TPBroker VisiBroker for Java Reference Manual

Release 3.1.1
Contents

Chapter 1
Preface 1-1
Overview ................................. 1-1
Organization of this manual ............ 1-1
Typographic conventions ............... 1-2
Platform conventions ................. 1-3
Syntax conventions .................. 1-3
Where to find additional information ... 1-3
Contacting Hitachi Technical Support ... 1-4

Chapter 2
Programmer tools 2-1
idl2ir .................................. 2-1
idl2java ................................ 2-2
java2idl ............................... 2-5
java2iiop .............................. 2-5
vbj ...................................... 2-6

Chapter 3
IDL to Java mapping 3-1
Names .................................. 3-2
Reserved names ....................... 3-2
Reserved words ....................... 3-3
Modules ................................ 3-3
Basic types ............................ 3-4
  IDL type extensions ............... 3-4
  Holder classes ..................... 3-5
  Boolean ................................ 3-8
  Char .................................. 3-8
  Octet ................................ 3-8
  String ................................ 3-9
  WString .............................. 3-9
  Integer types ....................... 3-9
  Floating point types ............... 3-9
Helper classes ....................... 3-9
Constants ............................ 3-10
  Constants within an interface .... 3-11
  Constants NOT within an interface . 3-11
Constructed types ................... 3-11
  Enum ................................ 3-12
  Struct ................................ 3-13
  Union ................................ 3-14
  Sequence ............................ 3-16
  Array ................................ 3-17
Interfaces ............................. 3-18
  Passing parameters................ 3-19
Server implementation with inheritance . 3-20
Server implementation with delegation . 3-21
Interface scope ...................... 3-22
Mapping for exceptions ............... 3-22
User-defined exceptions ............. 3-23
System exceptions ................... 3-24
Mapping for the Any type ............ 3-24
Mapping for certain nested types .... 3-24
Mapping for Typedef ................ 3-25
Simple IDL types ...................... 3-25
Complex IDL types .................... 3-25

Chapter 4
Generated classes 4-1
Overview ................................ 4-1
  Helper class ......................... 4-2
  Holder class ......................... 4-2
  Stub class ............................ 4-2
  ImplBase class ...................... 4-2
  Skeleton class ....................... 4-2
  Tie class ................................ 4-2
  Operations class .................... 4-2
  Examples class ...................... 4-3
  <interface_name>Helper .......... 4-3
    Methods for all Helper classes .... 4-3
    Methods generated for interfaces .. 4-4
    Methods generated for object wrappers . 4-5
  <interface_name>Holder ............ 4-6
    Member data ....................... 4-7
    Methods ............................ 4-7
  <interface_name>Operations .......... 4-7
    _st_<interface_name> ............. 4-8
    _sk_<interface_name> ............. 4-8
    _impl_<interface_name> .......... 4-8
    _tie_<interface_name> .......... 4-8

Chapter 5
Core interfaces and classes 5-1
BindOptions ............................ 5-1
  Constructors ......................... 5-2
  BOA .................................. 5-2
  IDL definition ........................ 5-3
  Methods ............................. 5-4
CompletionStatus ...................... 5-5
  IDL definition ........................ 5-5
  Methods ............................. 5-5
Chapter 6

Dynamic interfaces and classes  6-1

Any  ...........................................  6-2
Methods ....................................  6-2
Extraction methods  ......................  6-3
Insertion methods  .......................  6-4
ARG_IN ..................................  6-4
Variables ..................................  6-5
ARG_INOUT  .................................  6-5
Variables ..................................  6-5
ARG_OUT ..................................  6-5
Variables ..................................  6-5
ContextList  ...............................  6-5
IDL definition  ..............................  6-5
Methods ...................................  6-6
DynAny ....................................  6-6
   Important usage restrictions ........  6-7
   Methods ................................  6-7
   Extraction methods ....................  6-8
   Insertion methods ......................  6-9
DynArray ..................................  6-10
   Important usage restrictions ........  6-10
   Methods ................................  6-10
DynEnum ..................................  6-10
   Important usage restrictions ........  6-11
   Methods ................................  6-11
DynSequence ................................  6-11
   Important usage restrictions ........  6-12
   Methods ................................  6-12
DynStruct ..................................  6-12
   Important usage restrictions ........  6-13
   Methods ................................  6-13
DynUnion ..................................  6-13
   Important usage restrictions ........  6-14
   Methods ................................  6-14
DynamicImplementation ..................  6-15
   Constructors ................................  6-15
   Methods ................................  6-15
Environment ................................  6-16
   Methods ................................  6-16
ExceptionList  ............................  6-16
   IDL definition  ..........................  6-16
   Methods ................................  6-17
InputStream ................................  6-17
   Methods ................................  6-17
Invalid ....................................  6-18
InvalidSeq ................................  6-19
NamedValue ................................  6-19
   IDL definition  ..........................  6-19
   Methods ................................  6-19
NameValuePair ................................  6-20
   Variables ................................  6-20
   Constructors  .............................  6-20
NVList .....................................  6-20
   IDL definition  ..........................  6-20
   Methods ................................  6-21
OutputStream ................................  6-22
   Methods ................................  6-22
Request ....................................  6-23
   IDL definition  ..........................  6-24
   Methods ................................  6-24
ServerRequest  ............................  6-27
   IDL definition  ..........................  6-27
   Methods ................................  6-27
TCKind .....................................  6-28
   IDL definition  ..........................  6-28
   Methods ................................  6-28
TypeCode ..................................  6-29
   IDL definition  ..........................  6-29
   Methods ................................  6-29
UnknownUserException ....................  6-32

Chapter 7

Interface repository  7-1

AliasDef ..................................  7-2
   Methods ................................  7-3
ArrayDef ..................................  7-3
   Methods ................................  7-3
AttributeDef ................................  7-4
   Methods ................................  7-4
AttributeDescription .....................  7-4
   Variables ................................  7-5
   Methods ................................  7-5
Chapter 8

Activation interfaces

8-1
| Chapter 9 | Event handler interfaces and classes | 9-1 |
| --- | --- | --- |
| ClientEventHandler | IDL definition | 9-1 |
| Methods | 9-2 |
| HandlerRegistry | IDL definition | 9-3 |
| Methods | 9-4 |
| ImplEventHandler | IDL definition | 9-6 |
| Methods | 9-7 |

| Chapter 10 | Exceptions classes | 10-1 |
| --- | --- | --- |
| Introduction | . | 10-1 |
| SystemException | Attributes | 10-2 |
| UserException | Constructor | 10-3 |

| Chapter 11 | Interceptor and object wrapper interfaces and classes | 11-1 |
| --- | --- | --- |
| BindInterceptor | Methods | 11-1 |
| ChainImplDef | IDL definition | 11-4 |
| Activation policy | Examples | 11-4 |
| UnTypedObjectWrapperFactory | Methods | 11-8 |

| Chapter 12 | GIOP, IOP, and IIOP interfaces | 12-1 |
| --- | --- | --- |
| GIOP.ReplyHeader | IDL definition | 12-1 |
| Variables | 12-2 |
| Constructors | 12-2 |
| GIOP.ReplyStatusType | IDL definition | 12-2 |
| Variables | 12-3 |
| Constructors | 12-3 |
| GIOP.RequestHeader | IDL definition | 12-3 |
| Variables | 12-4 |
| Constructors | 12-4 |
| GIOP.Version | IDL definition | 12-4 |
| Variables | 12-4 |
| Constructors | 12-5 |
| IIO.1.ProfileBody | IDL definition | 12-5 |
| Variables | 12-5 |
| Constructors | 12-6 |
| IOP.1.1.ProfileBody | IDL definition | 12-6 |
| Variables | 12-6 |
| Constructors | 12-6 |
| IOP.ServiceContext | IDL definition | 12-7 |
| Variables | 12-7 |
| Constructors | 12-7 |
| IOP.TaggedProfile | IDL definition | 12-7 |
| Variables | 12-7 |
| Constructors | 12-8 |
| IIOP.IOR | IDL definition | 12-8 |
| Variables | 12-8 |
| Constructors | 12-8 |

| Chapter 13 | URL Naming interface | 13-1 |
| --- | --- | --- |
| Resolver | Methods | 13-1 |
| URL Naming interface | Methods | 13-2 |

| Chapter 14 | Location Service interfaces | 14-1 |
| --- | --- | --- |
| Agent | Methods | 14-1 |
| Location Service interfaces | Methods | 14-2 |
Tables

3.1 Basic type mappings ................. 3-4
3.2 Summary of supported IDL extensions . 3-4
3.3 IDL extensions for new types .......... 3-5
7.1 AttributeDescription globals and constants ....................... 7-5
7.2 AttributeMode globals and constants .......... 7-6
7.3 DefinitionKind globals and constants .......... 7-17
7.4 OperationMode globals and constants .......... 7-31
7.5 AttributeMode globals and constants .......... 7-33
10.1 List of system exceptions .............. 10-2
15.1 Common attributes for both TPool and TSession adapters .......... 15-2
15.2 Attributes for TPool adapters .......... 15-2
15.3 Read-only attributes for the Server interface ....................... 15-5
15.4 Read-write attributes for the Server interface ....................... 15-6
A.1 BOA_init options ....................... A-3
A.2 ORB.init options ....................... A-5
A.3 Location Service options ....................... A-8
Preface

This reference provides information on the classes supplied with VisiBroker for Java.

The Preface describes typographical and syntax conventions used throughout the manual and provides references for more information about Common Object Request Broker Architecture (CORBA).

Organization of this manual

- Page 1-1

- Typographic conventions
- Platform conventions
- Syntax conventions
- Where to find additional information
- Contacting Hitachi Technical Support

Overview

VisiBroker for Java allows you to develop and deploy applets that use distributed objects, as defined in the Common Object Request Broker specification.

Organization of this manual

This manual includes the following sections:

- Chapter 2, “Programmer tools,” describes the programming tools offered by VisiBroker for Java.
- Chapter 3, “IDL to Java mapping,” describes the IDL-to-Java mapping used by VisiBroker for Java.
Typographic conventions

- Chapter 4, “Generated classes,” describes the classes generated by the idl2java compiler.
- Chapter 5, “Core interfaces and classes,” describes all of the interfaces in the CORBA package.
- Chapter 6, “Dynamic interfaces and classes,” describes the dynamic interfaces and classes in the org.omg.CORBA package.
- Chapter 7, “Interface repository,” describes the interfaces and classes in the org.omg.CORBA package that are used with the interface repository.
- Chapter 8, “Activation interfaces,” describes the CreationImplDef, ImplementationDef, and OAD—dynamic interfaces and classes in the Activation package. They are used with the Object Activation Daemon (OAD).
- Chapter 9, “Event handler interfaces and classes,” describes the interfaces and classes that allow clients and servers to monitor the occurrence of specific ORB-related events.
- Chapter 10, “Exceptions classes,” describes the exception classes used in VisiBroker.
- Chapter 11, “Interceptor and object wrapper interfaces and classes,” describes the classes and interfaces that developers use when implementing interceptors.
- Chapter 12, “GIOP, IOP, and IIOP interfaces,” describes the CORBA-defined header and message formats.
- Chapter 13, “URL Naming interface,” describes the Resolver class used when implementing the Web Naming Service.
- Chapter 14, “Location Service interfaces,” provides detailed information about the Agent and TriggerHandler interfaces which can be used to locate object instances on a network of Smart Agents.
- Chapter 15, “ORB Manager interfaces.”
- Appendix A, “Using command-line options.”

