessary. The client catches this exception, and binds on the second invocation.
- `qqq--VB_NO_REBIND`

Note Be aware that if the effective policy for your client is `VB_TRANSPARENT` and your client is working with servers that hold state data, `VB_TRANSPARENT` could connect the client to a new server without the client being aware of the change of server, any state data held by the original server will be lost.

For more information on QoS policies and types, see the Messaging chapter of the CORBA 2.4 specification. Our QoS implementation is based on the OMG document orbos/98-05-05.

## IDL definition

---

```
#pragma prefix "omg.org"

module Messaging {
    typedef short RebindMode;
    const CORBA::PolicyType REBIND_POLICY_TYPE = 23;
    interface RebindPolicy::CORBA::Policy {
        readonly attribute RebindMode rebind_mode;
    };
}
```

## Policy Values

---

Table 13.1, “OMG Policy values,” describes the OMG Policy values that can be set as the `RebindPolicy`:

**Table 13.1**    OMG Policy values

Policy Value	Description
TRANSPARENT	This policy allows the ORB to silently handle object-forwarding and necessary reconnection when making a remote request. This is the least restrictive OMG policy value.

**Table 13.1** OMG Policy values (continued)

Policy Value	Description
NO_REBIND	This policy allows the ORB to silently handle reopening of closed connections while making a remote request, but prevents any transparent object-forwarding that would cause a change in the client-side effective QoS policies.
NO_RECONNECT	This policy prevents the ORB from silently handling object-forwards or the reopening of closed connections. This is the most restrictive OMG policy value.

The VisiBroker-specific values that can be set as the `RebindPolicy` are listed in the following table. For a detailed description of all policy behavior, see Table 13.3, “Behavior resulting from RebindMode policies.”

**Table 13.2** VisiBroker-specific RebindMode policies

Policy Value	Description
com.inprise.vbroker.QoSExt.VB_TRANSPARENT	<p>This policy extends <code>TRANSPARENT</code> behavior to failover. This is the default policy.</p> <p>When this policy is set, if a remote invocation fails because the server object goes down, the ORB tries to reconnect to another server using the osagent. The client ORB masks the communication failure if the rebind succeeds, and does not throw an exception to the calling thread.</p>
com.inprise.vbroker.QoSExt.VB_NOTIFY_REBIND	<p><code>VB_NOTIFY_REBIND</code> behaves like <code>VB_TRANSPARENT</code>, but throws an exception when the communication failure is detected. It will try to transparently reconnect to another object if another invocation is attempted.</p>
com.inprise.vbroker.QoSExt.VB_NO_REBIND	<p><code>VB_NO_REBIND</code> does not enable failover. It only allows the client ORB to reopen a closed connection to the same server; it does not allow object forwarding of any kind.</p>

The following table lists the behavior of the different `RebindMode` policies:

**Table 13.3** Behavior resulting from RebindMode policies

RebindMode type	Reestablish closed connection to the same object?	Allow object forwarding?	Object failover? <sup>1</sup>
NO_RECONNECT	No, throws <code>REBIND</code> exception.	No, throws <code>REBIND</code> exception.	No
VB_NO_REBIND	Yes	No	No

**Table 13.3** Behavior resulting from RebindMode policies (continued)

RebindMode type	Reestablish closed connection to the same object?	Allow object forwarding?	Object failover? <sup>1</sup>
NO_REBIND	Yes	Yes, if policies match No, throws REBIND exception.	No
TRANSPARENT	Yes	Yes	No
VB_NOTIFY_REBIND	Yes	Yes	Throws an exception after failure detection and then tries a failover on subsequent requests.
VB_TRANSPARENT	Yes	Yes	Yes, transparently.

1. The appropriate CORBA exception will be thrown in the case of a communication problem or an object failure.

## RelativeConnectionTimeoutPolicy

**public interface** com.inprise.vbroker.QoSExt.**RelativeConnectionTimoutPolicy**

The `RelativeConnectionTimeoutPolicy` indicates a timeout after which attempts to connect to an object using one of the available endpoints is aborted. The timeout situation is likely to happen with objects protected by firewalls, where HTTP tunneling is the only way to connect to the object.

**Note:** This Policy is not enforced for in-process communications.

The policy value is an integer that specifies the timeout in 100s of nanoseconds. It is applied to every endpoint that the ORB tries to connect to. Therefore, if multiple connection attempts are made, the elapsed time will be a multiple of the configured timeout. The accuracy is also limited by the Java virtual machine implementation.

### IDL definition

```
#pragma prefix "inprise.com"

module QoSExt{
  const CORBA::PolicyType RELATIVE_CONN_TIMEOUT_POLICY_TYPE = 0x56495304
  interface RelativeConnectionTimeoutPolicy : CORBA::Policy {
    readonly attribute TimeBase::TimeT relative_expiry;
  };
};
```

# DeferBindPolicy

---

**public interface** com.inprise.vbroker.QoSExt.DeferBindPolicy

The DeferBindPolicy determines if the ORB will attempt to contact the remote object when it is first created, or delay this contact until the first invocation is made. By default, the ORB connects to the (remote) object when on a `bind()` or a `string_to_object` call. If the DeferBindPolicy is set to `true`, the ORB does not contact the remote Object until the first invocation.

If you create a client object, and DeferBindPolicy is set to `true`, you may delay the server startup until the first invocation. This option existed before as an option to the `Bind` method on the generated helper classes.

## IDL definition

---

```
#pragma prefix "inprise.com"

module QoSExt{
const CORBA::PolicyType DEFER_BIND_POLICY_TYPE = 0x56495305
interface DeferBindPolicy: CORBA::Policy {
    readonly attribute boolean value:
};
```

# ExclusiveConnectionPolicy

---

**public interface** com.inprise.vbroker.QoSExt.ExclusiveConnectionPolicy:CORBA::Policy

The ExclusiveConnectionPolicy is a Visibroker-specific policy that gives you the ability to establish an exclusive (non-shared) connection to a specified server object. This policy has boolean value of `true` or `false`. A `true` value will open an exclusive connection to the server object. A `false` value will try to reuse an existing connection if possible and open a new connection only if reuse is not possible. The default value is `false`.

This policy provides the same features as `Object._clone()` in Visibroker 3.x.

## IDL definition

---

```
interface ExclusiveConnectionPolicy :CORBA::Policy {
    /** Returns the current setting of the ExclusiveConnectionPolicy */
    readonly attribute boolean value;
};
```

# SyncScopePolicy

---

**public interface** org.omg.CORBA.SyncScopePolicy:CORBA::Policy qq--is this right??

This interface is a local object derived from CORBA::Policy. It is applied to oneway operations to indicate the synchronization scope with respect to the target of that operation request. It is ignored when any non-oneway operation is invoked. This policy is also applied when the DII is used with a flag of INV\_NO\_RESPONSE since the implementation of the DII is not required to consult an interface definition to determine if an operation is declared oneway. The default value of this Policy is SYNC\_TRANSPORT. Applications must explicitly set the SyncScopePolicy to ensure portability across ORB implementations. When instances of SyncScopePolicy are created, a value of type Messaging::SyncScope is passed to CORBA::ORB::create\_policy. This policy is only applicable as a client-side override.

## IDL definition

---

```
interface SyncScopePolicy :CORBA::Policy {
    readonly attribute SyncScope synchronization;
};
```

The following table lists the behavior of the different SyncScope policy values.

**Table 13.4** SyncScope policy values

SyncScope type	Description
SYNC_NONE	The ORB returns control to the client (e.g. returns from the method invocation) before passing the request message to the transport protocol. The client is guaranteed not to block. Since no reply is returned from the server, no location-forwarding can be done with this level of synchronization.



**Table 13.4** SyncScope policy values

SyncScope type	Description
SYNC_WITH_SERVER	The server-side ORB is to send a reply before invoking the target implementation. If a reply of <code>NO_EXCEPTION</code> is sent, any necessary location-forwarding has already occurred. Upon receipt of this reply, the client-side ORB shall return control to the client application. The client blocks until all location-forwarding has been completed. For a server using a POA, the reply would be sent after invoking any <code>ServantManager</code> , but before delivering the request to the target Servant.
SYNC_WITH_TARGET	Equivalent to a synchronous, non-oneway operation in CORBA 2.2. The server-side ORB will only send the reply message after the target has completed the invoked operation. Note that any <code>LOCATION_FORWARD</code> reply will already have been sent prior to invoking the target and that a <code>SYSTEM_EXCEPTION</code> reply may be sent at anytime (depending on the semantics of the exception). Even though it was declared oneway, the operation actually behaves like a synchronous operation. This form of synchronization guarantees that the client knows that the target has seen and acted upon a request. As with CORBA 2.2, only with this highest level of synchronization can the OTS be used. Any operations invoked with lesser synchronization precludes the target from participating in the client's current transaction.

## QoS exceptions

Table 13.5, “QoS exceptions” lists possible QoS exceptions

**Table 13.5** QoS exceptions

Exception	Description
<code>org.omg.CORBA.INV_POLICY</code>	Raised when there is an incompatibility between Policy overrides.
<code>org.omg.CORBA.REBIND</code>	Raised when the <code>RebindPolicy</code> has a value of <code>NO_REBIND</code> , <code>NO_RECONNECT</code> , or <code>VB_NOTIFY_REBIND</code> and an invocation on a bound object references results in an object-forward or location-forward message.
<code>org.omg.CORBA.PolicyError</code>	Raised when the requested Policy is not supported.



## IOP and IIOP interfaces and classes

This chapter describes the VisiBroker implementation of the key General Inter-ORB Protocol interfaces and other structures defined by the CORBA specification. For a complete description of these interfaces, refer to Chapter 15 of the *OMG CORBA Specification*.

### IIOP.ProfileBody

---

public class **ProfileBody**

This class contains information about the protocol supported by an object.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

#### IDL definition

---

```
struct ProfileBody {  
    ::GIOP::Version iiop_version;  
    string host;  
    unsigned short port;  
    ::CORBA::OctetSequence object_key;  
    sequence<::IOP::TaggedComponent> components;  
};
```

## IIOP.ProfileBody variables

---

`public com.inprise.vbroker.GIOP.Version iiop_version;`

Represents the version of IIOP supported.

`public java.lang.String host;`

Represents the name of the host where the object is implemented.

`public short port;`

Indicates the port number to use for establishing a connection to the object.

`public byte[] object_key;`

Used to uniquely identify the object reference and contains information used to locate the servant that implements the object. The object key is stored in a vendor-specific format and is generated when an IOR is created.

`public com.inprise.vbroker.IOP.TaggedComponent[] components;`

A sequence of zero or more `TaggedComponent` objects used to hold additional information that may be used in making invocations on the object described by this profile.

## IIOP.ProfileBody constructors

---

`public ProfileBody()`

Creates an empty `ProfileBody`.

`public ProfileBody(com.inprise.vbroker.GIOP.Version iiop_version, java.lang.String host, short port, byte[] object_key, com.inprise.vbroker.IOP.TaggedComponent[] components)`

Creates a `ProfileBody` initialized with the specified IIOP version, host name, port number, object key, and components.

Parameter	Description
<code>iiop_version</code>	The version of IIOP supported.
<code>host</code>	The host where the object implemented.
<code>port</code>	The port number to use when establishing a connection to the object.
<code>object_key</code>	The object's key.
<code>components</code>	A sequence of zero or more <code>TaggedComponent</code> objects used to hold additional information that may be used in making invocations on the object described by this profile.

---

## IIOP.IORValue

---

`public class IORValue`

This class represents an Interoperable Object Reference and is used to provide important information about object references. Your client application can

create an IOR by invoking the `ORB::object_to_string` method described under the syntax statement “abstract public java.lang.String `object_to_string(org.omg.CORBA.Object obj)`” on page 5-20.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
valuetype IORValue {
    public string type_id;
    public ProfileValueSeq profiles;
    IOR toIOR();
    IORValue copy();
    boolean matchesTemplate (in IORValue iorv);
};
```

## IIOP.IORValue variables

---

public java.lang.String **type\_id**;

Describes the type of object reference that is represented by this IOR.

public com.inprise.vbroker.IOP.ProfileValue[] **profiles**;

Represents a sequence of one or more `TaggedProfile` objects, which contain information about the protocols that are supported.

public com.inprise.vbroker.IOP.IOR **toIOR**();

Converts to an IOR.

public com.inprise.vbroker.IOP.IORValue **copy**();

Makes a duplicate copy of the `IORValue`.

public boolean **matchesTemplate** (IORValue **iorv**);

Checks to see if the `IORValue` matches the template `IORValue`.

## IOR.ServiceContext

---

public class **ServiceContext**

This class represents service-specific context information that is passed along with a request or reply.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
struct ServiceContext {
    ::IOP::ServiceID context_id;
    ::CORBA::OctetSequence context_data;
};
```

## IOP.ServiceContext variables

---

public int **Context\_id**;

Identifies a particular service and data format.

public byte[] **context\_data**;

The context data associated with the particular service identified by the `Context_id`. The context data is encoded in a service-specific format and then encapsulated as a sequence of octets.

## IOP.ServiceContext constructors

---

public **ServiceContext**()

Creates an empty `ServiceContext`.

public **ServiceContext**(int **context\_id**, byte[] **context\_data**)

Creates a `ServiceContext` initialized with the specified identifier and data.

Parameter	Description
context_id	Identifies a particular service and data format.
context_data	Service-specific data, encapsulated as a sequence of octets.

## IOP.TaggedProfile

---

public class **TaggedProfile**

This class represents a supported protocol and encapsulates all the basic information the protocol needs to identify an object.

`Helper` and `Holder` versions of this class are also provided. See Chapter 4, “Generated interfaces and classes,” for more information on these classes and the methods they offer.