Typographic conventions

This manual uses the following conventions:

| Convention | Used for |
|------------|----------|
| boldface   | Bold type indicates that syntax should be typed exactly as shown. It is also used in code samples to highlight information of particular interest. For UNIX, bold type indicates database names, filenames, and similar terms. |
| italics    | Italic indicates information that the user or application provides, such as variables in syntax diagrams. It is also used to introduce new terms. |
| computer   | Computer typeface is used for sample command lines and code. |
Platform conventions

This manual uses the following symbols to indicate that information is platform-specific:

- **W** All Windows platforms including Windows 3.1, Windows NT, and Windows 95
- **NT** Windows NT only
- **95** Windows 95 only
- **U** All UNIX platforms

Syntax conventions

To view the usage syntax and a list of options for a command, enter the command followed by a dash and the word “help.” For example,

```
prompt>gatekeeper -help
```

The usage syntax and a list of options appear. For example,

```
Usage: idl2ir [-options] files...
where options include:
    -ir <name>The name of the interface repository
    -replace Replace definitions, instead of updating them
```

Usage and options are provided in this chapter. Here are the conventions used:

| Convention | Used for                      |
|------------|-------------------------------|
| [ ]        | Brackets indicate optional items. |
| { }        | Curly brackets indicate required information. |
| | A vertical bar separates two mutually exclusive choices. |

Where to find additional information

For more information about VisiBroker, refer to the following sources:

- *VisiBroker for Java Programmer’s Guide*. This guide contains the information on the developing applications using VisiBroker for Java.
Contacting Hitachi Technical Support

Hitachi offers a variety of support options to help you get the most from your Hitachi products.

North America

If you have purchased support for your Hitachi products, contact Hitachi’s Technical Support group:

- Phone: 1-800-745-4056, or
- E-Mail: cssc@hi.com
If you have not purchased support for your Hitachi products

- See the Hitachi website at http://www.hitachisoft.com, then click on ‘Support,’ then ‘Contact Information.’
- Contact our Sales Department at 1-800-558-1413.

Please be prepared to provide complete information about your environment, the version of the Hitachi product you are using, and a detailed description of the problem.

Europe

If you have purchased support for your Hitachi products, contact Hitachi’s Technical Support group:

- Phone: +44-(0)1628-585777
- E-Mail: tech-asc.support@hitachi-eu.com

If you have not purchased support for your Hitachi products:

- See the Hitachi website at http://www.hitachisoft.com, then click on ‘Support,’ then ‘Contact Information.’
- Contact our Sales Department at +44-(0)1628-585335, or
- Send an email to marketing-asc.sales@hitachi-eu.com for general information.
Contacting Hitachi Technical Support
Chapter 2

Programmer tools

This chapter describes the programmer tools offered by VisiBroker for Java. For information about the syntax used with these tools, see the Preface in this manual. This chapter includes the following major sections:

- idl2ir page 2-1
- idl2java page 2-2
- java2idl page 2-5
- java2iiop page 2-5
- vbj page 2-6

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 -ir 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 the IDL file. If the repository already contains an item with the same name as an item in the IDL file, the old item will be replaced.
Options

The following options are available for idl2ir.

| Option   | Description                                                                                                                                 |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------|
| -version | Displays or prints out the version of the following programs that your computer is currently running: VisiBroker for Java, Java, and its operating system. |
| -ir name | Specifies the instance name of the interface repository to which idl2ir will attempt to bind. If no name is specified, idl2ir will bind itself to the interface repository server found in the current domain. The current domain is defined by the OSAGENT_PORT environment variable. |
| -replace | Replaces definitions instead of updating them.                                                                                               |
| -verbose | Turns verbose mode on.                                                                                                                        |

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 page 2–7.

Arguments

The supported arguments for the idl2ir preprocessor are shown in the following table.

| Argument   | Description                                                                                                                                   |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| -C         | Retains comments from IDL file when the Java code is generated. Otherwhise, the comments will not appear in the Java code.                     |
| -H         | Prints the full paths of included files on the standard error output.                                                                         |
| -P         | Suppresses the generation of line number information.                                                                                         |
| -d         | Turns on preprocessor debugging mode.                                                                                                         |
| -D name    | Adds definition for symbol name.                                                                                                             |
| -D name=definition | Defines symbol. For example, #define name def.                                                        |
| -U name    | Removes the definition for symbol name.                                                                                                       |
| -I directory | Takes a directory as input when searching for #include files.                                                                                 |

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                                                                                                                                                                                                                                                                                                                                 |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -package name              | Generates code in the specified package. The package name for definitions is prepended with the specified package name. If a directory with the specified package name does not exist, it will be created. If the package directory exists, its contents will be updated. When this option is not set, idl2java generates code using the CORBA package resolution rules. |
| -root_dir directory        | Places all generated Java files in the specified directory, instead of in the current directory.                                                                                                                                                                                                                                  |
| -idl2package idl pkg       | Puts definitions in scope of idl into the specified Java package, for example: -idl2package ::CORBA org.omg.CORBA. When this option is not set, idl2java generates code using the CORBA package resolution rules.                                                                                       |
| -version                   | Displays the version of VisiBroker for Java.                                                                                                                                                                                                                                                                                         |
| -portable                  | Generates portable stubs using DII. Without this option set, idl2java uses proprietary Visigenic features to generate stubs specifically for use in ORB environments.                                                                                                                                                        |
| -strict                    | Generates code which can run on any CORBA-compliant ORB. This option is equivalent to setting -portable, -no_toString, and -no_bind while also not setting -serializable, -smart_stub, and -obj_wrappers. When -strict is not set, idl2java does not generate portable code. It also uses VisiBroker extensions and optimizations in the code it generates. |
| -smart_stub                | Generates the ability to use smart stubs. It creates additional methods in the interface Helper class. When this option is not set, idl2java does not generate code for installing or using smart stubs.                                                                                                    |
| -obj_wrapper               | Generates stubs, skeletons and helpers with object wrapper support. It also generates the base typed object wrapper from which all other object wrappers inherit, and a default object wrapper that performs the untyped object wrapper calls. When this option is not set, idl2java does not generate code for object wrappers. See “Methods generated for object wrappers” on page 4-5 for more information. |
| -incl_files_code           | Generates code for included files.                                                                                                                                                                                                                                                                                                   |
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 page 2–7.

**Arguments**
The supported arguments for the `idl2java` preprocessor are shown in the following table.

| Argument   | Description                                                                 |
|------------|-----------------------------------------------------------------------------|
| -C         | Retains comments from IDL file when the Java code is generated.             |
|            | Otherwise, the comments will not appear in the Java code.                   |
| -H         | Prints the full paths of included files on the standard error output.       |
| -P         | Suppresses the generation of line number information.                       |
| -d         | Turns on pre-processor debugging mode.                                      |
This command creates an IDL file from a Java file (in Java byte code). You can enter one or more Java file names. If you enter more than one filename, make sure you include spaces in between the file names.

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

Syntax
java2idl [options] {filename}

Example
java2idl Account Client Server

Description
Use this command when you want to create an IDL file 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                                          |
|----------|------------------------------------------------------|
| -o filename | Generates output to a specific file.                  |
| -verbose      | Turns verbose mode on.                              |
| -version       | Displays or prints out the version of VisiBroker for Java that you are currently running. |
| -wide           | Maps Java String/char to IDL wstring/wchar.          |

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 page 2–7.

java2iiop

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

For more information about using this command, see “Defining CORBA interfaces with Java” in the VisiBroker for Java Programmer’s Guide.
To use this command, you must have a virtual machine supporting JDK 1.1 or later.

Syntax

java2iiop [options] (filename)

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 Caffeine interfaces.

Options

The following options are available for java2iiop.

| Option        | Description                                                                 |
|---------------|-----------------------------------------------------------------------------|
| -root_dir directory | Places all generated files in the specified directory, instead of in the current directory. |
| -strict       | Generates code that can run on any CORBA-compliant ORB. When -strict is not set, java2iiop does not generate portable code, using Visigenic extensions and optimizations in the code it generates. |
| -smart_stub   | Generates the ability to use smart stubs. It creates additional methods in the interface Helper class. When this option is not set, java2iiop does not generate code for installing or using smart stubs. |
| -no_bind      | Suppresses the generation of bind() methods. |
| -no_comments  | Suppresses comments in generated code. |
| -no_examples  | Suppresses the generation of example code. |
| -no_tie       | Suppresses the generation of tie code. |
| -portable     | Generates portable stubs using DII. |
| -version      | Displays or prints out the version of VisiBroker for Java that you are currently running. |
| -W #          | Setting this option to 0 (zero) suppresses all warnings from the compiler. |
| -wide         | Maps Java String/char to IDL wstring/wchar. |

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 page 2–7.

vbj

This command starts the local Java interpreter.
**Syntax**

```bash
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.
- Can be configured on Windows platforms to launch a specified Java VM, rather than the JavaSoft VM.

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

- `ORBagentAddr` is set to the value of the `OSAGENT_ADDR` environment variable or registry setting. If the `-VBJfindagent 1` option is specified, `vbj` broadcasts to find the setting for `ORBagentAddr`.
- `ORBagentPort` 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 `ORBagentAddr` and `ORBagentPort` to insure the proper operation of VisiBroker.

**Options**

The following options are available for `vbj`.

| Option | Description |
|--------|-------------|
| `-version` | Displays or prints out the version of VisiBroker for Java that you are currently running. |
| `-VBJprop name=value` | Passes the property name and value pair into the java VM as a System Property by adding it as a `-D<name>=<value>` parameter to the executed “java.” |
| `-VBJjavavm java_executable` | Specifies path and flags to java VM (virtual machine). If you do not specify, it defaults to “java.” If this option isn’t specified, you can also set the `VBROKER_JAVAVM` environment variable. |
### Option Description

| Option              | Description                                                                                                                                 |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| -VBjfindagent 1     | Attempts to locate an osagent on the network and pass the agent’s location to the ORB runtime. This option should only be used when running a Java VM which is not capable of locating an Smart Agent via UPD broadcast (for example, JDK 1.0.x). |
| -VBjincppjar1       | Looks for classes in the vbcpp.jar file.                                                                                                                                                             |
| -VBjcppjarloc jarpath/jarfile | Specifies the vbcpp.jar file location. If you don’t specify, the default location is <installation_directory>/lib/vbcpp.jar. If this option isn’t specified, you can also set environment variable VBROKER_CPPJARLOC. |
| -VBjadm1            | Propagates VBROKER_ADM to spawned VM.                                                                                                                                                                |
| -VBjtag tag         | Specifies the tag to use for the parameters being passed to the Java VM. The default tag value is -D. You can also set this option using the VBROKER_TAG environment variable and registry setting. |
| -VBjquoteSpaces 0   | Turns off the automatic placement of quotation marks around arguments that contain spaces.                                                                                                          |
| -VBjclasspath classpath | Specifies an explicit setting for classpath.                                                                                                                                                        |
| -VBjdebug 1         | 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.

In Windows environments, you can also use the VisiBroker vregedit tool to set the following options in the registry:

- VBROKER_ADM
- OSAGENT_PORT
- VBROKER_JAVAVM
- VBROKER_TAG
Chapter 3

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.inprise.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

This chapter includes the following major sections:

- Names page 3-2
- Reserved names page 3-2
- Reserved words page 3-3
- Modules page 3-3
- Basic types page 3-4
- Helper classes page 3-9
- Constants page 3-10
- Constructed types page 3-11
- Interfaces page 3-18
- Mapping for exceptions page 3-22
- User-defined exceptions page 3-23
- System exceptions page 3-24
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 `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 (_) 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.
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 (_) prepended. The reserved keywords in the Java language are as follows:

|abstract| boolean| break| byte|
case | catch | char | class|
const | continue | default | do|
double | else | extends | final|
finally | float | for | goto|
if | implements | import | instanceof|
int | interface | long | native|
new | package | private | protected|
public | return | short | static|
super | switch | synchronized | this|
throw | throws | transient | try|
void | volatile | while|

Note  The word extensible is also a keyword; however, it is a VisiBroker extension to this list.

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 an enumeration 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               |
| long long    | long               |
| unsigned long long | long       |
| 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,

- The 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? |
|-------------------|-----------------------------------|
| long long         | yes                               |
| unsigned long long | yes                              |
| long double       | no¹                               |
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 classes,” of this guide.

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
- 0 for numeric and char types
- null for strings
- null for object references

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

**Table 3.2** Summary of supported IDL extensions (continued)

| Type  | Supported in VisiBroker for Java? |
|-------|-----------------------------------|
| wchar | yes*1                           |
| wstring | yes**                          |
| fixed | no1                             |

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                                      |
|------------|--------------------------------------------------|
| long long  | 64-bit signed 2's complements integers           |
| unsigned long long | 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 classes,” of this guide.

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
- 0 for numeric and char types
- null for strings
- null for object references

To support portable stubs and skeletons, Holder classes for use-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
// Java
package org.omg.CORBA;

final public class ShortHolder {
    public short value;
    public ShortHolder() {}
    public ShortHolder(short initial) {
        value = initial;
    }
}

final public class IntHolder {
    public int value;
    public IntHolder() {}
    public IntHolder(int initial) {
        value = initial;
    }
}

final public class LongHolder {
    public long value;
    public LongHolder() {}
    public LongHolder(long initial) {
        value = initial;
    }
}

final public class ByteHolder {
    public byte value;
    public ByteHolder() {}
    public ByteHolder(byte initial) {
        value = initial;
    }
}

final public class FloatHolder {
    public float value;
    public FloatHolder() {}
    public FloatHolder(float initial) {
        value = initial;
    }
}

final public class DoubleHolder {
    public double value;
    public DoubleHolder() {}
    public DoubleHolder(double initial) {
        value = initial;
    }
}
```
The Holder class for a user-defined type <foo> is shown below.
**Java null**

The Java `null` may only be used to represent null CORBA object references. For example, a zero length string, rather than `null` must be used to represent the empty string. This is also true for arrays. If you attempt to pass a `null` for a structure, it will raise a `NullPointerException`. If passing a zero length string is desired, use the `ORBnullString` option with `ORB.init()`. For more information about the `ORBnullString` option, see Appendix A, “Using command-line options,” in the *VisiBroker for Java Programmer’s Guide*.

---

**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 `char`s mapped from IDL `char`s 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`. 
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.

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.

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.

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

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. In addition, both the helper class associated with an IDL interface and the helper class for a mapped IDL interface have a `narrow` operation defined for them.

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

```java
// generated Java helper
public class <typename>Helper {
    public static void insert(org.omg.CORBA.Any a, <typename> t);
    public static <typename> extract(Any a);
```
Constants

```java
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(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(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 corresponding to the IDL interface.

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

```java
/* From Example.idl: */
module Example
    interface Foo{
        const long aLongerOne = -321;
    }
};
// Generated java
package Example;
public interface Foo {
    public static final int aLongerOne = (int) (-321L);
}
```

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

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

**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

```java
// 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
  private <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

```java
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

```java
// IDL
definition EnumType {a, b, c};

// generated Java
class EnumType {
    public static final int _a = 0;
    public static final EnumType a = new EnumType(_a);

    public static final int _b = 1;
    public static final EnumType b = new EnumType(_b);

    public static final int _c = 2;
    public static final EnumType c = new EnumType(_c);

    public int value() {...}
    public static EnumType from_int(int value) {...};

    // constructor
    private EnumType(int) {...}
};
```

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

```java
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

```java
/* From Example.idl: */
struct StructType {
    long field1;
    string field2;
};
```
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 discriminant 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 discriminant. 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 modifier method, named default() (_default() if name conflict) 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 discriminant. 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: */

```
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) {...}
}

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(int[] initial) {...};
    public void _read(org.omg.CORBA.portable.InputStream i) {

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**

```java
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**

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

final public class larrayHolder
    implements org.omg.CORBA.portable.Streamable {
    public int[] value;
```
IDL interfaces are mapped to public Java interfaces and given the same names. 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 holds 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.

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**

```java
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**

```java
/* From Example.idl: */
module Example {
    interface Foo {
        long method(in long arg) raises(e);
    }
}
```
Passing parameters

IDL in parameters which implement call-by-value semantics, 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, which implement call-by-result and call-by-value/result semantics, 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.
**Server implementation with inheritance**

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

```java
/* 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 {
    int operation(int inArg,
                  IntHolder outArg,
                  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

```java
// 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
Server implementation with delegation

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 skeleton abstract class that implements the Java version of the IDL interface. User defined server classes are then linked to the ORB by extending the skeleton class, as shown in Code sample 3.24.

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

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

// Generated java
package Example;
abstract public class _AccountImplBase extends org.omg.CORBA.Skeleton
  implements Account { ... }

// Linking an implementation to the ORB:
class AccountImpl extends Example._AccountImplBase { ... }
```

Server implementation with delegation

The use of inheritance to implement a server has one drawback: since the server class extends the 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 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 *Operations* interface does not extend *CORBA.Object* and, as a result, its instances cannot be treated directly as ORB objects. Instead, they have 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

```java
/* From Example.idl: */
module Example {
  interface Account {
    float balance();
  }
};
```
Interface scope

// Generated java
package Example;
public interface AccountOperations {
   public float balance();
};

package Example;
// Generated java
public class _tie_Account extends _AccountImplBase {
   private AccountOperations delegate;
   public _tie_Account(AccountOperations d) { ... }
   public _tie_Account(AccountOperations d, String name) { ... }
   public float balance() {
      return delegate.balance();
   }
}

// Linking an implementation to the ORB:
class AccountImpl implements Example.AccountOperations extends Whatever {
   ...
   Example.Account a_server = new Example._tie_Account(new AccountImpl());
   ...
}

Interface scope

Java 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 an underscore 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 \texttt{java.lang.Exception}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{user-defined-exceptions-diagram}
\end{figure}

\textbf{User-defined exceptions}

User-defined exceptions are mapped to final Java classes that extend \texttt{org.omg.CORBA.UserException} and are otherwise mapped just like the IDL \texttt{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.

\textbf{Code sample 3.26  Mapping user-defined exceptions}

\begin{verbatim}
// IDL
module Example {
    exception ex1 (string reason);}

// Generated Java
package Example;
final public class ex1 extends org.omg.CORBA.UserException {
    public String reason; // instance
    public ex1() {...} // default constructor
    public ex1(String r) {...} // constructor
}

final public class ex1Holder
    implements org.omg.CORBA.portable.Streamable {
    public ex1 value;
    public ex1Holder() {}
    public ex1Holder(ex1 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() {...}
}
\end{verbatim}
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 raised.

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 “Insertion methods” on page 6-4 and “Extraction methods” on page 6-3.

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.
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 3.27  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 {...}
```

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 {
    ...
}
```
This chapter describes the classes that can be generated by the \texttt{idl2java} compiler. It contains the following sections:

| Section                  | Page |
|--------------------------|------|
| Overview                 | 4-1  |
| <interface\_name>Helper  | 4-3  |
| <interface\_name>Holder  | 4-6  |
| <interface\_name>Operations | 4-7  |
| _st_<interface\_name>    | 4-8  |
| _<interface\_name>ImplBase | 4-8  |
| _sk_<interface\_name>    | 4-8  |
| _tie_<interface\_name>   | 4-8  |

### Overview

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

- public class \texttt{MyType}
- public class \texttt{MyTypeHelper}
- public class \texttt{MyTypeHolder}
- public class \_st\_\texttt{MyType}
- public class \_\texttt{MyTypeImplBase}
- public class \_\texttt{MyType}
- public class \_\texttt{example_MyType}
**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 structures, enumerations, and unions, 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.

**Stub class**

The stub class provides stub implementation for `<interface_name>` which the client calls.

**ImplBase class**

The ImplBase class is used to derive an implementation class for `<interface_name>`.

**Skeleton class**

The skeleton class has been deprecated. ImplBase class is currently used instead. A skeleton class is used to derive an implementation class for `<interface_name>`.

**Tie class**

This class inherits from the skeleton class and delegates every call to the real implementation class. For an example of the tie mechanism, see Chapter 10, “Tie mechanism,” in the *VisiBroker for Java Programmer’s Guide*.

**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. For an example of the tie mechanism, see Chapter 10, “Tie mechanism,” in the *VisiBroker for Java Programmer’s Guide*. 
Examples class

This class provides code you can fill in to implement the object.

<interface_name>Helper

abstract public class <interface_name>Helper

A Helper class is provided for most classes in the org.omg.COBRA 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, <interface_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 <interface_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 6-28.
Methods generated for interfaces

public static void write(org.omg.CORBA.portable.OutputStream -output, <interface_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 “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 java.lang.Class getStubClass()

This method tells you which stub class the client program is currently using—the smart stub or the default stub.
Methods generated for object wrappers

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 <interface_name>. |

public static void resetStubClass()

This method resets the stub class setting so that the client program uses the default stub.

public static void setStubClass(java.lang.Class c)

This method sets the stub class setting so that the client program uses the smart stub.

| Parameter | Description |
|-----------|-------------|
| c         | The smart stub class that you want to use with your client program. |

Methods generated for object wrappers

The following methods are generated for helper classes when you invoke the idl2java command with the -obj_wrapper option, as described on page 2-3. 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 will be invoked 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, returned by the invoking the ORB.init 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 will be invoked in the order in which they were registered.
<interface_name>Holder

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, returned by the invoking the ORB.init 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, returned 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 ORB the server wishes to use, returned by the invoking the ORB.init method. |

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
Member data

- public class AnyHolder
- public class TypeCodeHolder
- public class PrincipalHolder

The Holder class for a user defined type `<interface_name>` follows.

**Code sample 4.1  Holder class**

```java
// Java
final public class <interface_name>Holder
    implements org.omg.CORBA.portable.Streamable {
    public <interface_name> value;
    public <interface_name>Holder();
    public <interface_name>Holder(<interface_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 `<interface_name>` value**

This value represents the type contained by this object.

**Methods**

**public `<interface_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 `<interface_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>Operations**

abstract public interface `<interface_name>Operations`

An Operations class is generated by the idl2java compiler and contains the interface definitions of the methods for `<interface_name>`. 
_st_<interface_name>

abstract public class _st_<interface_name>

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>ImplBase

abstract public class _<interface_name>ImplBase

An implementation base class containing skeleton code is generated by the idl2java compiler and is used to derive an implementation class for <interface_name>.

_sk_<interface_name>

abstract public class _sk_<interface_name>

This class has been deprecated. Use of the implementation base class _<interface_name>ImplBase is recommended.

_tie_<interface_name>

abstract public class _tie_<interface_name>

A tie class is generated by the idl2java compiler for creating a delegator class for <interface_name>.
Chapter 5

Core interfaces and classes

This chapter describes the core interfaces and classes in the org.omg.CORBA package. It contains the following sections.

- BindOptions page 5-1
- BOA page 5-2
- CompletionStatus page 5-5
- Context page 5-6
- InvalidName page 5-8
- Object page 5-8
- ORB page 5-13
- Principal page 5-25

BindOptions

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 will 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 will be made to
Constructors

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 classes,” for more information on these classes and the methods they offer.

IDL definition

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

Constructors

```java
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 |
|------------|-------------|
| defer_bind | If set to true, connection establishment is delayed until the first operation request is invoked on the object. The default value is false. |
| enable_rebind | If set to true and the connection is broken to the server, the client attempts to bind to another server. The default value is true. |

BOA

```java
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.

For more information on the Object interface, see page 5-8.

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

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

```java
... 
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
org.omg.CORBA.BOA boa = orb.BOA_init();
```
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);
};
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 invoked 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 |
|-----------|-------------|
| object    | 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 |
|-----------|-------------|
| object    | The server’s implementation object to be activated. |
| service_name | The service name associated with the object implementation to be activated. |
| ref_data  | The reference data associated with the object to be activated. |

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

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

Methods

For more information about these methods, see “Enum” on page 3-12 of Chapter 3, “IDL to Java mapping.”

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)

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 on page 5-22.

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
   );
};

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 6-2.

| 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. The `NVList` is described on page 6-20.

| 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 on page 5-24.

**Helper** and **Holder** versions of this class are also provided. See Chapter 4, “Generated 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.