## IDL definition

---

```
struct TaggedProfile {
    ::IOP::ProfileId tag;
    sequence <octet> profile_data;
};
```

## IIOP.TaggedProfile variables

---

public int **tag**;

Identifies the contents of the profile data and should be one of the following values:

Value	Description
TAG_INTERNET_IOP	Indicates the protocol is standard IIOP.
TAG_MUTIPLE_COMPONENTS	Indicates the profile data contains a list of ORB services available using the protocol.
TAG_VSGN_LOCATOR	Indicates that the IOR is an interim, pseudo-object that is used until the real IOR is received by the osagent.
TAG_LOCAL_IPC_IOP	Indicates the protocol is IOP over a local IPC mechanism.

---

public byte[] **profile\_data**;

Encapsulates all the protocol information needed to identify an object.

## IIOP.TaggedProfile constructors

---

public **TaggedProfile**()

Creates an empty `TaggedProfile`.

public **TaggedProfile**(int **tag**, byte[] **profile\_data**)

Creates a `TaggedProfile` initialized with the specified tag and data.

Parameter	Description
tag	Identifies the contents of the profile data and should be one of TAG_INTERNET_IOP TAG_MUTIPLE_COMPONENTS TAG_VSGN_LOCATOR TAG_LOCAL_IPC_IOP
profile_data	The protocol information needed to invoke an operation on an IOR

---





## RMI interfaces and classes

This chapter describes the interfaces used to support RMI over IIOP. VisiBroker currently does not support the server side programming model for RMI over RMI in VisiBroker 4.x. While the server-side APIs are documented here, using them may throw a `NO_IMPLEMENT` exception.

### PortableRemoteObject

---

```
public abstract class javax.rmi. PortableRemoteObject { }
```

This class is the base class for all server implementation. RMI-IIOP server implementations may inherit from `javax.rmi.PortableRemoteObject` or implement an RMI-IIOP remote interface and then use the `exportObject` method to register themselves as a server object. Clients should use the `narrow` method to narrow generic remote interfaces to specific remote interfaces.

### Constructors

---

```
public static void exportObject (Remote obj)
```

Makes a server object ready to receive remote calls. Subclasses of `PortableRemoteObject` do not need to call this method, as it is called by the constructor.

Parameter	Description
<code>obj</code>	The server object to be exported.

## PortableRemoteObject methods

---

protected PortableRemoteObject ()

This method initializes the object by calling `exportObject ()`.

public static Remote toStub (Remote obj)

This method returns a stub for the given server object. The server must be ready to receive remote communication, which may require the use of the `PortableRemoteObject.connect (Remote,Remote)` method, if the object has not yet been passed as an argument on a remote method call. This method throws a `java.rmi.NoSuchObjectException` exception if a stub cannot be located for the given server object.

Parameter	Description
obj	The server object for which a stub is required. Must be a subclass of <code>PortableRemoteObject</code> or have been previously the target of a call to <code>PortableRemoteObject.exportObject ()</code> .

public static void unexportObject (Remote obj)

This method unregisters a server object from the runtime, allowing the object to become available for a garbage collection. This method throws a `java.rmi.NoSuchObjectException` exception if the remote object is not currently exported.

Parameter	Description
obj	The object to be exported.

public static java.lang.Object narrow (java.lang.Object narrowFrom, java.lang.Class narrowTo)

This method narrows this RMI-IIOP object to a stub of the remote interface of a class `narrowTo`. This method throws a `ClassCastException` exception if `narrowFrom` cannot be cast to `narrowTo`.

Parameter	Description
narrowFrom	The object to be cast to type.
narrowTo	The desired type of object.

public static void connect (Remote unconnected, Remote connected)

This method makes a remote object ready for remote communication. This normally happens implicitly when the object is sent or received as an argument on a remote method call, but in some circumstances it is useful to perform this action by making an explicit call. This method throws a

`java.rmi.RemoteException` exception if the connected object is not connected or if the unconnected object is already connected.

Parameter	Description
unconnected	Identifies the object to be connected.
connected	Identifies the previously connected object.



## URL Naming interfaces and classes

This chapter describes the Resolver interface and classes used in VisiBroker's URL Naming Service.

**Note** In previous versions of VisiBroker for Java, the URL Naming Service was called the Web Naming Service.

### Resolver

---

public interface **Resolver** extends Object

When using the URL Naming service, a `Resolver` is called by the ORB's `resolve_initial_references`. For more information about using `Resolver`, see Chapter 29, "Using URL naming," in the *VisiBroker for Java Programmer's Guide*.

```
interface Resolver {  
  
    // Read Operations  
    Object locate(in string url_s)  
        raises (InvalidURL, CommFailure, ReqFailure);  
  
    // Write Operations  
    void force_register_url(in string url_s, in Object obj)  
        raises (InvalidURL, CommFailure, ReqFailure);  
  
    void register_url(in string url_s, in Object obj)  
        raises (InvalidURL, CommFailure, ReqFailure, AlreadyExists);  
};
```

# Resolver methods

---

Object **locate**(String **url\_s**)

This method is called transparently by the `bind()` method in the following situation: when client applications need to bind to the `Resolver`, they simply specify the URL when they call the `bind` method. If the URL is invalid, an `InvalidURL` exception is raised.

Parameter	Description
<code>url_s</code>	The URL's string.

---

void **force\_register\_url**(String **url\_s**, Object **obj**)

This method registers a server's object by associating its IOR (Interoperable Object Reference) with a URL (Uniform Resource Locator).

If you attempt to associate a URL with an object's IOR using the `force_register` method and a URL has already been bound to that object, the new URL binding replaces the old binding.

Parameter	Description
<code>url_s</code>	The URL's string.
<code>obj</code>	The object whose IOR will be associated with the URL.

---

void **register\_url**(String **url\_s**, Object **obj**)

This method registers a server's object by associating its IOR (Interoperable Object Reference) with a URL (Uniform Resource Locator).

If you attempt to associate a URL with an object's IOR using the `register` method and a URL has already been bound to that object, an `AlreadyExists` exception is raised.

Parameter	Description
<code>url_s</code>	The URL's string.
<code>obj</code>	The object whose IOR will be associated with the URL.

---

## Location Service interfaces and classes

This chapter provides detailed information about the Location Services's Agent and TriggerHandler interfaces which can be used to locate object instances on a network of Smart Agents. For more information about the Location Service, see Chapter 17, "Using the Location Service," in the VisiBroker for Java *Programmer's Guide*.

### Agent

---

public interface **Agent** extends com.inprise.vbroker.CORBA.Object

The Location Service Agent is a collection of methods that enable you to discover objects on a network of Smart Agents. Methods for the Agent interface can be divided into two groups: those that query a Smart Agent for data describing instances, and those that register and unregisters *triggers*. You can query based on the interface's repository ID, or based on a combination of the interface's repository ID and the instance name. Results of a query can be returned as either *object references* or more complete *instance descriptions*. Triggers are a notification mechanism by which clients of the Location Service can be notified of changes to the availability of instances.

### IDL definition

---

```
interface Agent {
    HostnameSeq all_agent_locations()
        raises(Fail);
    RepositoryIdSeq all_repository_ids()
        raises(Fail);
}
```

```
ObjSeq all_instances(in string repository_id)
    raises(Fail);
ObjSeq all_replica(in string repository_id,in string instance_name)
    raises(Fail);
DescSeq all_instances_descs(in string repository_id)
    raises(Fail);
DescSeq all_replica_descs(in string repository_id,in string instance_name)
    raises(Fail);
void reg_trigger(in TriggerDesc desc,in TriggerHandler handler)
    raises(Fail);
void unreg_trigger(inTriggerDesc desc,in TriggerHandler handler)
    raises(Fail);
};
```

## Agent methods

```
public java.lang.String[] all_agent_locations() throws com.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves a sequence of host names on which osagents reside. This method throws the following exception:

Exception	Description
Fail	Either there is no agent available or there is unsuccessful communications with the osagent.

```
public org.omg.CORBA.Object[] all_instances(java.lang.String repository_id) throws
com.inprise.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves object references to instances of an interface which satisfy the given repository id.

Parameter	Description
repository_id	A string containing a repository identifier.

This method throws the following exception:

Exception	Description
Fail	The repository id is invalid.

```
public com.inprise.vbroker.ObjLocation.Desc[] all_instances_descs(java.lang.String repository_id)
throws com.inprise.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves full description information for instances of the interface which implement the given repository id.

Parameter	Description
repository_id	A string containing a repository identifier.



This method throws the following exception:

Exception	Description
Fail	The repository id is invalid.

```
public org.omg.CORBA.Object[] all_replica(java.lang.String repository_id, java.lang.String instance_name) throws com.inprise.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves object references to like-named instances of the interface which satisfy the given repository id and instance name.

Parameter	Description
repository_id	A string containing a repository identifier.
instance_name	A string containing an instance name.

This method throws the following exception:

Exception	Description
Fail	Either the repository id or object name is invalid.

```
public com.inprise.vbroker.ObjLocation.Desc[] all_replica_descs(java.lang.String repository_id, java.lang.String instance_name) throws com.inprise.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves full description information for like-named instances of the interface which implement the given repository id and have the given instance name.

Parameter	Description
repository_id	A string containing a repository identifier.
instance_name	A string containing an instance name.

This method throws the following exception:

Exception	Description
Fail	Either the repository id or object name is invalid.

```
public java.lang.String[] all_repository_ids() throws com.inprise.inprise.vbroker.ObjLocation.Fail;
```

This method retrieves all interfaces known to any osagent. This method throws the following exception:

Exception	Description
Fail	The repository id is invalid.

public void **reg\_trigger**(com.inprise.vbroker.ObjLocation.TriggerDesc **desc**,  
com.inprise.vbroker.ObjLocation.TriggerHandler **handler**) throws  
com.inprise.inprise.vbroker.ObjLocation.Fail;

This method registers a trigger handler.

Parameter	Description
desc	Description of the instance. The instance description can contain combinations of the following instance information: repository ID, instance name, and host name. The more instance information provided, the more particular your specification of the instance.
handler	The TriggerHandler object you want to register.

This method throws the following exception:

Exception	Description
Fail	There is no such trigger.

public void **unreg\_trigger**(com.inprise.vbroker.ObjLocation.TriggerDesc **desc**,  
com.inprise.vbroker.ObjLocation.TriggerHandler **handler**) throws  
com.inprise.inprise.vbroker.ObjLocation.Fail;

This method unregisters a trigger handler.

Parameter	Description
desc	Description of the instance. The instance description can contain combinations of the following instance information: repository ID, instance name, and host name. The more instance information provided, the more particular your specification of the instance.
handler	The TriggerHandler object you want to unregister.

Note

Triggers are “sticky.” A `TriggerHandler` is invoked every time an object satisfying the trigger description becomes accessible. You may only be interested in learning when the first instance becomes accessible; in this case, invoke the `Agent’s unreg_trigger()` method to unregister the trigger after the first occurrence is found.

This method throws the following exception:

Exception	Description
Fail	There is no such trigger.

## Desc

public interface **Desc**

This interface contains information you use to describe the characteristics of an object. You pass this structure as an argument to several of the Location Service

methods described in the chapter. The `Desc` structure, or a sequence of them, is returned by some of the Location Service methods.

## IDL definition

---

```
struct Desc {
    CORBA::Object ref;
    ::IIOP::ProfileBodyValue iiop_locator;
    string repository_id;
    string instance_name;
    boolean activable;
    string agent_hostname;
};
```

## Desc variables

---

public boolean **activable**;

If `true`, this object is registered with the Object Activation Daemon. If `false`, the object was started manually and is registered with the osagent.

public org.omg.CORBA.Object **ref**;

A reference to the object being described.

public com.inprise.vbroker.IIOP.ProfileBodyValue **iiop\_locator**;

A reference to the object being described.

public java.lang.String **repository\_id**;

The object's repository identifier.

public java.lang.String **instance\_name**;

The object's instance name.

public java.lang.String **agent\_hostname**;

The name of the host running the Smart Agent with which this object is registered.

## Desc constructor

---

```
public Desc( org.omg.CORBA.Object ref, com.inprise.vbroker.IIOP.ProfileBodyValue iiop_locator,
    java.lang.String repository_id, java.lang.String instance_name, boolean activable,
    java.lang.String agent_hostname)
```

Creates a `Desc` object, initialized with the specified parameters.

Parameter	Description
<code>ref</code>	A reference to the object being described.
<code>iiop_locator</code>	A reference to the object being described.

Parameter	Description
repository_id	The object’s repository identifier.
instance_name	The object’s instance name.
activable	Set to <code>true</code> if this object is registered with the Object Activation Daemon. It is set to <code>false</code> if the object was started manually and is registered with the <code>osagent</code> .
agent_hostname	The name of the host running the Smart Agent with which this object is registered.

## Desc methods

```
public java.lang.String toString()

    Returns a string containing the contents of this object.
```

## Fail

public interface **Fail** extends org.omg.CORBA.UserException

This exception interface may be thrown by the `Agent` interface to indicate various errors. The data member `FailReason` is used to indicate the nature of the failure.

## Fail variables

```
com.inprise.vbroker.ObjLocation.FailReason reason

    Indicates the nature of the failure. Must be one of the following values:
```

- NO\_AGENT\_AVAILABLE
- INVALID\_REPOSITORY\_ID
- INVALID\_OBJECT\_NAME
- NO\_SUCH\_TRIGGER
- AGENT\_ERROR

## TriggerDesc

```
public final interface TriggerDesc
```

This interface contains information you use to describe the characteristics of one or more objects for which you wish to register a `TriggerHandler`, described in “TriggerHandler” on page 17-8. These members may be set to `null` to monitor the widest possible set of objects. The more information specified, the smaller the resulting set of objects.

## IDL definition

---

```
struct TriggerDesc {
    string repository_id;
    string instance_name;
    string host_name;
};
```

## TriggerDesc variables

---

public java.lang.String **host\_name**;

Represents the host name where the object or objects to be monitored by the `TriggerHandler` are located. May be set to `null` to include all hosts in the network.

public java.lang.String **instance\_name**;

Represents the instance name of the object to be monitored by the `TriggerHandler`. May be set to `null` to include all possible instance names.

public java.lang.String **repository\_id**;

Represents the repository identifiers of the objects to be monitored by the `TriggerHandler`. May be set to `null` to include all possible repository identifiers.

## TriggerDesc constructor

---

public Desc( java.lang.String **repository\_id**, java.lang.String **instance\_name**,  
java.lang.String **hostname**)

Creates a Desc object, initialized with the specified parameters.

Parameter	Description
<code>repository_id</code>	The repository identifiers of the objects to be monitored by the <code>TriggerHandler</code> . May be set to <code>null</code> to include all possible repository identifiers.
<code>instance_name</code>	The instance name of the object to be monitored by the <code>TriggerHandler</code> . May be set to <code>null</code> to include all possible instance names.
<code>hostname</code>	The host name where the object or objects to be monitored by the <code>TriggerHandler</code> are located. May be set to <code>null</code> to include all hosts in the network.

---

## TriggerDesc methods

---

public java.lang.String **toString()**

Returns a string containing the contents of this object.

# TriggerHandler

---

public interface **TriggerHandler** extends com.inprise.vbroker.CORBA.Object

A **TriggerHandler** is a callback object that is invoked every time an object satisfying the trigger description becomes accessible. You implement a **TriggerHandler** by extending the `_TriggerHandlerImplBase` interface and implementing its `impl_is_ready` and `impl_is_down` methods.

## IDL definition

---

```
interface TriggerHandler {
    void impl_is_ready(in Desc desc);
    void impl_is_down(in Desc desc);
};
```

## TriggerHandler methods

---

public void **impl\_is\_ready**(com.inprise.vbroker.ObjLocation.Desc **desc**)

This method is called by the Location Service when an instance matching the `desc` becomes *accessible*.

Parameter	Description
desc	Description of the instance. The instance description can contain combinations of the following instance information: repository ID, instance name, and host name. The more instance information provided, the more particular your specification of the instance.

public void **impl\_is\_down**(com.inprise.vbroker.ObjLocation.Desc **desc**)

This method is called by the Location Service when an instance matching the `desc` becomes *inaccessible*.

Parameter	Description
desc	Description of the instance. The instance description can contain combinations of the following instance information: repository ID, instance name, and host name. The more instance information provided, the more particular your specification of the instance.



## Using command-line options

This appendix describes the options that may be set for the Basic Object Adaptor, the Object Request Broker, and the Location Service.

**Note** The options in this Appendix can be used with VisiBroker 3.x. However if you are using VisiBroker 4.5, see the properties in Appendix B, “Using VisiBroker properties.”

### How to set ORB and BOA options

---

There are three ways to set the ORB and BOA initialization options,

- Using command-line arguments with the `vbj` command.
- Using command-line arguments and starting your executable with `vbj`.
- Setting properties programmatically using methods.

The ORB initialization options are listed in “ORB options” on page A-5 and the BOA initialization options are listed in “BOA options” on page A-3.

### Using `vbj` with command-line arguments

---

You can use the `vbj` command to define command-line arguments customizing the behavior of the ORB and BOA when invoking your application. When using the `vbj` command with command-line, you must include an equal sign (=) when setting a value. For example,

```
vbj -DOAthreadMax=40 Server
```

**Note** You can also use the `java` command to define command-line properties customizing the behavior of the ORB and BOA when invoking your application; however, when you use the `vbj` command, the ORB checks environment variables.

## Using vbj and starting your executable

---

You can also start the program's executable using the `vbj` command and include command-line arguments. When entering the command-line argument, do not include the `D` after the dash (`-`) or use an equal sign (`=`) when setting a value. For example:

```
vbj <executable> -OathreadMax 40
```

**Note** When you use `vbj` to start the executable, you must call `ORB.init` and pass in the arguments. See “Example using `ORB.init()` with arguments” on page A-4.

## Applets

---

To set options in applets, you must use the parameter name and value. For example,

```
<param name=ORBtcpNoDelay value=60>
```

## Setting properties programmatically using methods

---

You can set properties programmatically using the `ORB` and `BOA` initialization methods: `ORB.init()` and `BOA_init()`. The following sections describe the use of these methods in more detail.

## BOA\_init() method

---

```
public org.omg.CORBA.BOA BOA_init(String boaType, java.util.Properties properties)
```

You use the `BOA_init()` to set the object adapter type and its properties for your application. There are two versions of `BOA_init()`.

- If you use the `BOA_init` method with no arguments, you accept the default thread pooling which is thread pooling.
- If you use `BOA_init` with arguments, you can set the adapter type and its properties. If you don't want to set any properties, you can pass a null instead.

Each time `BOA_init()` is called, it returns an instance of the object adapter type specified. It always returns the same instance of the object adapter. If you call `BOA_init()` with no arguments, it returns an instance of `TPool`. If you call `BOA_init()` with a `TPool` argument and properties, it modifies the existing `TPool`. If you repeatedly call `BOA_init()` with the same type, it updates the properties, if they are different than the previous invocation of `BOA_init()`.

If you run `BOA_init()` *with no arguments* and then invoke your application using a runtime parameter with different settings than `BOA_init()`, `BOA_init()` is overridden by the runtime arguments. However, if you use `BOA_init()` *with*



*arguments* and then use a runtime parameter, the `BOA_init()` takes precedence. The runtime parameter is never even checked.

Code sample A.1 shows the `BOA_init()` method specifying thread per session with no properties.

**Code sample A.1** `BOA_init()` method specifying thread per session, but no properties

```
try {
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
    org.omg.CORBA.BOA boa = orb.BOA_init("TSession", null);
    ...
}
```

The `BOA_init()` method shown in Code sample A.2 is used by your object implementation to set the thread policy as thread pooling with a maximum of 40 threads in the pool. These parameters are passed as arguments to the object implementation's server process when it is started.

**Code sample A.2** `BOA_init()` method specifying thread pool and maximum threads

```
...
java.util.Properties props = new java.util.Properties();
props.put("OathreadMax", "40");
org.omg.CORBA.BOA boa = orb.BOA_init("TPool", props);
...
```

## BOA options

---

The following table summarizes the `BOA_init()` options.

**Table A.1** `BOA_init` options

Options	Description
<code>OAconnectionMax &lt;#&gt;</code>	Specifies the maximum number of allowable incoming connection to the adapter. If you do not specify, the default is unlimited.
<code>OAconnectionMaxIdle &lt;#&gt;</code>	Specifies the number of seconds which a connection can be without any traffic before being shutdown by VisiBroker. The default setting is 0, which means that connections will never time-out. This option should be set for Internet applications.
<code>OAid &lt;TPool   TSession&gt;</code>	Specifies the thread policy to be used by the BOA. The default is TPool unless you are in backward compatibility mode; if you are in backward compatibility, the default is TSession.

**Table A.1** BOA\_init options (continued)

Options	Description
OAipAddr <hostname   ip_address>	Specifies the hostname or IP address to be used for the Object Adaptor. Use this option if your machine has multiple network interfaces and the BOA is associated with just one address. If no option is specified, the host's default address is used.
OAport <port_number>	Specifies the port number to be used by the object adapter when listening for a new connection.
OAthreadMax <#>	Specifies the maximum number of threads allowed when OAid TPool is selected. If you do not specify or you Specify 0, this selects unlimited number of threads or, to be more precise, a number of threads limited only by your system resources.
OAthreadMaxIdle <#>	This specifies number of seconds a thread can exist without servicing any requests before it is returned to the system. By default, this is set to 300 seconds. You can specify this only when OAid TPool is selected
OAthreadMin <#>	Specifies the minimum number of threads available in the thread pool. If you do not specify, the default is zero. You can specify this only when OAid TPool is selected.

## ORB.init() method

```
public static ORB init(String[] args, Properties props)
```

The `ORB.init()` method is used by your application to set such options as the IP address and port number of the Smart Agent to be used. These parameters are passed as arguments to the application process when it is started.

The parameters passed to `ORB.init()` are the same arguments that were passed to your application's main routine. The `ORB.init()` method ignores any arguments it does not recognize.

Code sample A.3 shows the `ORB.init` passing arguments that specify a port for the Smart Agent.

**Code sample A.3** Example using `ORB.init()` with arguments

```
public static void main(String[] args) {
    ...
    java.util.Properties props = new java.util.Properties();
    props.put("ORBagentPort", "9898");
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
    ...
}
```

## ORB options

The following table summarizes the `ORB.init()` options.

**Table A.2** ORB.init options

Options	Description
<code>ORBagentAddr &lt;hostname   ip_address&gt;</code>	Specifies the hostname or IP address of the host running the Smart Agent this client should use. If a Smart Agent is not found at the specified address or if this option is not specified, broadcast messages are used to locate a Smart Agent.
<code>ORBagentAddrFile &lt;file_name&gt;</code>	Specifies a file to be used in place of the default file, <code>agentaddr</code> .
<code>ORBagentNoFailOver &lt;false   true&gt;</code>	Disables fail-over of this Smart Agent to another Smart Agent. If you do not specify, the default value is false.
<code>ORBagentPort &lt;port_number&gt;</code>	Specifies the port number of the Smart Agent. This option can be useful if multiple ORB domains are required. If not specified, a default port number of 14000 is used.
<code>ORBalwaysProxy &lt;false   true&gt;</code>	Specifies whether or not clients must always connect using the Gatekeeper. The default is false. If set to true, <code>ORBgatekeeperIOR</code> must also be set.
<code>ORBalwaysTunnel &lt;false   true&gt;</code>	Specifies whether or not clients must always connect to the Gatekeeper using HTTP. The default is false. If set to true, <code>ORBgatekeeperIOR</code> must also be set.
<code>ORBbackCompat &lt;false   true&gt;</code>	If set to true, this option specifies that backward compatibility with previous versions of VisiBroker should be provided which signals the runtime to be compatible with previous versions of VisiBroker clients and servers. Use <code>ORBbackcompat true</code> when deploying servers and/or clients in an environment based on a previous version of VisiBroker. See the additional information about this option provided after this table.
<code>ORBconnectionCacheMax &lt;#&gt;</code>	Specifies the maximum number of outgoing connections that can be cached. By default, this value is set to 0 which means that there is no limit on the number of cached connections.
<code>ORBconnectionMax &lt;#&gt;</code>	Specifies the maximum number of outgoing connections that are allowed. If you do not specify this option, the default is allow an unlimited number of connections.

**Table A.2** ORB.init options (continued)

Options	Description
ORBconnectionMaxIdle <#>	This specifies the number of seconds that an outgoing connection can idle before it is shutdown by VisiBroker. By default, this is set to 0, which means that connections never time-out. This option should be set for Internet applications.
ORBdebug <false   true>	Turns debugging on.
ORBdebugDir <directory>	Specifies the directory where the thread debugging information is written. By default, the output is written to the current working directory.
ORBdebugThreads <false   true>	Turns thread debugging on.
ORBDefaultInitRef	Specifies the default initial reference.
ORBdisableAgentCache <false   true>	Disables caching of the Smart Agent. If you do not specify, the default value is false.
ORBdisableGatekeeperCallbacks<false   true>	Specifies whether Gatekeeper callbacks are enabled or disabled. If you do not specify, the default value is false and Gatekeeper callbacks are enabled. If set to true, ORBgatekeeperIOR must also be set.
ORBdisableLocator <false   true>	Disables the Smart Agent and Gatekeeper.
ORBgatekeeperIOR <URL>	Specifies a URL that is associated with the IOR.
ORBgcTimeout <#>	Specifies the interval, in seconds, at which ORB garbage collection occurs. By default, the interval is set at 30 seconds.
ORBInitRef	Specifies the initial reference.
ORBmbufSize <buffer_size>	Specifies the size of the intermediate buffer used by VisiBroker for operation request processing. To improve performance, the ORB performs more complex buffer management than in previous versions of VisiBroker. The default size of send and receive buffers is 4K. If data sent or received is larger than the default, new buffers are allocated for each request/reply. If your application frequently sends data larger than 4K and you wish to take advantage of buffer management you may adjust the default using this system property specify the number of bytes you wish to use for a default buffer size.
ORBsecureShutdown <false   true>	If set to false, a user can use the shutdown command to the ORB management interface to shut down the server. By default, this value is set to true, and a CORBA::NO_PERMISSION exception is thrown if a user attempts to use the shutdown command.
ORBservices <service>	Installs one of the ORB's special services. A service can be one that you have created, the ORBManager service supplied with VisiBroker, or one of the VisiBroker services that are sold separately, such as the Naming or Event service.
ORBsyncGC <false   true>	Specifies whether or not to perform synchronous garbage collection. The default value is true.

**Table A.2** ORB.init options (continued)

Options	Description
ORBtcpNoDelay <false   true>	When set to true, it sets all sockets to send requests immediately. The default is false which allows sockets to send requests in batches as the buffer fills.
ORBtcpTimeout <#>	Specifies the number of milliseconds a TCP socket will wait to send data before timing-out. If set to 0, there is no time-out—the application waits forever. If set to a positive value, the application waits for this number of milliseconds before it gives up and assumes the server is down. When a time-out occurs, the connection is broken and a COMM_FAILURE exception is thrown.
ORBwarn <#>	Specifies the level of warning message to be printed 0—Default setting; no warning messages. 1—Prints non-CORBA exceptions thrown by user-written code and the stack trace of the exception. 2—Same as level 1, plus prints CORBA exceptions and the stack trace of the exception.

## ORBbackcompat option

The `ORBbackcompat` option causes the runtime to do the following:

- Double-register with the Smart Agent in the previous version's style of interface name, as well as the 3.0 version's style of repository ID.
- Clients will use the interface name to locate the providers.
- Typecode encoding complies with the CDR encoding from previous versions of VisiBroker.
- Sets the BOA type to TSession (thread-per-session). Otherwise, it is the TPool (thread pool) type.

## Location Service options

These command-line options can be used by your client program to control various Location Service features. When your client application invokes the `ORB.init` method, the Location Services is initialized and receives any command-line arguments you have specified. Command-line options for the Location Service are processed and stripped from the argument list. All unrecognized options are ignored.

As with the command-line options for the BOA and ORB, the Location Service options take the form of type-value pairs.