A **Holder** version of this class is also provided, described in Chapter 4, “Generated classes.”

**IDL definition**

```idl
interface Object {
    boolean _is_a(
        in CORBA::RepositoryId repId);
    boolean _is_equivalent(
        in CORBA::Object other_object);
    boolean _non_existent();
    unsigned long_hash(
        in unsigned long maximum);
    CORBA::Object _duplicate();
    void _release();
    CORBA::Request _request(
        in CORBA::Identifier operation);
}
```
public org.omg.CORBA.BindOptions _bind_options()

This method returns the bind options associated with this object.

public void _bind_options(org.omg.CORBA.BindOptions _bind_options)

This method is used by client applications and sets the bind options associated with this object.

Note Bind options cannot be set on an implementation object. They can only be set on a client’s proxy object.

| Parameter    | Description                          |
|--------------|--------------------------------------|
| _bind_options | The bind options to be set for this 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 org.omg.CORBA.Object _clone()

This method creates a copy of this Object, which then has its own TCP connection to the server. If the object implementation is local, the pointer of the implementation object is returned.

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 on page 5-22.

| 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 on page 5-22.

| 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. |
| exceptions| A list of Typecode objects representing the exceptions that this request might raise. |
| contexts  | A list of Context objects. The list of Context objects supports type checking on context names. |

public org.omg.CORBA.InterfaceDef _get_interface()

This method returns the interface definition of this object. An interface repository must be available if this method is to be used. For information about starting an interface repository, see the VisiBroker for Java Installation and Administration Guide.
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 |
|-----------|-------------|
| maximum   | 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 |
|-----------|-------------|
| repId     | 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.

```idl
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_bound()

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

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 |
|-----------|-------------|
| other_object | A reference to the object to be compared with this object. |
Methods

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 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 java.lang.String _object_name()

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

public Principal _principal()

This method is used by object implementations to obtain the principal associated with a request on this object. The principal can be associated with a client’s proxy object or with the target of an active operation request. A null value is returned if this object is not currently processing an operation request.

Client applications and object implementations may set the principal using the _principal method which follows.

public void _principal(org.omg.CORBA.Principal principal)

This method sets the principal associated with this object. A principal can either be associated with a proxy object or with the target of an active invocation. The principal is stored by reference—be careful when modifying the value.

| Parameter   | Description                  |
|-------------|------------------------------|
| principal   | A principal to set for this object. |

public java.lang.String _repository_id()

This method returns the repository identifier of the object implementation’s most derived interface.
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 6-23 for more information initializing and sending dynamic invocation requests.

| Parameter | Description |
|-----------|-------------|
| operation | The name of the operation to be invoked. |

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 on page 5-24, to be invoked on the server-side to resolve the specified service. An object reference is returned which your client can narrow to the appropriate server type.

This method is typically used by client applications that wish to use the ORB Management interfaces, described in Chapter 15 and in the VisiBroker for Java Programmer’s Guide, to manage a server’s attributes.

| Parameter | Description |
|-----------|-------------|
| id        | The name of the interface to be resolved on the server-side. |

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, along with the Basic Object Adaptor, provides a variety of methods used by both clients and servers.

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

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

For more information on the Basic Object Adaptor, see “BOA” on page 5-2.
IDL definition

pseudo interface ORB {
    exception InvalidName {};
    typedef string ObjectId;
    typedef sequence<ObjectId> ObjectIdList;
    ObjectIdList list_initial_services();
    Object resolve_initial_references(in ObjectId object_name)
        raises(InvalidName);
    string object_to_string(in Object obj);
    Object string_to_object(in string str);
    NVList create_list(in long count);
    NVList create_operation_list(in OperationDef oper);
    NamedValue create_named_value(in String name, in Any value,
        in Flags flags);
    ExceptionList create_exception_list();
    ContextList create_context_list();
    Context get_default_context();
    Environment create_environment();
    void send_multiple_requests_oneway(in RequestSeq req);
    void send_multiple_requests_deferred(in RequestSeq req);
    boolean poll_next_response();
    Request get_next_response();

    // typecode creation
    TypeCode create_struct_tc (in RepositoryId id,
        in Identifier name,
        in StructMemberSeq members);
    TypeCode create_union_tc (in RepositoryId id,
        in Identifier name,
        in TypeCode discriminator_type,
        in UnionMemberSeq members);
    TypeCode create_enum_tc (in RepositoryId id,
        in Identifier name,
        in EnumMemberSeq members);
    TypeCode create_alias_tc (in RepositoryId id,
        in Identifier name,
        in TypeCode original_type);
    TypeCode create_exception_tc (in RepositoryId id,
        in Identifier name,
        in StructMemberSeq members);
    TypeCode create_interface_tc (in RepositoryId id,
        in Identifier name);
    TypeCode create_string_tc (in unsigned long bound);
    TypeCode create_sequence_tc (in unsigned long bound,
        in TypeCode element_type);
    TypeCode create_recursive_sequence_tc(in unsigned long bound,
        in unsigned long offset);
    TypeCode create_array_tc (in unsigned long length,
        in TypeCode element_type);

    // Additional operations for Java mapping
    TypeCode get_primitive_tc(in TCKind tcKind);
    Any create_any();
    OutputStream create_output_stream();
}
void connect(Object obj);
void disconnect(Object obj);
Current get_current();
// additional static methods for ORB initialization go here

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.

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.

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
  For a listing of the properties that can be set, see Appendix A, “Using command-line options,” in the Visibroker for C++ Reference that lists the properties you can set with BOA_init and ORB_init.
  For more information, see “BOA” on page 5-2.
This method can be called repeatedly and at anytime 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.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                                      |
|-------------|--------------------------------------------------|
| repository_id | The repository identifier that specifies the type in IDL. |
| type_name    | The unscoped type name of the type.              |
| original_type | The aliased type.                               |

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

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

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 org.omg.CORBA.DynAny create_basic_dyn_any(org.omg.CORBA.TypeCode type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynAny object for holding the specified basic data type, such as short, int, or string.

Note: The type of a DynAny cannot be changed during the lifetime of the object. 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                                      |
|-------------|--------------------------------------------------|
| type        | The TypeCode of the value this object is to hold. See “TypeCode” on page 6-29 for a list of valid values. |

public abstract ContextList create_context_list()

This method creates and returns an empty ContextList.
Methods

Core interfaces and classes

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 Any object used to initialize this object. |

public org.omg.CORBA.DynArray `create_dyn_array(org.omg.CORBA.TypeCode type)`

throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynArray object for holding an array of data types with the specified TypeCode.

**Note**
The type of a DynAny cannot be changed during the lifetime of the object. 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 “DynArray” on page 6-10 for complete details.

| Parameter | Description |
|-----------|-------------|
| type      | The TypeCode of the values this object is to hold. See “TypeCode” on page 6-29 for a list of valid values. |

public org.omg.CORBA.DynEnum `create_dyn_enum(org.omg.CORBA.TypeCode type)`

throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynEnum object for the specified data type. DynEnum objects are used to dynamically create or interpret enumeration values that were not defined at runtime.

A org.omg.CORBA.DynAnyPackage.InconsistentTypeCode exception will be raised if the type is invalid.

**Note**
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. See “DynEnum” on page 6-10 for complete details.

| Parameter | Description |
|-----------|-------------|
| type      | The TypeCode of the value this object is to hold should have a TCKind of _tk_enum. |

public org.omg.CORBA.DynSequence `create_dyn_sequence(org.omg.CORBA.TypeCode type)`

throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynSequence object for holding an sequence of values with the specified type.

**Note**
The type of a DynSequence cannot be changed during the lifetime of the object. DynSequence object cannot be used as parameters on operation requests or DII
requests nor can they be externalized using the ORB.object_to_string method. See “DynSequence” on page 6-11 for complete details.

| Parameter   | Description                                                                 |
|-------------|-----------------------------------------------------------------------------|
| type        | The TypeCode of the values this object is to hold. See “TypeCode” on page 6-29 for a list of valid values. |

public org.omg.CORBA.DynStruct create_dyn_struct(org.omg.CORBA.TypeCode type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynStruct object with the specified type. DynStruct objects are used to dynamically create or interpret structures whose type was not necessarily defined at runtime.

A DynAnyPackage.InconsistentTypeCode exception will be raised if the type is invalid.

Note DynStruct object cannot be used as parameters on operation requests or DII requests nor can they be externalized using the ORB.object_to_string method. See “DynStruct” on page 6-12 for complete details.

| Parameter   | Description                                                                 |
|-------------|-----------------------------------------------------------------------------|
| type        | The TypeCode of the structure this object is to hold. See “TypeCode” on page 6-29 for a list of valid values. |

public org.omg.CORBA.DynUnion create_dyn_union(org.omg.CORBA.TypeCode type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode

Creates a DynUnion object with the specified type. DynUnion objects are used to dynamically create or interpret unions that were not necessarily defined at runtime.

Note DynUnion object cannot be used as parameters on operation requests or DII requests nor can they be externalized using the ORB.object_to_string method. See “DynUnion” on page 6-13 for complete details.

| Parameter   | Description                                                                 |
|-------------|-----------------------------------------------------------------------------|
| type        | The TypeCode of the value this object is to hold. See “TypeCode” on page 6-29 for a list of valid values. |

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.                      |
Methods

Core interfaces and classes

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

This method creates and returns an empty Environment.

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

This method creates and returns a TypeCode describing an IDL estruct.

| Parameter     | Description                                                                 |
|---------------|-----------------------------------------------------------------------------|
| repository_id | The repository identifier that specifies the type in IDL.                    |
| type_name     | The unscoped type name of the type.                                         |
| base          | The estruct from which the current estruct inherits. Use NULL to indicate that this is the root structure. |
| members       | An array of structures defining the members of the type.                    |

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.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.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                                                                 |
|---------------|-----------------------------------------------------------------------------|
| repository_id | The repository identifier that specifies the type in IDL.                    |
| type_name     | 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 the NVList, see page 6-20.

| Parameter | Description |
|-----------|-------------|
| length    | 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 |
|-----------|-------------|
| name      | The name for the NamedValue. |
| value     | The value for the NamedValue. |
| flags     | The flags for the NamedValue: 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 |
|-----------|-------------|
| operationDef | The operation description that must be specified. |

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.TypeCode create_recursive_sequence_tc(int length, int offset)

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

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

Core interfaces and classes

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                                         |
|-------------|-----------------------------------------------------|
| length      | The length of the sequence to be created. A length of zero indicates an unbounded sequence is desired. |
| element_type| 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                                         |
|-----------|-----------------------------------------------------|
| length    | 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_wstring_tc(int length)

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

| Parameter | Description                                         |
|-----------|-----------------------------------------------------|
| length    | 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                                         |
|-------------|-----------------------------------------------------|
| 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_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                                         |
|-------------|-----------------------------------------------------|
| repository_id| The repository identifier that specifies the type in IDL. |
| type_name   | The unscoped type name of the type.                  |
abstract public org.omg.CORBA.BindOptions default_bind_options()

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

abstract public void default_bind_options(org.omg.CORBA.BindOptions options)

This method sets the global, default bind options.

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

abstract public Principal default_principal()

This method returns the global, default Principal.

abstract public void default_principal(Principal principal)

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

Caution The global, default Principal is stored by reference. Use care when modifying the value.

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 in available. The completed Request is returned. See also, send_multiple_requests_deferred on page 5–25.
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 |
|-----------|-------------|
| kind      | The kind of type code kind, as defined in TCKind. |

public static ORB init()

This method returns the ORB singleton.

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 |
|-----------|-------------|
| args      | The command-line arguments to pass to the program. |
| props     | 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 |
|-----------|-------------|
| app       | The applet to associate with this instance of the ORB. |
| props     | 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 |
|-----------|-------------|
| obj       | The object reference to be converted. |
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 on page 5–25.

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                                                                 |
|---------------|-----------------------------------------------------------------------------|
| identifier    | 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 Helper.bind 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 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                  | Returned type                                                                 |
|--------------------------------------------------|-------------------------------------------------------------------------------|
| InterfaceRepository                               | org.omg.CORBA.Repository                                                      |
| HandlerRegistry                                  | com.visigenic.vbroker.interceptor.HandlerRegistry                           |
| ChainBindInterceptor                             | com.visigenic.vbroker.interceptor.ChainBindInterceptor                     |
| ChainClientInterceptorFactory                    | com.visigenic.vbroker.interceptor.ChainClientInterceptorFactory            |
| ChainServerInterceptorFactory                    | com.visigenic.vbroker.interceptor.ChainServerInterceptorFactory            |
| ChainClientUntypedObjectWrapperFactory           | com.visigenic.vbroker.interceptor.ChainClientUntypedObjectWrapperFactory   |
| ChainServerUntypedObjectWrapperFactory           | com.visigenic.vbroker.interceptor.ChainServerUntypedObjectWrapperFactory   |
| URLNamingResolver                                | com.visigenic.vbroker.URLNaming.resolver                                   |

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. |

Principal

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 on page 5–22. The principal may also be retrieved using the boa.get_principal method described on page 5–4.

IDL definition

pseudo interface Principal {
  attribute sequence<octet> name;
}

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.
Chapter 6

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 page 6-2
ARG_IN page 6-4
ARG_INOUT page 6-5
ARG_OUT page 6-5
ContextList page 6-5
DynAny page 6-6
DynArray page 6-10
DynEnum page 6-10
DynSequence page 6-11
DynStruct page 6-12
DynUnion page 6-13
Environment page 6-16
InputStream page 6-17
Invalid page 6-18
InvalidSeq page 6-19
NamedValue page 6-19
NameValuePair page 6-19
NVList page 6-20
public abstract class Any

The Any class 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. By default, an Any contains a NULL until it is initialized.

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 class is also provided, described in Chapter 4, “Generated classes.”

Methods

public abstract InputStream create_input_stream()

This method creates an input stream containing the Any’s value.

public abstract OutputStream create_output_stream()

This method creates an empty output stream.

abstract public boolean equal(Any rhs)

This method returns true if the value contained by the Any is the same as the value contained by the argument Any.rhs. Otherwise, false is returned.

| Parameter | Description |
|-----------|-------------|
| rhs       | The Any whose value is compared with the value of this Any. |
Extraction methods

public abstract void read_value(InputStream input, 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 |
|-----------|-------------|
| input     | A GIOP input stream from which the specified type’s value will be read. |
| type      | The type to be read from the input stream. See “TypeCode” on page 6-29 for the possible values for this parameter. |

abstract public TypeCode type()

This method returns the TypeCode representing the type contained in this Any.

abstract public void type(org.omg.CORBA.TypeCode type)

This method sets the TypeCode representing the type contained in this Any.

| Parameter | Description |
|-----------|-------------|
| type      | The type to be set for this Any object. See “TypeCode” on page 6-29 for the possible values for this parameter. |

public abstract void write_value(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 |
|-----------|-------------|
| output    | A GIOP output stream into which the specified Any’s value is written. |

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 class

abstract public short extract_short()
abstract public int extract_long()
abstract public long extract_longlong()
abstract public short extract_ushort()
abstract public int extract_ulong()
abstract public long extract_ulonglong()
abstract public float extract_float()
abstract public double extract_double()
abstract public boolean extract_boolean()
abstract public char extract_char()
abstract public char extract_wchar()
**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 class**