```
vbj <executable> -LOCdebug 1 -LOctimeout 10 -LOCverify 0
```

The table below summarizes the Location Service command-line options.

**Table A.3**    Location Service options

Type/Value pair	Purpose
LOCdebug <0   1>	If set to 1, enables using the Location Service debugging output, described in Chapter 14, “IOP and IIOP interfaces and classes.” If this option is not specified, debugging output is disabled.
LOTimeout <seconds>	Indicates the number of seconds to wait for a response from a server when verifying the existence of an object. This option is only used when -LOCverify has been set to one. The default value is one second.
LOCverify <0   1>	If set to 1, the Location Service verifies the existence of an object before returning an object reference to the client application. If set to 0, the Location Service offers faster performance, but it may not return the most current information. The default value for this option is 0.

## Using VisiBroker properties

This appendix describes the properties that may be set for VisiBroker for Java.

**Note** The properties in this Appendix can be used with VisiBroker 4.x. However if you want to use VisiBroker 3.x, see the options in Appendix A, “Using command-line options.”

### JAVA RMI over IIOP properties

This table lists the Java RMI over IIOP properties.

Property	Default	Description
<code>javax.rmi.CORBA.StubClass</code>	<code>com.inprise.vbroker.rmi.CORBA.StubImpl</code>	Specifies the name of the implementation of the Stub base class from which all RMI-IIOP stubs must inherit.
<code>javax.rmi.CORBA.UtilClass</code>	<code>com.inprise.vbroker.rmi.CORBA.UtilImpl</code>	Specifies the name of the implementation of the Utility class that provides methods that can be used by stubs and ties to perform common operations.
<code>javax.rmi.CORBA.PortableRemoteObjectClass</code>	<code>com.inprise.vbroker.rmi.CORBA.PortableRemoteObjectImpl</code>	Specifies that the RMI-IIOP server implementation objects may inherit from <code>javax.rmi.PortableRemoteObject</code> or simply implement an RMI-IIOP remote interface and then use the <code>exportObject</code> method to register themselves as a server object.

OSAgent properties

Property	Default	Description
java.rmi.server.codebase	<not set>	Specifies where a server can locate unknown classes.
java.rmi.server.useCodebaseOnly	false	Specifies if a server is allowed to locate unknown classes, If set to true, does not allow the server to locate remote classes even if the client sends the location of the remote classes to the server.

OSAgent properties

This table lists the OSAgent (Smart Agent) properties.

Property	Default	Old property	Description
vbroker.agent.addr	null	ORBagentAddr	Specifies the IP address or host name of the host running the OSAgent. The default value, <i>null</i> , instructs VisiBroker applications to use the value from the OSAGENT_ADDR environment variable. If this OSAGENT_ADDR variable is not set, then it is assumed that the OSAgent is running on a local host.
vbroker.agent.addrFile	null	ORBagentAddrFile	Specifies a file that stores information on where the IP address(es) or host name(s) OSAgent maybe found.
vbroker.agent.debug	false	ORBdebug	When set to true, specifies that the system will display debugging information about communication of VisiBroker applications with the OSAgent.
vbroker.agent.enableCache	true	ORBagentCache	When set to true, allows VisiBroker applications to cache IOR.
vbroker.agent.enableLocator	true	ORBdisableLocator	When set to false, does not allow VisiBroker applications to communicate with the OSAgent.
vbroker.agent.failOver	true	ORBagentNoFailOver	When set to true, allows a VisiBroker application to fail over to another OSAgent.
vbroker.agent.port	14000	ORBagentPort	Specifies the port number that defines a domain within your network. VisiBroker applications and the OSAgent work together when they have the same port number. This is the same property as the OSAGENT_PORT environment variable.



# ORB properties

This table lists the ORB properties.

Property	Default	Old property	Description
<code>vbroker.orb.alwaysProxy</code>	false	<code>ORBalwaysProxy</code>	When set to true, specifies that clients must always connect to the server using the Gatekeeper.
<code>vbroker.orb.alwaysSecure</code>	false		When set to true, specifies that clients must always make secure connections to the server.
<code>vbroker.orb.alwaysTunnel</code>	false	<code>ORBalwaysTunnel</code>	When set to true, specifies that clients always make http tunnel (IIOP wrapper) connections to the server.
<code>vbroker.orb.autoLocateStubs</code>	false		Turns on the ability to locate stubs when reading object references. This is done using <code>read_Object</code> , based on the object's repository id instead of either the generic object or the stubs for passed formal class argument.
<code>vbroker.orb.bidOrder</code>	<code>inprocess:liop:ssl:iiop:proxy:hiop:locator</code>		<p>You can specify the relative order of importance for the various transports. Transports are given precedence as follows:</p> <ol style="list-style-type: none"> <li>1 inprocess</li> <li>2 liop</li> <li>3 ssl</li> <li>4 iiop</li> <li>5 proxy</li> <li>6 hiop</li> <li>7 locator</li> </ol> <p>The transports that appear first have higher precedence. For example: If an IOR contains both LIOP and IIOP profiles, the first chance goes to LIOP. Only if the LIOP fails is IIOP used. (The critical bid, specified by the <code>vbroker.orb.bids.critical</code> property, has highest precedence no matter where it is specified in the bid order. )</p>

## ORB properties

Property	Default	Old property	Description
vbroker.orb.bids.critical	inprocess		The critical bid has highest precedence no matter where it is specified in the bid order. If there are multiple values for critical bids, then their relative importance is decided by the <code>bidOrder</code> property.
vbroker.orb.defAddrMode	0 ( <i>Key</i> )		The default addressing mode that client ORB uses. If it is set to 0, the addressing mode is <i>Key</i> , if set to 1, the addressing mode is <i>Profile</i> , if set to 2, the addressing mode is <i>IOR</i> .
vbroker.orb. bufferCacheTimeout	6000		Specifies the time in which a message chunk has been cached before it is discarded.
vbroker.orb.bufferDebug	false		When set to true, this property allows the internal buffer manager to display debugging information.
vbroker.orb.debug	false		When set to true, allows the ORB to display debugging information.
vbroker.orb.DefaultInitRef	null	ORBDefaultInitRef	Specifies the default initial reference.
vbroker.orb.dynamicLibs	null		Specifies a list of available services used by the ORB.
vbroker.orb.embedCodeset	true		When an IOR is created, the ORB embeds the codeset components into the IOR. This may produce problems with some non-compliant ORBs. By turning off the <code>embedCodeset</code> property, you instruct the Visibroker ORB not to embed codesets in IORs. When set to false, specifies that character and wide character conversions between the client and the server are not to be negotiated.
vbroker.orb.enableVB4backcompat	true		This property enables workarounds to deal with behavior that is not GIOP 1.2-compliant in VisiBroker 4.0 and 4.1. In a multi-vendor environment with GIOP 1.2-compliant ORBs when <code>wchar/wstring</code> data types are transmitted, this flag needs to be set to false.

Property	Default	Old property	Description
<code>vbroker.orb.enableBiDir</code>			You can selectively make bidirectional connections. If the client defines <code>vbroker.orb.enableBiDir=client</code> and the server defines <code>vbroker.orb.enableBiDir=server</code> the value of <code>vbroker.orb.enableBiDir</code> at the gatekeeper determines the state of the connection. Values of this property are: <code>server</code> , <code>client</code> , <code>both</code> or <code>none</code> . See the Gatekeeper Guide for more information.
<code>vbroker.orb.enableKeyId</code>	<code>true</code>		When set to <code>false</code> , disables the use of key IDs in client requests.
<code>vbroker.orb.enableNullString</code>	<code>true</code>	<code>ORBNullString</code>	If set to <code>true</code> , enables marshalling of null strings.
<code>vbroker.orb.enableServerManager</code>	<code>false</code>		When set to <code>true</code> , enables Server Manager within a server so that clients can access it.
<code>vbroker.orb.fragmentSize</code>	<code>0</code>		Specifies the GIOP message fragment size. It must be a multiple of GIOP message chunk size. Assigning a zero to this property will eventually turn off fragmentation.
<code>vbroker.orb.InitRef</code>	<code>null</code>	<code>ORBInitRef</code>	Specifies the initial reference.
<code>vbroker.orb.streamChunkSize</code>	<code>4096</code>		Specifies the GIOP message chunk size. It must be a power of 2.
<code>vbroker.orb.gcTimeout</code>	<code>30</code>	<code>ORBgcTimeout</code>	Specifies the time in seconds that must pass before important resources that are not used are cleared.
<code>vbroker.orb.logger.appName</code>	<code>VBJ-Application</code>		Specifies the application name that appears in the log.
<code>vbroker.orb.logger.catalog</code>	<code>com.inprise.vbroker.Logging.ORBMsgs</code>		Specifies the message catalog of messages used by the ORB when logging is enabled.
<code>vbroker.orb.logger.output</code>	<code>stdout</code>		Specifies the output of the logger. It can be the standard output or a file name.
<code>vbroker.orb.logLevel</code>	<code>emerg</code>		Specifies the logging level of message that will be logged. The default value, <i>emerg</i> , means that the system logs messages when the system is unusable, or in a panic condition.
<code>vbroker.orb.procId</code>	<code>0</code>		Specifies the process ID of the server.

## ORB properties

Property	Default	Old property	Description
<code>vbroker.orb.sendLocate</code>	<code>false</code>		When set to true, forces the system to send a locate request before making invocations on a IIOP 1.2 target.
<code>vbroker.orb.systemLibs.applet</code>	<code>com.inprise.vbroker.IIOP.Init,</code> <code>com.inprise.vbroker.LIOP.Init,</code> <code>com.inprise.vbroker.qos.Init,</code> <code>com.inprise.vbroker.URLNaming.Init,</code> <code>com.inprise.vbroker.HIOP.Init,</code> <code>com.inprise.vbroker.firewall.Init,</code> <code>com.inprise.vbroker.dynamic.Init,</code> <code>com.inprise.vbroker.naming.Init</code>		Provides a list of system libraries loaded in applet.
<code>vbroker.orb.systemLibs.application</code>	<code>com.inprise.vbroker.IIOP.Init,</code> <code>com.inprise.vbroker.LIOP.Init,</code> <code>com.inprise.vbroker.qos.Init,</code> <code>com.inprise.vbroker.ds.Init,</code> <code>com.inprise.vbroker.URLNaming.Init,</code> <code>com.inprise.vbroker.dynamic.Init,</code> <code>com.inprise.vbroker.ir.Init,</code> <code>com.inprise.vbroker.naming.Init</code>		Provides a list of system libraries loaded in application.
<code>vbroker.orb.tcIndirection</code>	<code>true</code>		Specifies that indirection be turned off when writing the typecodes. May be necessary when inter operating with ORBs from other vendors. When set to the value, false, it is not possible to marshall recursive typecodes.
<code>vbroker.orb.warn</code>	<code>0</code>	<code>ORBwarn</code>	Specifies a value of 0, 1, or 2 which indicates the level of warning messages to be printed.

## POA properties

This table lists the POA properties.

Property	Default	Description
<code>vbroker.poa.logLevel</code>	<code>emerg</code>	Specifies the logging level of messages to be logged. The default value, <code>emerg</code> , means that messages are logged when the system is unusable or during a panic condition.

## Server Manager properties

This table lists the Server Manager properties.

Property	Default	Description
<code>vbroker.serverManager.name</code>	<code>null</code>	Specifies the name of the Server Manager.
<code>vbroker.serverManager.enableOperations</code>	<code>true</code>	When set to true, enables operations exposed, by the Server Manager, to be invoked.
<code>vbroker.serverManager.enableSetProperty</code>	<code>true</code>	When set to true, enables properties, exposed by the Server Manager, to be changed.

## Location Service properties

This table lists the Location Service properties.

Property	Default	Description
<code>vbroker.locationservice.debug</code>	<code>false</code>	When set to true, allows the Location Service to display debugging information.
<code>vbroker.locationservice.verify</code>	<code>false</code>	When set to true, allows the Location Service to check for the existence of an object referred by an object reference sent from the OSAgent.
<code>vbroker.locationservice.timeout</code>	<code>1</code>	Specifies the connect/receive/send timeout when trying to interact with the Location Service.

# Event Service properties

This table lists the Event Service properties.

Property	Default	Description
<code>vbroker.events.maxQueueLength</code>	100	Specifies the number of messages to be queued for slow consumers.
<code>vbroker.events.factory</code>	false	When set to true, allows the event channel factory to be instantiated, instead of an event channel.
<code>vbroker.events.debug</code>	false	When set to true, allows output of debugging information.
<code>vbroker.events.interactive</code>	false	When set to true, allows the event channel to be executed in a console-driven, interactive mode.

# Naming Service properties

Naming Service properties are listed in the VisiBroker for Java *Programmer's Guide*, Chapter 18, "Using the Naming Service."

# OAD properties

This table lists the OAD properties that can be set.

Property	Default	Description
<code>vbroker.oad.spawnTimeOut</code>	20	After the OAD spawns an executable, specifies how long, in seconds, the system will wait to receive a callback from the desired object before throwing a NO_RESPONSE exception.
<code>vbroker.oad.verbose</code>	false	Allows the OAD to print detailed information about its operations.
<code>vbroker.oad.readOnly</code>	false	When set to true, does not allow you to register, unregister, or change the OAD implementation.
<code>vbroker.oad.iorFile</code>	<code>Oadj.ior</code>	Specifies the filename for the OAD's stringified IOR.
<code>vbroker.oad.quoteSpaces</code>	false	Specifies whether to quote a command.
<code>vbroker.oad.killOnUnregister</code>	false	Specifies whether to kill spawned servers once they are unregistered.
<code>vbroker.oad.verifyRegistration</code>	false	Specifies whether to verify the object registration.

This table lists the OAD properties that cannot be overridden in a property file. They can however be overridden with environment variables or from the command line.

Property	Default	Description
<code>vbroker.oad.implName</code>	<code>impl_rep</code>	Specifies the filename for the implementation repository.
<code>vbroker.oad.implPath</code>	<code>null</code>	Specifies the directory where the implementation repository is stored.
<code>vbroker.oad.path</code>	<code>null</code>	Specifies the directory for the OAD.
<code>vbroker.oad.systemRoot</code>	<code>null</code>	Specifies the root directory.
<code>vbroker.oad.windir</code>	<code>null</code>	Specifies the Windows directory.
<code>vbroker.oad.vbj</code>	<code>vbj</code>	Specifies the VBJ directory.

## Interface Repository properties

This table lists the Interface Repository (IR) properties.

Property	Default	Description
<code>vbroker.ir.debug</code>	<code>false</code>	When set to true, allows the IR resolver to display debugging information.
<code>vbroker.ir.ior</code>	<code>null</code>	When the <code>vbroker.ir.name</code> property is set to the default value, <i>null</i> , the ORB will try to use this property to locate the IR.
<code>vbroker.ir.name</code>	<code>null</code>	Specifies the name that is used by the ORB to locate the IR.

## URL Naming properties

This table lists the URL Naming properties.

Property	Default	Description
<code>vbroker.URLNaming.allowUserInteraction</code>	<code>true</code>	When set to true, allows the URL Naming Service to initiate the graphical user interface (GUI) for user interaction.
<code>vbroker.URLNaming.debug</code>	<code>false</code>	When set to true, specifies that the URLNaming Service display debugging information.

# Client-Side Connection properties

This table lists the Client-Side Connection properties.

Property	Default	Description
vbroker.ce.iiop.ccm.connectionCacheMax	5	Specifies the maximum number of cache connections on a client. The connection is cached when the client releases it. Consequently, the next time the client needs a new connection, it first tries to get an available one from the cache, instead of creating a new one.
vbroker.ce.iiop.ccm.connectionMax	0	Specifies the maximum number of total connections within a client. This value is equal to active connections plus those that are cached. The default value, zero, means that the client will not try to close any of the old active or cached connections.
vbroker.ce.iiop.ccm.connectionMaxIdle	360	Specifies the time in msec that the client uses to determine if a cached connection should be closed or not. In other words, if a cached connection has been idle longer than this time, then the client will close it.
vbroker.ce.iiop.connection.tcpNoDelay	false	When set to <i>true</i> , turns off buffering for the socket so that all packets are sent as soon as they are ready. The default value, <i>false</i> , turns on buffering.
vbroker.ce.iiop.ccm.type	Pool	Specifies what type of client connection management is used by a client. The default value, <i>Pool</i> , means connection pool.

# Client-Side In Process Connection properties

This table lists the Client-Side In-Process Connection properties.

Property	Default	Description
vbroker.ce.inprocess.ccm.bid	9488	Specifies the bid value for the POA bidder. It affects an automatic process in the ORB used in picking a protocol to handle client connections.



Property	Default	Description
<code>vbroker.ce.iiop.ccm.bid</code>	10000	Specifies the bid value for the iiop bidder. It affects an automatic process in the ORB used in picking a protocol to handle client connections.
<code>vbroker.ce.iiop.ccm.bid</code>	9744	Specifies the bid value for the liop bidder. It affects an automatic liop in the ORB used in picking a protocol to handle client connections.

## Server-Side Server Engine properties

This table lists the Server-Side Server Engine properties.

Property	Default	Description
<code>vbroker.se.default</code>	<code>iiop_tp</code>	Specifies the default server engine.
<code>vbroker.se.&lt;se&gt;.scm.&lt;scm&gt;.listener.giopVersion</code>	1.2	This property can be used to resolve interoperability problems with older ORBs that cannot handle unknown minor GIOP versions correctly. Legal values for this property are 1.0, 1.1 and 1.2. For example, to make the nameservice produce a GIOP 1.1 ior, start it like this:  nameserv -VBJprop <code>vbroker.se.iiop_tp.scm.iiop_tp.listener.giopVersion=1.1</code>

## Server-Side Thread Session IIOP\_TS/IIOP\_TS Connection properties

This table lists the Server-Side Thread Session IIOP\_TS/IIOP\_TS Connection properties.

Property	Default	Description
<code>vbroker.se.iiop_ts.host</code>	<code>null</code>	Specifies the host name used by this server engine. The default value, <i>null</i> , means use the host name from the system.
<code>vbroker.se.iiop_ts.proxyHost</code>	<code>null</code>	Specifies the proxy host name used by this server engine. The default value, <i>null</i> , means use the host name from the system.
<code>vbroker.se.iiop_ts.scms</code>	<code>iiop_ts</code>	Specifies the list of Server Connection Manager name(s).
<code>vbroker.se.iiop_ts.scm.iiop_ts.manager.type</code>	<code>Socket</code>	Specifies the type of Server Connection Manager.

Property	Default	Description
vbroker.se.iiop_ts.scm.iiop_ts.manager.connectionMax	0	Specifies the maximum number of connections the server will accept. The default value, 0 (zero), implies no restriction.
vbroker.se.iiop_ts.scm.iiop_ts.manager.connectionMaxIdle	0	Specifies the time in seconds the server uses to determine if an inactive connection should be closed or not.
vbroker.se.iiop_ts.scm.iiop_ts.listener.type	IIOP	Specifies the type of protocol the listener is using.
vbroker.se.iiop_ts.scm.iiop_ts.listener.port	0	Specifies the port number that is used with the host name property. The default value, 0 (zero), means the system will pick a random port number.
vbroker.se.iiop_ts.scm.iiop_ts.listener.proxyPort	0	Specifies the proxy port number used with the proxy host name property. The default value, 0 (zero), means the system will pick a random port number.
vbroker.se.iiop_ts.scm.iiop_ts.dispatcher.type	"ThreadSession"	Specifies the type of thread dispatcher used in the Server Connection Manager.

## Server-Side Thread Session BOA\_TS/BOA\_TS Connection properties

This protocol has the same set of properties as the thread session `iiop_ts/iiop_ts` connection properties (see above table), by replacing all `iiop_ts` with `boa_ts` in all the properties. For example, the `vbroker.se.iiop_ts.scm.iiop_ts.manager.connectionMax` will become `vbroker.se.boa_ts.scm.boa_ts.manager.connectionMax`. Also, the default value for `vbroker.se.boa_ts.scm` is `boa_ts`.

# Server-Side Thread Pool IIOP\_TP/IIOP\_TP Connection properties

This table lists the Server-Side Thread Pool IIOP\_TP/IIOP\_TP Connection properties.

Property	Default	Description
vbroker.se.iiop_tp.host	null	Specifies the host name that can be used by this server engine. The default value, <i>null</i> , means use the host name from the system.
vbroker.se.iiop_tp.ProxyHost	null	Specifies the proxy host name that can be used by this server engine. The default value, <i>null</i> , means use the host name from the system.
vbroker.se.iiop_tp.scms	iiop_tp	Specifies the list of Server Connection Manager name(s).
vbroker.se.iiop_tp.scm.iiop_tp.connection.tcpNoDelay	false	When set to <i>true</i> , turns off buffering for the socket so that all packets are sent as soon as they are ready. The default value, <i>false</i> , turns on buffering.
vbroker.se.iiop_tp.scm.iiop_tp.manager.type	Socket	Specifies the type of Server Connection Manager.
vbroker.se.iiop_tp.scm.iiop_tp.manager.connectionMax	0	Specifies the maximum number of cache connection on the server. The default value, 0 (zero), implies no restriction.
vbroker.se.iiop_tp.scm.iiop_tp.manager.connectionMaxIdle	0	Specifies the time in seconds that the server uses to determine if an inactive connection should be closed or not.
vbroker.se.iiop_tp.scm.iiop_tp.listener.type	IIOP	Specifies the type of protocol the listener is using.
vbroker.se.iiop_tp.scm.iiop_tp.listener.port	0	Specifies the port number used with the host name property. The default value, 0 (zero), means that the system will pick a random port number.
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort	0	Specifies the proxy port number used with the proxy host name property. The default value, 0 (zero), means that the system will pick a random port number.
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.type	ThreadPool	Specifies the type of thread dispatcher used in the Server Connection Manager.
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin	0	Specifies the minimum number of threads that the Server Connection Manager can create.

Property	Default	Description
<code>vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax</code>	0	Specifies the maximum number of threads that the Server Connection Manager can create. The default value, 0 implies no restriction.
<code>vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle</code>	300	Specifies the time in seconds before a idle thread will be destroyed.

## Server-Side Thread Pool BOA\_TP/BOA\_TP Connection properties

This protocol has the same set of properties as the thread pool `iiop_tp/iiop_tp` connection properties (see above table), by replacing all `iiop_tp` with `boa_tp` in all the properties. For example, the `vbroker.se.iiop_tp.scm.iiop_tp.manager.connectionMax` will become `vbroker.se.boa_tp.scm.boa_tp.manager.connectionMax`. Also, the default value for `vbroker.se.boa_tp.scm` is `boa_tp`.

## Properties that support bidirectional communication

This table lists the properties that support bidirectional communication.

**NOTE:** These properties are evaluated only once -- when the SCMs are created. In all cases, the `exportBiDir` and `importBiDir` properties on the SCMs are given priority over the `enableBiDir` property. In other words, if both properties are set to conflicting values, the SCM-specific properties will take effect. This allows you to set the `enableBiDir` property globally and specifically turn off bidirectionality in individual SCMs.

Property	Default	Description