```java
abstract public void insert_short(short s);
abstract public void insert_long(int i);
abstract public void insert_longlong(long l);
abstract public void insert_ushort(short s);
abstract public void insert_ulong(int i);
abstract public void insert_ulonglong(long l);
abstract public void insert_float(float f);
abstract public void insert_double(double d);
abstract public void insert_boolean(boolean b);
abstract public void insert_char(char c);
abstract public void insert_wchar(char c);
abstract public void insert_octet(byte b);
abstract public void insert_any(org.omg.CORBA.Any a);
abstract public void insert_Object(org.omg.CORBA.Object o);
abstract public void insert_Object(org.omg.CORBA.Object o,
       org.omg.CORBA.TypeCode t);
abstract public void insert_string(String s);
abstract public void insert_wstring(String s);
abstract public void insert_TypeCode(org.omg.CORBA.TypeCode t);
abstract public void insert_Principal(org.omg.CORBA.Principal p);
abstract public void insert_Streamable(org.omg.CORBA.portable.Streamable s);
```

---

**ARG_IN**

public final class **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” on page 6-23 and “NVList” on page 6-20.
Variables

final public static int value = (int) 1;

ARG_INOUT

public final class 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” on page 6-23 and “NVList” on page 6-20.

Variables

final public static int value = (int) 3;

ARG_OUT

public final class 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” on page 6-23 and “NVList” on page 6-20.

Variables

final public static int value = (int) 2;

ContextList

public class 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;
}
Methods

```java
public abstract void add(String ctx);
string item(in unsigned long index)
  raises(CORBA::Bounds);
void remove(in unsigned long index)
  raises(CORBA::Bounds);

Methods

public abstract void add(String ctx)
This method adds a string to the context list.

| Parameter | Description |
|-----------|-------------|
| ctx       | The name of the context list. |

public abstract int count()
This method returns the number of elements in the context list.

public abstract 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 abstract 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.Object
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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes 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.

DynAny objects for basic types are created using the ORB.create_basic_dyn_any method, described in Chapter 5. A DynAny object may also be created and initialized from an Any object using the ORB.create_dyn_any method, also described in Chapter 5.

The following interfaces are derived from DynAny and provide support for constructed types that are dynamically managed.

| Constructed type | Interface |
|------------------|-----------|
| Array            | “DynArray” on page 6-10 |
| Enumeration      | “DynEnum” on page 6-10 |
| Sequence         | “DynSequence” on page 6-11 |
| Structure        | “DynStruct” on page 6-12 |
| Union            | “DynUnion” on page 6-13 |

**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.

**Methods**

public void assign(org.omg.CORBA.DynAny dyn_any) throws org.omg.CORBA.DynAnyPackage.Invalid

    Initializes the current component in this object from the specified DynAny.

    An org.omg.CORBA.DynAnyPackage.Invalid exception is raised if the type contained in the Any does not match the type contained by this object.

public org.omg.CORBA.DynAny copy()

    Returns a copy of this object.

public org.omg.CORBA.DynAny current_component()

    Returns the current component in this object.

public void destroy()

    Destroys this object.

public void from_any(org.omg.CORBA.Any value) throws org.omg.CORBA.DynAnyPackage.Invalid

    Initializes the current component of this object from the specified Any object.
Extraction methods

An org.omg.CORBA.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() throws org.omg.CORBA.DynAnyPackage.Invalid

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.

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.CORBA.DynAnyPackage.Invalid exception is raised if the value contained in this DynAny does not match the expected return type for the extraction method used.
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.CORBA.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 class

```java
public void insert_any(org.omg.CORBA.Any 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_longdouble(long 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 wchar(char value)
public void insert wstring(java.lang.String value)
```
public interface DynArray extends org.omg.CORBA.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 DynAny, may be used to navigate through the components.

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

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.CORBA.DynAnyPackage.InvalidSeq

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 Any objects whose values will be set in this DynArray. |

DynEnum

public interface DynEnum extends org.omg.CORBA.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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes 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.

### Methods

```java
public java.lang.String value_as_string();

Returns the `DynEnum` object’s value as a string.

public void value_as_string(java.lang.String value_as_string)

Sets the value contained in this `DynEnum` from the specified string.

| Parameter          | Description            |
|--------------------|------------------------|
| value_as_string    | A string that will be used to set the value in this DynEnum. |

public int value_as_ulong()

Returns an `int` containing the `DynEnum` object’s value.

public void value_as_ulong(int value_as_ulong)

Sets the value contained in this `DynEnum` from the specified `int`.

| Parameter          | Description            |
|--------------------|------------------------|
| value_as_ulong     | An integer that will be used to set the value in this DynEnum. |
```

### DynSequence

The `DynSequence` 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.
Important usage restrictions

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated classes,” for more information on these classes 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.toAny method to convert a DynSequence object to a sequence of Any objects, which can be used as a parameter.

Methods

```java
public org.omg.CORBA.Any[] get_elements()

Returns a sequence of Any objects containing the values stored in this object.

public int length()

Returns the number of components contained in this DynSequence.

public void length(int length)

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.CORBA.DynAnyPackage.InvalidSeq

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

```java
public interface DynStruct extends org.omg.CORBA.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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes 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.

**Methods**

```java
public java.lang.String current_member_name()

  Returns a string containing the member name of the current component.
```

```java
public org.omg.CORBA.TCKind current_member_kind()

  Returns the TypeCode associated with the current component.
```

```java
public org.omg.CORBA.NameValuePair[] get_members()

  Returns the members of the structure as an array of NameValuePair objects.
```

```java
public void set_members(org.omg.CORBA.NameValuePair[] value) throws org.omg.CORBA.DynAnyPackage.InvalidSeq

  Sets the structure members from the array of NameValuePair objects.

  An org.omg.CORBA.DynAnyPackage.InvalidSeq 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**

```java
public interface DynUnion extends org.omg.CORBA.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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes 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 objects, which can be used as a parameter.

Methods

public org.omg.CORBA.DynAny discriminator()

    Returns a DynAny object containing the discriminator for the union.

public org.omg.CORBA.TCKind discriminator_kind()

    Returns the type code of the discriminator for the union.

public org.omg.CORBA.DynAny member()

    Returns a DynAny object for the current component which represents a member in the union.

public org.omg.CORBA.TCKind member_kind()

    Returns the type code for the current component, which represents a member in the union.

public java.lang.String member_name()

    Returns the member name of the current component.

public void member_name(java.lang.String member_name)

    Sets the member name of the current component to the specified name.

| Parameter    | Description                                      |
|--------------|--------------------------------------------------|
| member_name  | The member name to set for the current component.|

public boolean set_as_default()

    Returns true if the current component is the default member.

public void set_as_default(boolean set_as_default)

    Enables or disables the current component as the default member, based on the value of the set_as_default parameter.

| Parameter    | Description                                      |
|--------------|--------------------------------------------------|
| set_as_default | If set to true, the current component will become the default member. |
DynamicImplementation

public abstract class DynamicImplementation extends org.omg.CORBA.portable.ObjectImpl

The DynamicImplementation is an abstract class 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 6-27.

For more information about using dynamic skeletons, see Chapter 9, “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                                           |
|-----------------|-------------------------------------------------------|
| object_name     | The name of the instance. If null, the instance is transient (anonymous). |
| repository_id   | The interface’s repository identifier.                |

protected DynamicImplementation(String object_name, String[] repository_ids)

| Parameter       | Description                                                                 |
|-----------------|-----------------------------------------------------------------------------|
| object_name     | The name of the instance. If null, the instance is transient (anonymous).    |
| repository_id   | An array of repository identifiers, one for each interface. The most derived interface should be element zero in the array. |

Methods

abstract public void invoke(org.omg.CORBA.ServerRequest request)

This method provides the functionality of the server.

| Parameter | Description                                      |
|-----------|--------------------------------------------------|
| request   | A description of the request which the server is to perform. |
Environment

public abstract class 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.

Methods

public abstract 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 abstract 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 abstract 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 class 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);
};
Methods

public abstract 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 abstract int count()

This method returns the number of items in the exception list.

public abstract 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 abstract 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.”