<code>vbroker.orb.enableBiDir</code>		You can selectively make bidirectional connections. If the client defines <code>vbroker.orb.enableBiDir=client</code> and the server defines <code>vbroker.orb.enableBiDir=server</code> the value of <code>vbroker.orb.enableBiDir</code> at the gatekeeper determines the state of the connection. Values of this property are: <code>server</code> , <code>client</code> , <code>both</code> or <code>none</code> .
<code>vbroker.se.&lt;se&gt;.scm.&lt;scm&gt;.manager.exportBiDir</code>	By default, not set by the ORB.	A client-side property. Setting it to <code>true</code> enables creation of a bidirectional callback POA on the specified server engine. Setting it to <code>false</code> disables creation of a bidirectional POA on the specified server engine.
<code>vbroker.se.&lt;se&gt;.scm.&lt;scm&gt;.manager.importBiDir</code>	By default, not set by the ORB.	A server-side property. Setting it to <code>true</code> allows the server-side to reuse the connection already established by the client for sending requests to the client. Setting it to <code>false</code> prevents reuse of connections in this fashion.

# Index

## Symbols

---

... ellipsis 1-4  
[] brackets 1-4  
| vertical bar 1-4

## A

---

AbstractInterfaceDef interface 7-1  
activation  
    interfaces 8-1  
    policies 8-3  
ActivationImplDef 8-1  
ActivationImplDef interface 8-1  
Activator  
    impl\_is\_ready 5-4  
Activator interface 8-2  
additional information 1-5  
Agent interface 17-1  
AliasDef interface 7-3  
Any  
    Any type mapping 3-25  
applications  
    locating osagent via vbj command 2-9  
ARG\_IN class 17-4  
arguments  
    -ORBshmsize 2-9  
ArrayDef interface 7-4  
arrays  
    mapping 3-11  
AttributeDef interface 7-5  
AttributeDescription class 7-6  
AttributeMode class 7-7  
attributes  
    list of exception classes 11-2

## B

---

backward compatibility A-5, A-6  
basic types  
    IDL types 3-3  
Binding  
    interface 9-8  
binding  
    BindOptions class 5-1  
Binding structure 9-8  
BindingIterator 9-10  
    class 9-9  
    interface 9-9  
BindingList  
    interface 9-8  
BindingList sequence 9-8

BindInterceptor 12-3  
BindOptions class 5-1  
BOA  
    initialization options A-1, A-2  
        OAConnectionMax A-3  
        OAipaddr A-4  
        OAport A-4  
        OAThreadMax A-4  
    options A-3  
BOA interface 5-2  
BOA\_init  
    method A-2  
BOA\_init() 5-23  
BOA\_init(String boaType, java.util.Properties  
    properties) 5-24, A-2  
boolean type  
    mapping 3-8  
buffer size  
    setting A-6

## C

---

Caffeine compiler  
    description 2-6  
char type  
    mapping 3-8  
class  
    BindInterceptor 12-3  
    ClientInterceptor 12-7, 12-8  
classes  
    Any 6-1  
    ARG\_IN 6-4, 17-4  
    ARG\_INOUT 6-4  
    ARG\_OUT 6-4  
    AttributeDescription 7-6  
    AttributeMode 7-7  
    BindOptions 5-1  
    BOA 5-2  
    Closure 12-14  
    CompletionStatus 5-5  
    ConstantDescription 7-8  
    ContainedPackage.Description 7-12  
    ContainerPackage.Description 7-19  
    ContextList 6-5  
    DefinitionKind 7-20  
    dynamic 6-1  
    DynamicImplementation 6-17  
    DynAny 6-6  
    DynAnyFactory 6-10  
    DynArray 6-9  
    DynEnum 6-11

- DynFixed 6-12
- DynSequence 6-12
- DynStruct 6-13
- DynUnion 6-14
- DynValue 6-16
- Environment 6-18
- ExceptionDescription 7-22
- ExceptionList 6-18
- Fail 17-6
- FullInterfaceDescription 7-28
  - generated 4-1
- Helper 4-1, 4-3
- Holder 4-1, 4-6
- ImplementationDef 8-7
- InterfaceDescription 7-29
- Invalid 6-20
- InvalidName 5-8
- InvalidSeq 6-21
- IORValue 14-2
- java.lang.Runtime 11-1
- ModuleDescription 7-31
- NamedValue 6-21
- NameValuePair 6-22
- NVList 6-22
- OperationDescription 7-34
- OperationMode 7-35
- Operations 4-3
- operations 4-2
- ORB 5-13
- ParameterDescription 7-36
- ParameterMode 7-37
- POA 4-2
- POATie 4-2
- PortableServer.AdaptorActivator 5-25
- PortableServer.POAManager 5-36
- PortableServer.ServantActivator 5-37
- PortableServer.ServantLocator 5-38
- PortableServer.ServantManager 5-40
- PrimitiveKind 7-37
- Principal 5-40
- ProfileBody 14-1
- ServiceContext 14-3
- skeleton 4-7
- stub 4-2, 4-7
- SystemException 11-1
- TaggedProfile 14-4
- TCKind 6-30
- tie 4-8
- TypeDescription 7-43
- UnionMember 7-45
- UnknownUserException 6-34
- UserException 11-3
- CLASSPATH 2-9
- classPath 2-10
- classpath 2-10
- client
  - locating osagent via vbj command 2-9
  - property list 5-6
- ClientInterceptor 12-7, 12-8
- Client-Side Connection
  - properties B-10
- Client-Side In-Process Connection
  - properties B-10
- Closure 12-14
- commands
  - idl2ir 2-1, 2-2
  - idl2java 2-3
  - java2idl 2-5
  - java2iiop 2-6
  - vbj 2-9
- CompletionStatus class 5-5
- connect\_push\_consumer
  - method 10-7
- connection timeout 13-8
- connections
  - idle time A-3, A-6
  - maximum number A-3, A-5
- ConstantDef interface 7-8
- ConstantDescription class 7-8
- constants
  - mapping 3-10
- constructed types
  - mapping 3-11
- constructors
  - DynamicImplementation 6-17
  - NameValuePair in NameValuePair 6-22
- ConsumerAdmin
  - interface 10-1
  - method 10-2
- contacting Borland 1-5
- contacting Inprise 1-5
- Contained interface 7-10
- ContainedPackage.Description class 7-12
- Container interface 7-12
- ContainerPackage.Description class 7-19
- Context interface 5-6
- ContextList class 6-5
- conventions 1-4
  - platform 1-4
  - platform icons 1-4
  - typographic 1-4
- CosEventChannelAdmin 10-11
- CosEventComm 10-10
- CreationImplDef interface 8-3

## D

---

- DeferBindPolicy
  - interface 13-9
- DefinitionKind class 7-20
- delegation

- with server implementations 3-22

- Desc
  - interface 17-4
- developer support 1-5
- DII
  - generating portable stubs 2-7
- disconnect\_push\_consumer()
  - method 10-8
- DynamicImplementation
  - classes 6-17
- DynAny class 6-6
- DynAnyFactory
  - classes 6-10
- DynArray
  - classes 6-9
- DynEnum
  - classes 6-11
- DynFixed
  - classes 6-12
- DynSequence
  - classes 6-12
- DynStruct
  - classes 6-13
- DynUnion
  - classes 6-14
- DynValue
  - classes 6-16

## E

---

- EnumDef interface 7-21
- enums
  - Location 12-14
  - mapping 3-11
- Environment
  - classes 6-18
- Environment class 6-18
- environment variables 8-5
- environment variables in CreationImplDef 8-4
- event handlers
  - interfaces 9-1, 10-1
- Event Service
  - properties B-8
- EventChannel
  - class 10-2
  - interface 10-2
  - methods 10-2
- EventLibrary
  - class 10-3
  - interface 10-3
  - methods 10-3, 10-4
  - create 10-4
- ExceptionDef interface 7-21
- ExceptionDescription class 7-22
- ExceptionList class 6-18
- exceptions

- class hierarchy 11-1
- ForwardRequestException 12-10
- list of attributes 11-2
- mapping 3-23
- Quality of Service 13-11
- system 3-24
- user-defined 3-23
- executable
  - starting A-2
- ExtendedNamingContextFactory
  - class 9-11
  - interface 9-11
  - method 9-12
- extensions
  - VisiBroker ORB 5-22
- extract method
  - in Helper classes 4-3

## F

---

- Fail
  - class 17-6
- FixedDef
  - interface 7-23
- floating point
  - mapping 3-9
- FullInterfaceDescription class 7-28
- FullValueDescription
  - interface 7-24

## G

---

- generated classes 4-1

## H

---

- Helper classes 4-3
  - mapping 3-9
  - overview 4-1
- Holder classes 4-6
  - mapping 3-5
  - overview 4-1
- hostname A-4

## I

---

- IDL
  - creating from Java 2-5
  - generating Java code 2-3
  - mapping constants 3-10
  - mapping constructed types 3-11
  - mapping interfaces 3-18, 3-21
  - mapping modules 3-3
  - mapping names to Java 3-1
  - mapping nested types 3-25
  - mapping parameters 3-20

- mapping to Java 3-1
- mapping typedefs 3-25
- mapping types 3-3
- reserved names 3-2
- reserved words 3-2
- type extensions 3-4
- using Java to define IDL 2-6
- IDL type
  - basic types 3-3
  - boolean 3-8
  - char 3-8
  - complex 3-26
  - floating point type 3-9
  - Holder classes 3-5
  - integer type 3-9
  - octet 3-8
  - simple 3-26
  - string 3-8
  - wstring 3-9
- idl2ir 2-1
  - command info 2-1, 2-2
  - description 2-1, 2-2
  - options 2-2
- idl2java
  - command info 2-3
  - generating portable stubs for DII 2-3
  - options 2-3
- IDLType interface 7-25
- ImplementationDef 8-7
- ImplementationDef class 8-7
- implementations
  - using delegation 3-22
- ImplementationStatus
  - struct 8-8
- ImplementationStatus struct 8-8
- information, where to find 1-5
- inheritance
  - implementing servers 3-21
- InputStream interface 6-19
- installation support 1-5
- integer
  - mapping 3-9
- interceptor
  - managers 12-1
- InterceptorManager
  - interface 12-2
- interceptors
  - Closure 12-14
  - Closure objects 12-14
- Interface Repository
  - populating with idl2ir 2-1, 2-2
  - properties B-9
- interface scope
  - mapping 3-23
- interface\_name
  - operations 4-3
  - POA 4-7
  - POATie 4-8
  - stub 4-7
- InterfaceDef interface 7-26
- InterfaceDescription class 7-29
- interfaces
  - AbstractInterfaceDef 7-1
  - ActivationImplDef 8-1
  - Activator 8-2
  - ActiveObjectLifeCycleInterceptor 12-9
  - ActiveObjectLifeCycleInterceptorManager 12-9
  - Agent 17-1
  - AliasDef 7-3
  - Any 6-1
  - ARG\_IN 6-4
  - ARG\_INOUT 6-4
  - ARG\_OUT 6-4
  - ArrayDef 7-4
  - AttributeDef 7-5
  - Binding 9-8
  - BindingIterator 9-9
  - BindingList 9-8
  - BindInterceptor 12-3
  - BindInterceptorManager 12-4
  - BOA 5-2
  - ChainUntypedObjectWrapperFactory 12-15
  - ClientRequestInterceptor 12-5
  - ClientRequestInterceptorManager 12-7
  - ConstantDef 7-8
  - ConsumerAdmin 10-1
  - Contained 7-10
  - Container 7-12
  - Context 5-6
  - CORBA Object 13-3
  - CreationImplDef 8-3
  - DeferBindPolicy 13-9
  - Desc 17-4
  - dynamic 6-1
  - EnumDef 7-21
  - EventChannel 10-2
  - EventLibrary 10-3
  - ExceptionDef 7-21
  - ExtendedNamingContextFactory 9-11
  - FullValueDescriptionf 7-24
  - IDLType 7-25
  - InputStream 6-19
  - InterceptorManager 12-2
  - InterceptorManagerControl 12-2
  - InterfaceDef 7-26
  - IORCreationInterceptor 12-13
  - IORInterceptorManager 12-13
  - IRObject 7-30
  - mapping 3-18, 3-21
  - ModuleDef 7-31



- NamingContext
  - NamingContext
    - interface 9-1
- NamingContextExt 9-6
- NamingContextFactory 9-10
- NativeDef 7-32
- NVList 6-22
- OAD 8-7
- Object 5-8, 13-4
- OperationDef 7-32
- OutputStream 6-24
- POA 5-26
- POALifeCycleInterceptor 12-7
- POALifeCycleInterceptorManager 12-8
- PolicyCurrent 13-3
- PolicyManager 13-1
- PortableRemoteObject 15-1
- PortableServer.POAManager 5-36
- PortableServer.ServantActivator 5-37
- PortableServer.ServantLocator 5-38
- PortableServer.ServantManager 5-40
- PrimitiveDef 7-37
- ProxyPullConsumer 10-4
- ProxyPullSupplier 10-6
- ProxyPushConsumer 10-5
- ProxyPushSupplier 10-7
- PullConsumer 10-8, 10-9
- PullSupplier 10-9
- PushConsumer 10-8
- PushSupplier 10-10
- RebindPolicy 13-5
- Repository 7-38
- Request 6-25
- Resolver 16-1
- SequenceDef 7-40
- ServerRequest 6-29
- ServerRequestInterceptor 12-10
- ServerRequestInterceptorManager 12-12
- StringDef 7-41
- StructDef 7-42
- StructMember 7-42
- SupplierAdmin 10-11
- TriggerHandler 17-8
- TypeCode 6-31
- TypedefDef 7-43
- UnionDef 7-44
- UntypedObjectWrapper 12-16
- UntypedObjectWrapperFactory 12-18
- ValueBoxDefValueBoxDef
  - interface 7-46
- ValueDef 7-46
- ValueDescription
  - interface 7-49
  - ValueDescription
    - interface 8-7

- ValueMemberDef
  - interface 7-50
- WstringDef 7-51
- interfaces FixedDef 7-23
- interfacesPrincipal 5-40
- Inter-operable Object Reference (see IOR) 14-2
- Invalid
  - classes 6-20
- InvalidName class 5-8
- InvalidSeq
  - classes 6-21
- IOR
  - associating with a URL 16-2
  - templates 12-2
- IORValue
  - class 14-2
- IP address A-4, A-5
- ir2idl 2-2
  - options 2-3
- IRObjct 7-30
- IRObjct interface 7-30

## J

---

- Java
  - creating an IDL file from Java 2-5
  - defining IDL interfaces 2-6
  - generating code from IDL file 2-3
  - mapping from IDL 3-1
  - null 3-8
  - RMI over IIOP properties B-1
  - starting the interpreter via vbj 2-9
- java2idl
  - command info 2-5
  - description 2-5
  - options 2-5
- java2iiop
  - command info 2-6
  - generating portable stubs for DII 2-7
  - options 2-7
- JVM 2-11

## L

---

- List 9-6
- Location
  - enum 12-14
- Location Service
  - options A-7
  - properties B-7
  - query a Smart Agent 17-1
  - registering triggers 17-1
  - TriggerDesc 17-6

# M

## manual

- conventions 1-4
- mapping 3-9
  - Any type 3-25
  - arrays 3-11
  - boolean type 3-8
  - char type 3-8
  - constants 3-10
  - constructed types 3-11
  - enums 3-11
  - exceptions 3-23
  - floating point 3-9
  - Holder classes 3-5
  - IDL names 3-1
  - IDL type 3-3
  - integer 3-9
  - interface scope 3-23
  - interfaces 3-18
  - modules 3-3
  - nested types 3-25
  - octet 3-8
  - passing parameters 3-20
  - reserved names 3-2
  - reserved words 3-2
  - sequences 3-11
  - string 3-8, 3-9
  - structs 3-11
  - typedefs 3-25
  - unions 3-11
- member data in Holder 4-7
- messages
  - broadcasting via vbj command 2-9
- method
  - FullInterfaceDescription in FullInterfaceDescription 7-29
- methods 4-5
  - \_bind\_options in Object 5-12
  - \_boa in Object 5-12
  - \_create\_array\_tc in ORB 5-16
  - \_create\_request in Object 5-9
  - \_hash in Object 5-10
  - \_is\_a in Object 5-10
  - \_is\_bound in Object 5-12
  - \_is\_equivalent in Object 5-11
  - \_is\_local in Object 5-12
  - \_is\_persistent in Object 5-12
  - \_is\_remote in Object 5-12
  - \_non\_existent in Object 5-11
  - \_object\_name in Object 5-12
  - \_repository\_id in Object 5-13
  - \_request in Object 5-11
  - \_resolve\_reference in Object 5-13
  - \_work\_pending in ORB 5-22
  - absolute\_name in Contained 7-10
  - abstract\_base\_values in ValueDef 7-47
  - access in ValueMemberDef 7-51
  - activate in Activator 8-2
  - activate in PortableServer.POAManager 5-37
  - activate\_object in PortableServer.POA 5-26
  - activate\_object\_with\_id in PortableServer.POA 5-27
  - activation\_policy in CreationImplDef 8-5
  - activator\_obj in ActivationImplDef 8-1
  - add in ActiveObjectLifeCycleInterceptor 12-10
  - add in BindInterceptorManager 12-5
  - add in ChainUntypedObjectWrapperFactory 12-16
  - add in ClientRequestInterceptorManager 12-7
  - add in ContextList 6-5
  - add in ExceptionList 6-19
  - add in IORInterceptorManager 12-14
  - add in NVList 6-23
  - add in POALifeCycleInterceptorManager 12-8
  - add in ServerRequestInterceptorManager 12-12
  - add\_in\_arg in Request 6-26
  - add\_inout\_arg in Request 6-26
  - add\_item in NVList 6-23
  - add\_named\_in\_arg in Request 6-26
  - add\_named\_inout\_arg in Request 6-27
  - add\_named\_out\_arg in Request 6-27
  - add\_out\_arg in Request 6-27
  - add\_value in NVList 6-23
  - addClientObjectWrapper in Helper class 4-5
  - addServerObjectWrapperClass in Helper class 4-5
  - all\_agent\_locations in Agent 17-2
  - all\_instances in Agent 17-2
  - all\_instances\_descs in Agent 17-2
  - all\_replica in Agent 17-3
  - all\_replica\_descs in Agent 17-3
  - all\_repository\_ids in Agent 17-3
  - args in CreationImplDef 8-5
  - arguments in Request 6-27
  - assign in DynAny 6-7
  - AttributeDescription in AttributeDescription 7-6
  - base\_interfaces in InterfaceDef 7-1, 7-27
  - base\_value in ValueDef 7-47
  - bind in BindInterceptor 12-3
  - bind in Helper 4-4
  - bind in NamingContext 9-2
  - bind in ORB 5-23
  - bind\_context in NamingContext 9-3
  - bind\_failed in BindInterceptor 12-4
  - bind\_new\_context in NamingContext 9-5
  - bind\_succeeded in BindInterceptor 12-4
  - BindOptions in BindOptions 5-2
  - boa\_activate\_obj in OAD 8-9

boa\_deactivate\_obj in OAD 8-10  
 BOA\_init in ORB 5-23, 5-24  
 bound in SequenceDef 7-40  
 bound in StringDef 7-41  
 bound in WstringDef 7-51  
 change\_implementation in OAD 8-9  
 clear in Environment 6-18  
 completion status methods 5-5  
 connect in PortableRemoteObject 15-2  
 connect\_push\_supplier in  
   ProxyPushConsumer 10-5  
 containing\_repository in Contained 7-10  
 content\_type in TypeCode 6-32  
 contents in Container 7-14  
 context\_name in Context 5-6  
 contexts in OperationDef 7-33  
 contexts in Request 6-27  
 copy in DynAny 6-7  
 count in  
   ChainUntypedObjectWrapperFactory 12-16  
 count in ContextList 6-5  
 count in ExceptionList 6-19  
 count in NVList 6-23  
 create in ActiveObjectLifeCycleInterceptor 12-9  
 create in EventLibrary 10-3  
 create in IORInterceptor 12-13  
 create in POALifeCycleInterceptor 12-7  
 create in UntypedObjectWrapperFactory 12-18  
 create with specified boa 10-4  
 create\_abstract\_interface in Container 7-2  
 create\_alias in Container 7-15  
 create\_alias\_tc in ORB 5-15  
 create\_any in ORB 5-16  
 create\_array in Repository 7-39  
 create\_array\_tc in ORB 5-16  
 create\_attribute in ValueDef 7-48  
 create\_channel in EventLibrary 10-4  
 create\_child in Context 5-7  
 create\_constant in Container 7-15  
 create\_context in NamingContextFactory 9-11  
 create\_context\_list in ORB 5-16  
 create\_CreationImplDef in OAD 8-10  
 create\_dyn\_any in DynAnyFactory 6-10  
 create\_dyn\_any\_from\_type\_code in  
   DynAnyFactory 6-11  
 create\_enum in Container 7-15  
 create\_enum\_tc in ORB 5-16  
 create\_environment in ORB 5-16  
 create\_exception in Container 7-16  
 create\_exception\_tc in ORB 5-17  
 create\_id\_assignment\_policy in  
   PortableServer.POA 5-28  
 create\_id\_uniqueness\_policy in  
   PortableServer.POA 5-28  
 create\_implicit\_activation\_policy in  
   PortableServer.POA 5-27  
 create\_input\_stream in Any 6-1  
 create\_input\_stream in ORB 5-24  
 create\_interface in Container 7-15, 7-16  
 create\_interface\_tc in ORB 5-17  
 create\_lifespan\_policy in  
   PortableServer.POA 5-29  
 create\_list in ORB 5-17  
 create\_module in Container 7-16  
 create\_named\_value in ORB 5-17  
 create\_operation\_list in ORB 5-18  
 create\_operations in ValueDef 7-48  
 create\_output\_stream in Any 6-1  
 create\_output\_stream in ORB 5-24  
 create\_POA in PortableServer.POA 5-29  
 create\_recursive\_sequence\_tc in ORB 5-18  
 create\_reference in PortableServer.POA 5-27  
 create\_reference\_with\_id in  
   PortableServer.POA 5-28  
 create\_request\_processing\_policy in  
   PortableServer.POA 5-29  
 create\_sequence in Repository 7-39  
 create\_sequence\_tc in ORB 5-18  
 create\_servant\_retention\_policy in  
   PortableServer.POA 5-30  
 create\_string in Repository 7-39  
 create\_string\_tc in ORB 5-18  
 create\_struct in Container 7-17, 7-18, 7-19  
 create\_struct\_tc in ORB 5-18  
 create\_thread\_policy in  
   PortableServer.POA 5-30  
 create\_union in Container 7-17  
 create\_union\_tc in ORB 5-19  
 create\_value\_member in ValueDef 7-48  
 create\_wstring in Repository 7-39  
 create\_wstring\_tc in ORB 5-19  
 ctx in Request 6-27  
 ctx in ServerRequest 6-29  
 current\_component in DynAny 6-7  
 current\_member\_kind in DynStruct 6-14  
 current\_member\_kind in DynValue 6-16  
 current\_member\_name in DynStruct 6-14  
 current\_member\_name in DynValue 6-16  
 deactivate in Activator 8-2  
 deactivate in PortableServer.POAManager 5-37  
 deactivate\_obj in BOA 5-3  
 deactivate\_object in PortableServer.POA 5-31  
 def\_kind in IRObj 7-31  
 default\_bind\_options in ORB 5-24  
 default\_index in TypeCode 6-32  
 default\_principal in ORB 5-25  
 defined\_in in Contained 7-11  
 delegate in POATie 4-8  
 delete\_values in Context 5-7

- describe in Contained 7-11
- describe\_contents in Container 7-17
- describe\_interface in InterfaceDef 7-3, 7-28
- describe\_value in ValueDef 7-48
- destroy in
  - ActiveObjectLifeCycleInterceptor 12-9
- destroy in BindingIterator 9-10
- destroy in DynAny 6-7
- destroy in EventChannel 10-2
- destroy in IRObjct 7-31
- destroy in NamingContext 9-6
- destroy in POALifeCycleInterceptor 12-8
- destroy in PortableServer.POA 5-31
- destroy\_on\_unregister in OAD 8-10
- disconnect\_pull\_supplier 10-10, 10-11
- disconnect\_push\_consumer in
  - PullConsumer 10-8, 10-9
- disconnect\_push\_supplier in
  - PushSupplier 10-10
- discriminator\_kind in DynUnion 6-15
- discriminator\_type in TypeCode 6-32
- discriminator\_type in UnionDef 7-44
- discriminator\_type\_def in UnionDef 7-44
- element\_type in ArrayDef 7-4
- element\_type in SequenceDef 7-41
- element\_type\_def in ArrayDef 7-4, 7-5
- element\_type\_def in SequenceDef 7-41
- env in CreationImplDef 8-6
- env in Request 6-27
- equal in Any 6-2
- equal in TypeCode 6-32
- etherealize in
  - PortableServer.ServantActivator 5-37
- except in ServerRequest 6-30
- exception in Environment 6-18
- exception\_occurred in BindInterceptor 12-4
- exceptions in OperationDef 7-33
- exceptions in Request 6-27
- extract in Helper 4-3
- extraction methods in Any 6-2, 6-8
- find\_POA in PortableServer.POA 5-32
- flags in NamedValue 6-21
- for\_suppliers in EventChannel 10-2
- force\_register\_url in Resolver 16-2
- from\_any in DynAny 6-7
- from\_int in TCKind 6-31
- generated\_command in OAD 8-10
- generated\_environment in OAD 8-10
- get\_policy\_overrides in Object 13-4
- get\_as\_string in DynEnum 6-11
- get\_as\_ulong in DynEnum 6-11
- get\_client\_policy in Object 13-4
- get\_cluster\_manager in
  - NamingContextFactory 9-11
- get\_default\_context in ORB 5-19
- get\_discriminator in DynUnion 6-15
- get\_elements in DynArray 6-10
- get\_elements in DynSequence 6-13
- get\_implementation in OAD 8-11
- get\_manager in
  - InterceptorManagerControl 12-3
- get\_members in DynStruct 6-14
- get\_members in DynValue 6-16
- get\_members\_as\_dyn\_any in DynStruct 6-14
- get\_members\_as\_dyn\_any in DynValue 6-16
- get\_next\_response in ORB 5-19
- get\_object in PortableServer.Current 5-26
- get\_POA in PortableServer.Current 5-26
- get\_policy in Object 13-3
- get\_policy\_overrides in PolicyManager 13-2
- get\_primitive in Repository 7-40
- get\_primitive\_tc in ORB 5-19
- get\_principal in BOA 5-4
- get\_response in Request 6-28
- get\_servant in PortableServer.POA 5-32
- get\_servant\_manager in
  - PortableServer.POA 5-32
- get\_status in OAD 8-11
- get\_status\_all in OAD 8-11, 8-12
- get\_status\_interface in OAD 8-11
- get\_value in DynFixed 6-12
- get\_values in Context 5-7
- has\_no\_active\_member in DynUnion 6-16
- hold requests in
  - PortableServer.POAManager 5-37
- Holder in Holder 4-7
- id in ActivationImplDef 8-1
- id in Contained 7-11
- id in CreationImplDef 8-6
- id in Helper 4-3
- id in TypeCode 6-32
- id\_to\_reference in PortableServer.POA 5-32
- id\_to\_servant in PortableServer.POA 5-32
- impl\_is\_down in TriggerHandler 17-8
- impl\_is\_ready in BOA 5-4
- impl\_is\_ready in TriggerHandler 17-8
- incarnate in
  - PortableServer.ServantActivator 5-38
- init in ORB 5-20, 5-25
- initializers in ValueDef 7-47
- insert in Helper 4-3
- insertion methods in Any 6-3, 6-8
- invoke in DynamicImplementation 6-17
- invoke in Request 6-28
- is\_a in InterfaceDef 7-3, 7-28
- is\_a in ValueDef 7-48
- is\_abstract in ValueDef 7-47
- is\_custom in ValueDef 7-47
- is\_trucatable in ValueDef 7-47
- item in ContextList 6-5

- item in ExceptionList 6-19
- item in NVList 6-24
- kind in PrimitiveDef 7-37
- kind in TypeCode 6-33
- length in ArrayDef 7-4
- length in TypeCode 6-33
- list in NamingContext 9-6
- list\_all\_roots in NamingContextFactory 9-11
- list\_initial\_services in ORB 5-20
- locate in Resolver 16-2
- lookup in Container 7-18
- lookup\_id in Repository 7-40
- lookup\_name in Container 7-18
- member in DynUnion 6-15
- member\_count in TypeCode 6-33
- member\_kind in DynUnion 6-15
- member\_label in TypeCode 6-33
- member\_name in DynUnion 6-15
- member\_name in TypeCode 6-33
- member\_type in TypeCode 6-34
- members in EnumDef 7-21
- members in ExceptionDef 7-22
- members in StringDef 7-41
- members in StructDef 7-42
- members in UnionDef 7-45
- members in WstringDef 7-51
- mode in AttributeDef 7-5
- mode in OperationDef 7-33
- move in Contained 7-11
- name in Contained 7-11
- name in NamedValue 6-21
- name in Principal 5-41
- name in StructMember 7-43
- name in TypeCode 6-34
- narrow in Helper 4-5
- narrow in PortableRemoteObject 15-2
- new\_context in NamingContext 9-5
- next in DynAny 6-7
- next\_n in BindingIterator 9-10
- next\_one in BindingIterator 9-10
- nil in ORB 5-25
- obj\_is\_ready in BOA 5-5
- object\_name in CreationImplDef 8-6
- object\_to\_string in ORB 5-20
- obtain\_pull\_consumer in SupplierAdmin 10-11
- obtain\_pull\_consumer() 10-11
- obtain\_pull\_supplier 10-2
- obtain\_push\_consumer in SupplierAdmin 10-11
- obtain\_push\_consumer() 10-11
- obtain\_push\_supplier 10-2
- op\_name in ServerRequest 6-30
- operation in Request 6-28
- ORB 5-23, 5-25
- ORB\_init()
  - ORBshmsize 2-9
- original\_type\_def in AliasDef 7-4
- original\_type\_def in ValueBoxDef 7-46
- params in OperationDef 7-33
- params in ServerRequest 6-30
- parent in Context 5-7
- path\_name in CreationImplDef 8-6
- perform\_work in ORB 5-20