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 abstract boolean read_boolean();
    public abstract char read_char();
    public abstract char read_wchar();
    public abstract byte read_octet();
public abstract short read_short();
public abstract short read_ushort();
public abstract int read_long();
public abstract int read_ulong();
public abstract long read_longlong();
public abstract long read_ulonglong();
public abstract float read_float();
public abstract double read_double();
public abstract String read_string();
public abstract String read_wstring();
public abstract void read_boolean_array(boolean[] value, int offset, int length);
public abstract void read_char_array(char[] value, int offset, int length);
public abstract void read_wchar_array(char[] value, int offset, int length);
public abstract void read_octet_array(byte[] value, int offset, int length);
public abstract void read_short_array(short[] value, int offset, int length);
public abstract void read_ushort_array(short[] value, int offset, int length);
public abstract void read_long_array(int[] value, int offset, int length);
public abstract void read_ulong_array(int[] value, int offset, int length);
public abstract void read_longlong_array(long[] value, int offset, int length);
public abstract void read_ulonglong_array(long[] value, int offset, int length);
public abstract void read_float_array(float[] value, int offset, int length);
public abstract void read_double_array(double[] value, int offset, int length);
public abstract Object read_estruct(String expected_type);
public abstract org.omg.CORBA.Object read_Object();
public abstract org.omg.CORBA.TypeCode read_TypeCode();
public abstract org.omg.CORBA.Any read_any();
public abstract org.omg.CORBA.Principal read_Principal();

Invalid

public class Invalid extends org.omg.CORBA.UserException

    An object of this class is thrown if you attempt to assign an Any object to a DynAny object with an incompatible type.
InvalidSeq

public class InvalidSeq extends org.omg.CORBA.UserException

An object of this class 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 that the DynSequence.

NamedValue

public abstract class 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 6-20.

IDL definition

interface NamedValue {
    readonly attribute CORBA::Identifier name;
    readonly attribute any value;
    readonly attribute CORBA::Flags flags;
};

Methods

public abstract 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 abstract String name()

This method returns the name of this NamedValue. If no name has been set, NULL is returned.

public abstract 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 class NameValuePair

This class is used to represent a structure member contained in a DynStruct object.

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.

Constructors

public NameValuePair()

Creates an empty NameValuePair.

public NameValuePair(java.lang.String id, org.omg.CORBA.Any value)

Creates a 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);
}
Methods

```java
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);
```

### Methods

**public abstract 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 abstract 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 abstract 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 NamedValue to be added. |
| value          | The value of the NamedValue, represented as an Any. The Any interface is described on page 6-2. |
| flags          | The mode of the parameter to be added. |

**public abstract int count()**

This method returns the number of NamedValue items in this NVList.
OutputStream

public abstract 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 NamedValue 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 NamedValue 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.

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

```java
public abstract InputStream create_input_stream();
public abstract void write_boolean(boolean value);
public abstract void write_char(char value);
public abstract void write_wchar(char value);
public abstract void write_octet(byte value);
public abstract void write_short(short value);
public abstract void write_ushort(short value);
public abstract void write_long(int value);
public abstract void write_ulong(int value);
public abstract void write_ulonglong(long value);
public abstract void write_float(float value);
public abstract void write_double(double value);
public abstract void write_string(String value);
public abstract void write_wstring(String value);
public abstract void write_boolean_array(boolean[] value, int offset, int length);
```
public abstract void write_char_array(char[] value, int offset, int length);
public abstract void write_wchar_array(char[] value, int offset, int length);
public abstract void write_octet_array(byte[] value, int offset, int length);
public abstract void write_short_array(short[] value, int offset, int length);
public abstract void write_ushort_array(short[] value, int offset, int length);
public abstract void write_long_array(int[] value, int offset, int length);
public abstract void write_ulong_array(int[] value, int offset, int length);
public abstract void write_longlong_array(long[] value, int offset, int length);
public abstract void write_ulonglong_array(long[] value, int offset, int length);
public abstract void write_float_array(float[] value, int offset, int length);
public abstract 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 abstract void write_Object(org.omg.CORBA.Object value);
public abstract void write_TypeCode(org.omg.CORBA.TypeCode value);
public abstract void write_any (org.omg.CORBA.Any value);
public abstract 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 on page 6-16, containing exceptions
- Context, described 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();
};

Methods

public abstract Any add_in_arg()

    Adds an IN argument to the request.

public abstract Any add_inout_arg()

    Adds an INOUT argument to the request.

public abstract 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 abstract 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 abstract 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 abstract Any **add_out_arg**()  

 Adds an OUT argument to the request.

public abstract 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 abstract String[] **contexts**()  

 This method returns the context list. The list will be empty if the operation does not specify any contexts.

public abstract 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 abstract 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 abstract 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 6-16.

public abstract 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 abstract 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.
Methods

public abstract 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 abstract String operation()

This method returns the name of the operation, or method name, associated with this request.

public abstract boolean poll_response()

This method returns true if a response for a 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 abstract 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 abstract Any return_value()

This method gets the return value of the operation as an Any.

public abstract 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 on page 6–26.

public abstract 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 abstract 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 abstract 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.
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 page 6-15.

For more information about using dynamic skeletons, see Chapter 9, “Dynamic Skeleton interface,” in the *VisiBroker for Java Programmer’s Guide*.

**IDL definition**

```idl
define 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);
}
```

**Methods**

```java
abstract public org.omg.CORBA.Context ctx()

This method returns the current `Context` associated with this `ServerRequest`.

abstract 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 |
|-----------|-------------|
| `except`  | The Exception that was raised. |

abstract public String op_name()

This method returns the operation name associated with this request.

abstract 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 9, “Dynamic Skeleton interface,” in the *VisiBroker for Java Programmer’s Guide*.

| Parameter | Description |
|-----------|-------------|
| `params`  | The NVList where the parameters are to be stored. |
**TCKind**

abstract 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 class **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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.

**IDL definition**

```java
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
};
```

**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 “Enum” on page 3-12.

| Parameter | Description |
|-----------|-------------|
| value     | The enum value. |
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 on page 6-2. 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 class, also described on Chapter 5.

A Holder version of this class is also provided, described on Chapter 4, “Generated classes.”

IDL definition

```idl
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);
};
```

Methods

```java
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 TKind kind()

This method returns the kind of type associated with this type code. Type codes kind constants are defined by TKind, described on page 6-28.

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.
Methods

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
UnknownUserException

- tk_union
- tk_enum
- tk_alias
- tk_except

A BAD_PARAM exception will be raised if the type code is not one of these types.

public class 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.
This chapter describes the interfaces and classes in the `org.omg.CORBA` package that are used with the interface repository.

- **AliasDef**  
  page 7-2
- **ArrayDef**  
  page 7-3
- **AttributeDef**  
  page 7-4
- **AttributeDescription**  
  page 7-4
- **AttributeMode**  
  page 7-5
- **ConstantDef**  
  page 7-6
- **ConstantDescription**  
  page 7-7
- **Contained**  
  page 7-8
- **ContainedPackage.Description**  
  page 7-10
- **Container**  
  page 7-10
- **ContainerPackage.Description**  
  page 7-10
- **DefinitionKind**  
  page 7-16
- **EnumDef**  
  page 7-18
- **EstructDef**  
  page 7-18
- **ExceptionDef**  
  page 7-19
- **ExceptionDescription**  
  page 7-20
- **IDLType**  
  page 7-21
- **InterfaceDef**  
  page 7-22
- **InterfaceDescription**  
  page 7-24
public interface AliasDef extends org.omg.CORBA.TypedefDef

This interface is used to represent a typedef that is stored in the Interface Repository. This interface provides methods for setting and obtaining the IDL type of the original typedef.

For more information on the TypedefDef interface, see page 7–38.

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

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 classes,” for more information on these classes and the methods they offer.

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 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       |
|---------------|-------------------|
| original_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.Contained**

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 classes,” for more information on these classes and the methods they offer.

---

**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               |
|-----------|---------------------------|
| type_def  | The IDLType of this object. |

**public org.omg.CORBA.IDLType type_def()**

This method returns the attribute’s IDLType.

---

**AttributeDescription**

---

**public class AttributeDescription**

The AttributeDescription class describes an attribute that is stored in the interface repository.

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

Table 7.1  AttributeDescription globals and constants

| Constant     | Represents                                         |
|--------------|----------------------------------------------------|
| name         | The name of the attribute.                        |
| id           | The repository id of the attribute.               |
| defined_in   | The name of the module or interface in which this attribute is defined. |
| version      | The attribute’s version.                         |
| type         | The attribute’s IDL type.                        |
| mode         | The mode of the attribute.                       |

Methods

public AttributeDescription()

This method is the default constructor for an AttributeDescription.

public AttributeDescription(String name, String id, String defined_in, org.omg.CORBA.VersionSpec version, org.omg.CORBA.TypeCode type, org.omg.CORBA.AttributeMode mode)

This method constructs an AttributeDescription, using the supplied parameters.

| Parameter     | Description                                                                 |
|---------------|-----------------------------------------------------------------------------|
| name          | The name of this attribute.                                                 |
| id            | The repository id for this attribute.                                       |
| defined_in    | The module or interface in which this attribute is defined.                 |
| version       | The object’s version.                                                       |
| type          | The attribute’s IDL type code.                                              |
| mode          | The mode of this attribute; read-only or read-write. See AttributeMode on page 7-5. |

AttributeMode

public class AttributeMode

This class is used to represent the mode of an attribute; either read-only or normal (read-write).

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

Table 7.2 AttributeMode globals and constants

| Constant       | Represents                                           |
|----------------|------------------------------------------------------|
| ATTR_NORMAL    | An attribute definition as Normal.                  |
| ATTR_READONLY  | An attribute definition as read-only.                |

Methods

public int value()

This method returns the value of this object, which is either _ATTR_NORMAL or _ATTR_READONLY.

ConstantDef

public interface ConstantDef extends org.omg.CORBA.Contained

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 classes,” for more information on these classes and the methods they offer.

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 original typedef for this object.

| Parameter    | Description                |
|--------------|----------------------------|
| type_def     | The IDLType of this object.|

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 |
|-----------|-------------|
| value     | An Any object that represents this object's value. |

ConstantDescription

public class ConstantDescription

The ConstantDescription class describes a constant that is stored in the interface repository.

Variables

public String name

This variable represents the name of the constant.

public String id

This variable represents the repository id of the constant.

public String defined_in

This variable represents the name 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.

Methods

public ConstantDescription()

This method is the default constructor for a ConstantDescription.

public constantDescription(String name, String id, 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.
public interface Contained extends org.omg.CORBA.IRObject

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 classes,” for more information on these classes and the methods they offer.

IDL definition

interface Contained : CORBA::IRObjec {  
    attribute CORBA::RepositoryId id;
    attribute CORBA::Identifier name;
    attribute CORBA::VersionSpec version;
    readonly attribute CORBA::Container defined_in;
    readonly attribute CORBA::ScopedName absolute_name;
    readonly attribute CORBA::Repository containing_repository;
    struct Description {  
        CORBA::DefinitionKind kind;
        any value;
    };
    CORBA::Contained::Description describe();
    void move(  
        in CORBA::Container new_container,
        in CORBA::Identifier new_name,
        in CORBA::VersionSpec new_version);
};

| 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.                     |
Methods

public 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 String id()

    This method returns this object’s repository identifier.

public void id(String id)

    This method sets the repository identification that uniquely identifies this object.

| Parameter | Description         |
|-----------|---------------------|
| id        | The repository identifier for this object. |

public String name()

    This method returns this object’s name.

public void name(String name)

    This method sets the name for this object.

| Parameter | Description |
|-----------|-------------|
| name      | The object’s name. |

public org.omg.CORBA.VersionSpec version()

    This method returns this object’s version.

public void version(org.omg.CORBA.VersionSpec 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, org.omg.CORBA.VersionSpec 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**

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 classes,” for more information on these classes and the methods they offer.

**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.

**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 “ContainedPackage.Description” on page 7-16 for more information. |
| value     | An Any object that represents the value for this item. |
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 classes,” for more information on these classes and the methods they offer.