- POA 5-26
- poa in POATie 4-8
- poll\_next\_response in ORB 5-20
- poll\_response in Request 6-28
- post\_method in UntypedObjectWrapper 12-17
- postinvoke in ClientRequestInterceptor 12-6
- postinvoke in
  - PortableServer.ServantLocator 5-39
- postinvoke\_premarshal in
  - ServerRequestInterceptor 12-11
- pre\_method in UntypedObjectWrapper 12-17
- preinvoke in
  - PortableServer.ServantLocator 5-39
- preinvoke in ServerRequestInterceptor 12-11
- preinvoke\_postmarshal in
  - ClientRequestInterceptor 12-6
- preinvoke\_premarshal in
  - ClientRequestInterceptor 12-5
- pull in PullSupplier 10-10
- read in Helper 4-3
- read methods in InputStream 6-19
- read\_estruct in InputStream 6-20
- read\_value in Any 6-2
- rebind in NamingContext 9-3
- rebind\_context in NamingContext 9-4
- reference\_to\_id in PortableServer>POA 5-33
- reference\_to\_servant in
  - PortableServer.POA 5-33
- reg\_implementation in OAD 8-12, 8-13
- reg\_trigger in Agent 17-4
- register\_url in Resolver 16-2
- RemoteToStub in PortableRemoteObject 15-2
- remove in
  - ChainUntypedObjectWrapperFactory 12-16
- remove in ContextList 6-6
- remove in ExceptionList 6-19
- remove in NVList 6-24
- remove\_state\_contexts in
  - NamingContextFactory 9-11
- removeServerObjectWrapperClass in Helper
  - class 4-6
- repository\_id in CreationImplDef 8-7
- resolve in NamingContext 9-4
- resolve\_initial\_references in ORB 5-20
- resolve\_str in NamingContextExt 9-8
- result in OperationDef 7-33
- result in Request 6-28

- result in ServerRequest 6-30
- result\_def in OperationDef 7-33
- return\_value in Request 6-28
- rewind in DynAny 6-7
- root\_context in
  - ExtendedNamingContextFactory 9-12
- run in ORB 5-21
- seek in DynAny 6-7
- send\_deferred in Request 6-28
- send\_multiple\_requests\_deferred in ORB 5-21
- send\_multiple\_requests\_oneway in ORB 5-22
- send\_oneway in Request 6-28
- servant\_to\_id in PortableServer.POA 5-33
- servant\_to\_reference in
  - PortableServer.POA 5-34
- service\_name in ActivationImplDef 8-1
- set\_as\_string in DynEnum 6-11
- set\_as\_ulong in DynEnum 6-12
- set\_discriminator in DynUnion 6-15
- set\_elements in DynArray 6-10
- set\_elements in DynSequence 6-13
- set\_length in DynSequence 6-13
- set\_member in DynStruct 6-14
- set\_members\_as\_dyn\_any in DynStruct 6-14
- set\_members\_as\_dyn\_any in DynValue 6-16
- set\_one\_value in Context 5-7
- set\_policy\_overrides in Object 13-4
- set\_policy\_overrides in PolicyManager 13-2
- set\_return\_type in Request 6-29
- set\_servant in PortableServer.POA 5-34
- set\_servant\_manager in
  - PortableServer.POA 5-35
- set\_to\_default\_member in DynUnion 6-15
- set\_to\_no\_active\_member in DynUnion 6-16
- set\_value in DynFixed 6-12
- set\_values in Context 5-8
- shutdown in NamingContextFactory 9-11
- string\_to\_object in ORB 5-22
- supported\_interfaces in ValueDef 7-47
- target in Request 6-29
- the\_activator in PortableServer.POA 5-35
- the\_name in PortableServer.POA 5-35
- the\_parent in PortableServer.POA 5-35
- the\_POAManager in PortableServer.POA 5-35
- the\_policies in PortableServer.POA 5-36
- to\_any in DynAny 6-8
- to\_name in NamingContextExt 9-7
- to\_string in NamingContextExt 9-7
- to\_url in NamingContextExt 9-8
- toString in Desc 17-6
- toString in TriggerDesc 17-7
- try\_pull in PullSupplier 10-10
- type in Any 6-2
- type in AttributeDef 7-5
- type in ConstantDef 7-8
- type in DynAny 6-8
- type in ExceptionDef 7-22
- type in Helper 4-4
- type in IDLType 7-26
- type in ValueMemberDef 7-50
- type\_def in AttributeDef 7-5
- type\_def in ConstantDef 7-8
- type\_def in ValueMemberDef 7-50
- TypeDescription in TypedefDef 7-44
- unbind in NamingContext 9-5
- unexportedObject in
  - PortableRemoteObject 15-2
- UnionMember in UnionMember 7-45
- unknown\_adaptor in
  - PortableServer.AdapterActivator 5-25
- unreg\_implementation in OAD 8-13
- unreg\_interface in OAD 8-14
- unreg\_trigger in Agent 17-4
- unregister\_all in OAD 8-14
- validate\_connection in Object 13-4
- value in ConstantDef 7-8
- value in DefinitionKind 7-20
- value in NamedValue 6-21
- value in PrimitiveKind 7-38
- value in TCKind 6-31
- ValueDescription in ValueDescription 7-50
- ValueMember in ValueMemberDef 7-50
- version in Contained 7-11
- void length in ArrayDef 7-4
- void type in Any 6-2
- void\_the\_activator in PortableServer.POA 5-35
- write in Helper 4-4
- write methods in OutputStream 6-24
- write\_estruct in OutputStream 6-25
- write\_value in Any 6-2
- ModuleDef interface 7-31
- ModuleDescription class 7-31
- modules
  - mapping 3-3

## N

---

- NamedValue class 6-21
- NameValuePair
  - classes 6-22
- NamingContext 9-3, 9-4, 9-5, 9-6
  - class 9-1
  - methods
    - bind 9-2
    - rebind 9-3
- NamingContextExt
  - interface 9-6
- NamingContextFactory
  - class 9-10
  - interface 9-10
  - methods 9-11

- NativeDef
  - interface 7-32
- nested types
  - mapping 3-25
- null
  - Java 3-8
- NVList
  - classes 6-22
- NVList interface 6-22

## O

---

- OACConnectionMax
  - ORB initialization option A-3
- OAcconnectionMaxIdle
  - ORB initialization option A-3, A-6
- OAD
  - properties B-8
- OAD interface 8-7
- OAid
  - BOA initialization option A-3
- OAipaddr A-4
- OApport
  - ORB initialization option A-4
- OAtcpNoDelay
  - ORB initialization option A-7
- OAThreadMax
  - ORB initialization option A-4
- OAThreadMaxIdle
  - ORB initialization option A-4
- OAThreadMin
  - ORB initialization option A-4
- Object
  - vbroker.CORBA interface 13-4
  - VisiBroker extension 5-11
- Object interface 5-8
- object server
  - activation policies 8-3
- object wrappers
  - adding typed object wrappers for clients 4-5
  - adding typed object wrappers for servers 4-5
  - generated interfaces 4-5
  - removing typed object wrappers for clients 4-5
  - removing typed object wrappers for servers 4-6
- objects
  - activating 5-2
  - CORBA interface 13-3
  - deactivating 5-2
- obtain\_pull\_consumer()
  - method 10-11
- obtain\_push\_consumer()
  - method 10-11
- octet
  - mapping 3-8
- OMG
  - ORB definition 5-22

- OperationDef interface 7-32
- OperationDescription class 7-34
- OperationMode class 7-35
- operations
  - interface\_name 4-3
  - reporting status 5-5
- operations classes
  - description 4-2
- options
  - BOA A-3
  - Location Service A-7
  - ORB A-5
- ORB
  - backward compatibility A-5, A-6
  - initialization options A-1, A-4
  - methods 5-23 to 5-25
  - OMG definition 5-22
  - options A-5
  - properties B-3
  - VisiBroker extensions 5-22
- ORB class 5-13
- ORB.init A-4
- ORB\_init() method
  - ORBshmsize 2-9
- ORBagentaddr
  - ORB initialization option A-5
- ORBagentPort A-5
- ORBagentport
  - ORB initialization option A-5
- ORBbackcompat
  - description of behavior A-7
  - ORB initialization option A-5, A-6
- ORBconnectionMax
  - ORB initialization option A-5
- ORBdebug
  - ORB initialization option A-6
- ORBdisableLocator
  - ORB initialization option A-6
- ORBgatekeeperIOR
  - ORB initialization option A-6
- ORBmbufSize A-6
- ORBmbufsize
  - ORB initialization option A-6
- ORBservices
  - ORB initialization option A-6
- ORBTcpTimeout
  - ORB initialization option A-7
- OSAgent
  - locating via vbj command 2-9
  - properties B-2
- OSAGENT\_ADDR 2-9
- OSAGENT\_PORT 2-9
- OutputStream interface 6-24

# P

---

ParameterDescription class 7-36  
ParameterMode class 7-37  
parameters  
    mapping 3-20  
platform designation with icons 1-4  
POA  
    classes 4-2  
    interface\_name 4-7  
    interfaces 5-26  
    methods 5-26  
    PortableObject Adaptor 5-26  
    PortableServer.AdaptorActivator 5-25  
    properties B-7  
POATie  
    classes 4-2  
    interface\_name 4-8  
policy values  
    Quality of Service 13-6  
PolicyCurrent  
    interface 13-3  
PolicyManager  
    interface 13-1  
port number A-4, A-5  
PortableRemoteObject  
    interface 15-1  
PortableServer.AdaptorActivator  
    classes 5-25  
PrimitiveDef interface 7-37  
PrimitiveKind class 7-37  
Principal class 5-40  
ProfileBody  
    class 14-1  
programmer tools  
    idl2ir 2-1  
    ir2idl 2-2  
programming interface  
    BindInterceptor 12-3  
    ClientInterceptor 12-7, 12-8  
properties  
    Client-Side Connection B-10  
    Client-Side In-Process Connection B-10  
    Event Service B-8  
    Interface Repository B-9  
    Java RMI over IIOP B-1  
    Location Service B-7  
    OAD B-8  
    ORB B-3  
    OSAgent B-2  
    POA B-7  
    Server Manager B-7  
    Server-Side Server Engine B-11, B-14  
    Server-Side Thread Pool BOA\_TP  
        Connection B-14

    Server-Side Thread Pool IIOP\_TP  
        Connection B-13  
    Server-Side Thread Session BOA\_TS  
        Connection B-12  
    Server-Side Thread Session IIOP\_TS  
        Connection B-11  
    setting with BOA A-2  
    URL Naming B-9  
ProxyPullConsumer  
    class 10-4  
    interface 10-4  
ProxyPullSupplier  
    interface 10-6, 10-7  
ProxyPushConsumer  
    interface 10-5  
ProxyPushSupplier  
    interface 10-7  
ProxyPushSupplier methods  
    connect\_push\_consumer 10-7  
PullConsumer  
    interface 10-7, 10-8, 10-9  
PullSupplier  
    interface 10-9  
    methods 10-10, 10-11  
push()  
    method 10-8  
PushConsumer  
    interface 10-8  
PushConsumer methods  
    disconnect\_push\_consumer() 10-8  
    push() 10-8  
PushSupplier  
    interface 10-10

# Q

---

QoS  
    Quality of Service 13-1  
Quality of Service  
    QoS 13-1

# R

---

RebindPolicy  
    interface 13-5  
references 1-5  
RelativeConnectionTimeout 13-8  
removeClientObjectWrapperClass in Helper  
    class 4-5  
Repository interface 7-38  
Request interface 6-25  
RequestHeader 14-1  
reserved  
    keywords 3-2  
reserved names  
    mapping 3-2



- reserved words
  - mapping 3-2
- resolve\_initial\_references 5-20
- Resolver interface 16-1

## S

---

- send\_multiple\_requests\_oneway 5-22
- SequenceDef interface 7-40
- sequences
  - mapping 3-11
- Server Manager
  - properties B-7
- ServerRequest interface 6-29
- servers
  - using inheritance 3-21
- Server-Side Server Engine
  - properties B-11, B-14
- Server-Side Thread Pool BOA\_TP Connection
  - properties B-14
- Server-Side Thread Pool IIOP\_TP Connection
  - properties B-13
- Server-Side Thread Session BOA\_TS Connection
  - properties B-12
- Server-Side Thread Session IIOP\_TS Connection
  - properties B-11
- ServiceContext
  - class 14-3
- sockets
  - batching requests A-7
- string
  - mapping 3-8
- StringDef interface 7-41
- struct
  - ImplementationStatus 8-8
- StructDef interface 7-42
- StructMember
  - interface 7-42
- structs
  - mapping 3-11
- structure
  - GIOP::RequestHeader 14-1
  - TriggerDesc 17-6
- stub
  - interface\_name 4-7
- stubs
  - classes 4-2
  - generating portable for DII 2-3, 2-7
- SupplierAdmin
  - interface 10-11
- support options 1-5
- symbols
  - ellipsis (...) 1-4
  - vertical bar | 1-4
- SystemException class 11-1

## T

---

- TaggedProfile
  - class 14-4
- TCKind class 6-30
- technical support 1-5
- templates
  - IOR 12-2
- threading A-4
  - idle time A-4
  - thread policy A-3
- timeout policy 13-8
- tools
  - idl2ir 2-1, 2-2
  - idl2java 2-3
  - java2idl 2-5
  - java2iiop 2-6
  - vbj 2-9
- TriggerDesc 17-6
- TriggerHandler interface 17-8
- TSession A-4
- type extensions 3-4
- TypeCode interface 6-31
- TypedefDef interface 7-43
- typedefs
  - mapping 3-25
- TypeDescription class 7-43
- types
  - mapping 3-3
- typographic conventions 1-4

## U

---

- UnionDef interface 7-44
- UnionMember class 7-45
- unions
  - mapping 3-11
- UnknownUserException class 6-34
- URL associating with IOR 16-2
- URL Naming
  - properties B-9
  - Resolver interface 16-1
- UserException class 11-3

## V

---

- ValueDef
  - interfaces 7-46
- valuetype
  - ActivationImplDef 8-1
- variables
  - defined\_in in AttributeDescription 7-6
  - id in NameValuePAir 6-22
  - mode in AttributeDescription 7-6
  - name in AttributeDescription 7-6
  - string Id in AttributeDescription 7-6

- type in AttributeDescription 7-6
- value in NameValuePair 6-22
- version in AttributeDescription 7-6
- vbj
  - using A-2
  - using command-line arguments A-1
- vbj command
  - description 2-9
- version of product 2-1, 2-2, 2-3, 2-5, 2-6, 2-9
- VisiBroker for Java
  - additional information 1-5

## W

---

- web naming
  - Resolver interface 16-1
- web sites
  - CORBA specification 1-5
- What's New 1-1
- words
  - reserved 3-2
- wstring
  - mapping 3-9
- WstringDef interface 7-51