IDL definition

```idl
interface Container : CORBA::IRObject {
    CORBA::Contained lookup(in CORBA::ScopedName search_name);
    CORBA::ContainedSeq contents(in CORBA::DefinitionKind limit_type,
        in boolean exclude_inherited);
    CORBA::ContainedSeq lookup_name(in CORBA::Identifier search_name,
        in long levels_to_search,
        in CORBA::DefinitionKind limit_type,
        in boolean exclude_inherited);
    struct Description {
        CORBA::Contained contained_object;
        CORBA::DefinitionKind kind;
        any value;
    };
    typedef sequence<CORBA::Container::Description> DescriptionSeq;
    CORBA::Container::DescriptionSeq describe_contents(
        in CORBA::DefinitionKind limit_type,
        in boolean exclude_inherited,
        in long max_returned_objs);
    CORBA::ContainedSeq private_lookup(in CORBA::Identifier search_name,
        in long levels_to_search,
        in CORBA::DefinitionKind limit_type,
        in boolean exclude_inherited,
        in boolean exclude_inherited,
        in long max_returned_objs);
    void private_add_name(in CORBA::Contained cont);
    void private_remove_name(in CORBA::Contained cont);
    CORBA::ModuleDef create_module(in CORBA::RepositoryId id,
        in CORBA::Identifier name,
        in CORBA::VersionSpec version);
    CORBA::ConstantDef create_constant(
        in CORBA::RepositoryId id,
        in CORBA::Identifier name,
        in CORBA::VersionSpec version,
        in CORBA::IDLType type,
        in any value);
    CORBA::StructDef create_struct(
        in CORBA::RepositoryId id,
        in CORBA::Identifier name,
        in CORBA::VersionSpec version,
        in CORBA::IDLType type,
        in CORBA::IDLType type,
        in any value);
}
```
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 AliasDef create_alias(String id, String name, org.omg.CORBA.VersionSpec version, org.omg.CORBA.IDLType original_type)

This method creates a AliasDef object in this Container with the specified attributes and returns a reference to the newly created object.

| Parameter     | Description                                      |
|---------------|--------------------------------------------------|
| id            | The alias’ repository id.                       |
| name          | The alias’ name.                                |
| version       | The alias’ version.                             |
| original_type | The IDL type of the original object for which this is an alias. |

public org.omg.CORBA.ConstantDef create_constant(String id, String name, org.omg.CORBA.VersionSpec 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                                      |
|---------------|--------------------------------------------------|
| id            | The constant’s repository id.                   |
| name          | The constant’s name.                            |
| version       | The constant’s version.                         |
| type          | The constant’s IDL type.                        |
| value         | The constant’s value, represented by an Any object. |

public org.omg.CORBA.EnumDef create_enum(String id, String name, org.omg.CORBA.VersionSpec version, String members[])

This method creates a EnumDef object in this Container with the specified attributes and returns a reference to the newly created object.

| Parameter     | Description                                      |
|---------------|--------------------------------------------------|
| id            | The enumeration’s repository id.                 |
| name          | The enumeration’s name.                         |
| version       | The enumeration’s version.                      |
| members       | A list of the enumeration’s values.              |

public org.omg.CORBA.EstructDef create_estruct(String id, String name, org.omg.CORBA.VersionSpec version, org.omg.CORBA.EstructDef base, org.omg.CORBA.StructMember members[])

This method creates an EstructDef object in this Container with the specified attributes and returns a reference to the newly created object.

| Parameter     | Description                                      |
|---------------|--------------------------------------------------|
| id            | The extensible structure’s repository id.       |
| name          | The extensible structure’s name.                |
| version       | The extensible structure’s version.             |
public org.omg.CORBA.ExceptionDef
create_exception(String id, String name,
org.omg.CORBA.VersionSpec version, org.omg.CORBA.StructMember members[])

This method creates a \texttt{ExceptionDef} object in this \texttt{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(String id, String name,
org.omg.CORBA.VersionSpec version, org.omg.CORBA.InterfaceDef base_interfaces)

This method creates a \texttt{InterfaceDef} object in this \texttt{Container} with the specified attributes and returns a reference to the newly created object.

| 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(String id, String name,
org.omg.CORBA.VersionSpec version)

This method creates a \texttt{ModuleDef} object in this \texttt{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.StructDef
create_struct(String id, String name,
org.omg.CORBA.VersionSpec version, org.omg.CORBA.StructMember members[])

This method creates a \texttt{StructDef} object in this \texttt{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.                                                      |
### Methods

**public org.omg.CORBA.UnionDef create_union(String id, String name, org.omg.CORBA.VersionSpec 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                                                      |
|--------------------|------------------------------------------------------------------|
| version            | The structure’s version.                                         |
| members            | The values for the structure’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                                                      |
|--------------------|------------------------------------------------------------------|
| limit_type         | The interface object types to be returned.                       |
| exclude_inherited  | If set to true, inherited objects will not be returned.          |
| max_returned_objs  | The maximum number of object to be returned. Setting this parameter to –1 will return all objects. |

**public org.omg.CORBA.Contained lookup(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                                                      |
|--------------------|------------------------------------------------------------------|
| search_name        | The name of the object to be located.                            |
ContainerPackage.Description

public org.omg.CORBA.Contained[] lookup_name(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                                                                 |
|-------------------------|-----------------------------------------------------------------------------|
| search_name             | The name of the object or objects to be located.                             |
| levels_to_search        | 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. |
| limit_type              | The interface object types to be returned.                                  |
| exclude_inherited       | If set to true, inherited objects will not be returned.                     |

ContainerPackage.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 classes,” for more information on these classes and the methods they offer.

Variables

public org.omg.CORBA.Contained contained_object

This variable represents contained item.

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.

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 an Description, using the supplied parameters.

| Parameter      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| contained_object | The contained item.                                                         |
| kind           | This item’s kind. See “ContainerPackage.Description” on page 7-16 for more information. |
| value          | An Any object that represents the value for this item.                      |

public class DefinitionKind

The DefinitionKind class contains the constants that define the possible types of interface repository objects. 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 class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.

Methods

public int value()

This method returns an integral value representing the constant.

Variables

Table 7.3 DefinitionKind globals and constants

| Constant    | Represents                                                                 |
|-------------|-----------------------------------------------------------------------------|
| dk_none     | Exclude all types (used in repository lookup methods).                      |
| dk_any      | All possible types (used in repository lookup methods).                    |
| dk_Alias    | An Alias.                                                                  |
| dk_Array    | An Array.                                                                  |
| dk_Attribute | An Attribute.                                                              |
| dk_Constant | A Constant.                                                                |
| dk_Enum     | An Enum.                                                                   |
| dk_Estruct  | An extended structure.                                                    |
| dk_Exception | An Exception                                                               |
| dk_Interface | An Interface.                                                             |
| dk_Module   | A Module.                                                                  |
| dk_Operation | An Interface Operation.                                                 |
| dk_Primitive | A primitive type (such as int or long).                                   |
| dk_TypeDef  | A Typedef.                                                                 |
public interface EnumDef extends org.omg.CORBA.TypeDef

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 classes,” for more information on these classes and the methods they offer.

## Methods

public String[] members()

This method returns the enumeration’s list of members.

dk_Union

dk_Repository

dk_Sequence

dk_String

dk_Struct

dk_Wstring

A Union.

A Repository.

A Sequence.

A String.

A Struct.

A Unicode string.

public void members(String members[])

This method sets the enumeration’s list of members.

| Parameter | Description |
|-----------|-------------|
| members   | The list of members. |
**Methods**

**public org.omg.CORBA.TypeCode base()**

This method returns the TypeCode of the base class for this Estruct.

**public void base_def(org.omg.CORBA.EstructDef base_def)**

This method sets the base class for this EstructDef.

| Parameter | Description               |
|-----------|---------------------------|
| base_def  | An EstructDef that represents the base class. |

**public org.omg.CORBA.EstructDef base_def()**

This method returns the EstructDef that represents the base class for this Estruct.

**public org.omg.CORBA.StructMember[] members()**

This method returns the list of members for this EstructDef.

**public void members(StructMember members[])**

This method sets the list of members for this EstructDef.

| Parameter | Description               |
|-----------|---------------------------|
| members   | The list of members for this EstructDef. |

**public void members(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.Contained**

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 classes,” for more information on these classes and the methods they offer.
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 class ExceptionDescription

The ExceptionDescription class describes a exception that is stored in the interface repository.

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

Variables

public String name

This variable represents name of the exception.

public String id

This variable represents the repository id of the exception.

public String defined_in

This variable represents the name of the module or interface in which this exception is defined.

public org.omg.CORBA.VersionSpec version

This variable represents the exception’s version.

public org.omg.CORBA.TypeCode type

This variable represents the exception’s IDL type.

public org.omg.CORBA.Any value

This variable represents the value of this exception.
Methods

public ExceptionDescription()  
This method is the default constructor for a ExceptionDescription.

public ExceptionDescription(String name, String id, 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                       |
|-------------|-----------------------------------|
| name        | The name of this exception.       |
| id          | The repository id for this exception. |
| defined_in  | The module or interface in which this exception is defined. |
| version     | The object’s version.             |
| type        | The exception’s IDL type code.    |
| value       | The value of this exception.      |

IDLType

public interface IDLType extends org.omg.CORBA-IRObj

The IDLType interface is used to represent the TypeCode associated with an interface repository object. The TypeCode uniquely identifies the object’s type and is described on page 6-29.

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

IDL definition

interface IDLType : CORBA::IRObj {
    readonly attribute TypeCode type;
};

Methods

public org.omg.CORBA-TypeCode type()  
This method returns the TypeCode of the current IRObj.
The interface is used to represent an object implementation interface that is stored in the Interface Repository. This interface provides methods for setting and retrieving the base interface as well as creating attributes, operations and an interface description.

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

**IDL definition**

```idl
define InterfaceDef : CORBA::Container, CORBA::Contained, CORBA::IDLType {}  
attribute CORBA::InterfaceDefSeq base_interfaces;  
boolean is_a(in CORBA::RepositoryId interface_id);  
struct FullInterfaceDescription {}  
  CORBA::Identifier name;  
  CORBA::RepositoryId id;  
  CORBA::RepositoryId defined_in;  
  CORBA::VersionSpec version;  
  CORBA::OpDescriptionSeq operations;  
  CORBA::AttrDescriptionSeq attributes;  
  CORBA::RepositoryIdSeq base_interfaces;  
  TypeCode type;  
};  
CORBA::InterfaceDef::FullInterfaceDescription describe_interface();  
CORBA::AttributeDef create_attribute(  
in CORBA::RepositoryId id,  
in CORBA::Identifier name,  
in CORBA::VersionSpec version,  
in CORBA::IDLType type,  
in CORBA::AttributeMode mode);  
CORBA::OperationDef create_operation(  
in CORBA::RepositoryId id,  
in CORBA::Identifier name,  
in CORBA::VersionSpec version,  
in CORBA::IDLType result,  
in CORBA::OperationMode mode,  
in CORBA::ParDescriptionSeq params,  
in CORBA::ExceptionDefSeq exceptions,  
in CORBA::ContextIdSeq contexts);  
};
```
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(String id, String name, org.omg.CORBA.VersionSpec 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.        |

public org.omg.CORBA.OperationDef create_operation(String id, String name, org.omg.CORBA.VersionSpec version, org.omg.CORBA.IDLType result, org.omg.CORBA.OperationMode mode, org.omg.CORBA.ParameterDescription[] params, org.omg.CORBA.ExceptionDef[] exceptions, 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 type.                                        |
| mode       | The operation’s mode.                                            |
| 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.InterfaceDef.FullInterfaceDescription describe_interface()

This method returns an interface description for this object.
public boolean is_a(String interface_id)

This method returns true if this object has the specified interface identifier.

| Parameter   | Description                  |
|-------------|------------------------------|
| interface_id| The interface identifier to compare with this object. |

### InterfaceDefPackage.FullInterfaceDescription

public class FullInterfaceDescription

This class provides a description of an interface that is stored in the interface repository.

### Variables

public String name

This variable represents name of the interface.

public String id

This variable represents the repository id of the interface.

public String defined_in

This variable represents the name of the module in which this interface is defined.

public org.omg.CORBA.VersionSpec version

This variable represents the interface’s version.

public org.omg.CORBA.OperationDescription[] operations

This variable represents the list of operations offered by the interface.

public org.omg.CORBA.AttributeDescription[] attributes

This variable represents interface’s list of attributes.

public org.omg.CORBA.TypeCode type

This variable represents the interface’s IDL type.

### Methods

public FullInterfaceDescription()

This method is the default constructor for a FullInterfaceDescription.
public FullInterfaceDescription(String name, String id, String defined_in,
org.omg.CORBA.VersionSpec version, org.omg.CORBA.OperationDescription[] operations,
org.omg.CORBA.AttributeDescription[] attributes, org.omg.CORBA.TypeCode type)

This method constructs an FullInterfaceDescription, using the supplied parameters.

| Parameter   | Description                                      |
|-------------|--------------------------------------------------|
| name        | The name of this interface.                     |
| id          | The repository id for this interface.           |
| defined_in  | The module or interface in which this attribute is defined. |
| version     | The object’s version.                           |
| operations  | The list of operation offered by this interface.|
| attributes  | The list of this interface’s attributes.         |
| type        | The interface’s IDL type code.                   |

public class InterfaceDescription

The InterfaceDescription class describes a constant that is stored in the interface repository.

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

Variables

public String Name

This variable represents name of the interface.

public String id

This variable represents the repository id of the interface.

public String defined_in

This variable represents the name of the module or interface in which this interface is defined.

public org.omg.CORBA.VersionSpec version

This variable represents the interface’s version.

public String[] base_interfaces

This variable represents a list of base interfaces for this interface.
### Methods

**public InterfaceDescription()**

This method is the default constructor for an InterfaceDescription.

**public InterfaceDescription(String name, String id, String defined_in, org.omg.CORBA.VersionSpec version, 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 or interface in which this interface is defined. |
| version            | The object’s version.            |
| base_interfaces    | The interface’s list of base interfaces. |

### IRObjec

**public interface IRObjec extends org.omg.CORBA.Object**

The IRObjec 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 classes,” for more information on these classes and the methods they offer.

#### IDL definition

```idl
interface IRObjec {
    readonly attribute CORBA::DefinitionKind def_kind;
    void destroy();
};
```

#### Methods

**public org.omg.CORBA.DefinitionKind def_kind()**

This method returns the type of this IRObjec. For a list of defined types, see “TCKind” on page 6-28.

**public void destroy()**

This method deletes this IRObjec from the Interface Repository.
public interface ModuleDef extends org.omg.CORBA.Container, org.omg.CORBA.Contained

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 classes,” for more information on these classes and the methods they offer.

public class ModuleDescription

The ModuleDescription class describes a constant that is stored in the interface repository. Helper and Holder versions of this class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.

Variables

public String name

This variable represents name of the module.

public String id

This variable represents the repository id of the module.

public String defined_in

This variable represents the name of the module in which this interface is defined.

public org.omg.CORBA.VersionSpec version

This variable represents the module’s version.

Methods

public ModuleDescription()

This method is the default constructor for an ModuleDescription.
public ModuleDescription(String name, String id, String defined_in,
org.omg.CORBA.VersionSpec version)

This method constructs an ModuleDescription, using the supplied parameters.

| Parameter   | Description                        |
|-------------|------------------------------------|
| name        | The name of this interface.        |
| id          | The repository id for this interface. |
| defined_in  | The module or interface in which this interface is defined. |
| version     | The object’s version.              |

public interface OperationDef extends org.omg.CORBA.Contained

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 classes,” for more information on these classes and the methods they offer.

Methods

public String[] contexts()

This method returns the contexts associated with this operation.

public void contexts(String[] contexts)

This method sets this operation’s list of context.

| 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 7-31 for more details. |

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 class OperationDescription

The OperationDescription class describes a constant that is stored in the interface repository.

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

**Variables**

public String name

This variable represents name of the operation.

public String id

This variable represents the repository id of the operation.
Methods

public String defined_in

This variable represents the name of the operation in which this interface is defined.

public org.omg.CORBA.VersionSpec version

This variable represents the operation’s version.

public org.omg.CORBA.TypeCode result

This variable represents the operation’s result.

public org.omg.CORBA.OperationMode mode

This variable represents the operation’s mode.

public String[] contexts

This variable represents the operation’s contexts.

public org.omg.CORBA.ParameterDescription[] parameters

This variable represents the operation’s parameters.

public org.omg.CORBA.ExceptionDescription[] exceptions

This variable represents the exceptions that this operation may raise.

Methods

public OperationDescription()

This method is the default constructor for an OperationDescription.

public OperationDescription(String name, String id, String defined_in, org.omg.CORBA.VersionSpec version, org.omg.CORBA.TypeCode result, org.omg.CORBA.OperationMode mode, String[] contexts, org.omg.CORBA.ParameterDescription[] parameters, org.omg.CORBA.ExceptionDescription[] exceptions)

This method constructs an OperationDescription, using the supplied parameters.

| Parameter      | Description                                                                 |
|----------------|------------------------------------------------------------------------------|
| name           | The name of this interface.                                                  |
| id             | The repository id for this interface.                                       |
| defined_in     | The operation or interface in which this interface is defined.              |
| version        | The object’s version.                                                        |
| result         | The IDL type of the result of the operation.                                |
| mode           | The operation’s mode.                                                       |
| contexts       | A list of Context objects for this operation.                               |
| parameters     | The list of parameters for this operation.                                  |
| exceptions     | The list of exceptions that this operation may raise.                       |
public class OperationMode

The class is used to represent the mode of an operation; either oneway or normal. Oneway operations are those for which the client application does not expect a response. 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 classes,” for more information on these classes and the methods they offer.

Variables

| Table 7.4 | OperationMode globals and constants |
|-----------|-------------------------------------|
| Constant  | Represents                          |
| OP_NORMAL | A normal operation.                 |
| OP_ONEWAY | A one-way operation.                |

Methods

public int value()

This method returns the value of this object, which is either _OP_NORMAL or _OP_ONEWAY.

ParameterDescription

public 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 classes,” for more information on these classes and the methods they offer.

Variables

public String name

This variable represents name of the parameter.

public org.omg.CORBA.TypeCode type

This variable represents the parameter’s type.
Methods

public org.omg.CORBA.IDLType type_def

This variable represents the parameter’s IDL type.

public org.omg.CORBA.ParameterMode mode

This variable represents the parameter’s mode.

Methods

public ParameterDescription()

This method is the default constructor for an ParameterDescription.

public ParameterDescription(String name, org.omg.CORBA.TypeCode type, org.omg.CORBA.IDLType type_def, org.omg.CORBA.ParameterMode mode)

This method constructs an 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 class ParameterMode

The class is used to represent the mode of a parameter. Parameters may be used in one of the following three ways:

- IN Used for input from the client to the server.
- OUT Used for output of results from the server to the client.
- INOUT Used for both input from the client and output from the server.

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

| Constant       | Represents          |
|----------------|---------------------|
| PARAM_IN       | A IN parameter.     |
| OP_PARAM_OUT   | An OUT parameter.   |
| OP_PARAM_INOUT | An INOUT parameter. |

Methods

public int value()

This method returns the value of this object, which is either _PARAM_IN, _PARAM_OUT, or _PARAM_INOUT.

PrimitiveDef

public interface PrimitiveDef extends org.omg.CORBA.IDLType

The interface is used to represent an primitive (such as an int or a long) that is stored in the Interface Repository. This interface provides a methods for retrieving the kind of primitive that is being represented.

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

Methods

public org.omg.CORBA.PrimitiveKind kind()

This method returns the kind of primitive represented by this object.

PrimitiveKind

public class PrimitiveKind

The PrimitiveKind class contains the constants that define the possible types of primitives interface repository objects. There are a set of integer constants, prefixed with pk_, that correspond to all the possible kinds of primitives.

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

public int value()

This method returns an integral value representing the constant.

Variables

| Constant     | Represents               |
|--------------|-------------------------|
| pk_null      | A null.                 |
| pk_void      | A void.                 |
| pk_short     | A short.                |
| pk_long      | A long.                 |
| pk_ushort    | A unsigned short.       |
| pk_ulong     | An unsigned long.       |
| pk_float     | A float.                |
| pk_double    | A double.               |
| pk_boolean   | A boolean.              |
| pk_char      | A character.            |
| pk_octet     | An octet string.        |
| pk_any       | An Any object.          |
| pk_TypeCode  | A TypeCode object.      |
| pk_Principal | A Principal object.     |
| pk_string    | A string.               |
| pk_objref    | An object reference.    |
| pk_longlong  | A long long.            |
| pk_ulonglong | An unsigned long.       |
| pk_longdouble| A long double.          |
| pk_wchar     | A Unicode character.    |
| pk_wstring   | A Unicode string.       |

Repository

public interface Repository extends org.omg.CORBA.Container

The Repository provides an interface to the Interface Repository, which is used to contain the definitions of objects that are available to clients. The Repository interface provides methods for storing and retrieving definitions.

Helper and Holder versions of this class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.
**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                               |
|---------------|-------------------------------------------|
| length        | The number of elements in the array. This value must be greater than zero. |
| element_type  | 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 sequence definition in the repository with the specified bound and element type. A reference to the SequenceDef that is created is returned.

| Parameter     | Description                               |
|---------------|-------------------------------------------|
| bound         | The maximum length of the sequence. This value must be greater than zero. |
| element_type  | 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 bound. A reference to the StringDef that is created is returned.

| Parameter     | Description                               |
|---------------|-------------------------------------------|
| bound         | The maximum bounds of the string. This value must be greater than zero. |

public org.omg.CORBA.WstringDef create_wstring(int bound)

This method creates a Unicode string definition in the repository with the specified bound. A reference to the WstringDef that is created is returned.

| Parameter     | Description                               |
|---------------|-------------------------------------------|
| bound         | The maximum bounds of the string. This value must be greater than zero. |

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’s kind. |
**SequenceDef**

```java
public Contained lookup_id(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. |

**SequenceDef**

```java
public interface SequenceDef extends org.omg.CORBA.IDLType

The interface is used to represent an 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 classes,” for more information on these classes and the methods
they offer.
```

**Methods**

```java
public int bound()

This method returns the bounds of the sequence.
```

```java
public void bound(int bound)

This method sets the bound of the sequence.
```

| Parameter  | Description          |
|------------|----------------------|
| members    | The list of members. |

```java
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 |
|-----------------|-------------|
| element_type_def| The IDL type to set. |
**StringDef**

public interface StringDef extends org.omg.CORBA.IDLType

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 classes,” for more information on these classes and the methods they offer.

**Methods**

public int bound()

This method returns the bounds of the String.

public void members(int bound)

This method sets the bounds of the String.

| Parameter | Description  |
|-----------|--------------|
| members   | The list of members. |

**StructDef**

public interface StructDef extends org.omg.CORBA.TypedefDef

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 classes,” for more information on these classes and the methods they offer.

**Methods**

public org.omg.CORBA.StructMember[] members()

This method returns the structure’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. |
**TypedefDef**

public interface **TypedefDef** extends org.omg.CORBA.Contained, org.omg.CORBA.IDLType

This abstract interface represents a user-defined structure that is stored in the Interface Repository. The following interfaces all inherit from this interface:

- AliasDef, described on page 7-2.
- EnumDef, described on page 7-18.
- EstructDef, described on page 7-18.
- StructDef, described on page 7-37.
- UnionDef, described on page 7-39.
- WstringDef, described on page 7-41.

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

**TypeDescription**

public class **TypeDescription**

The **TypeDescription** class describes a type for an operation that is stored in the interface repository.

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

**Variables**

public String **name**

This variable represents name of the type.

public String **id**

This variable represents the repository id of the type.

public String **defined_in**

This variable represents the name of the module or interface in which this type is defined.

public org.omg.CORBA.VersionSpec **version**

This variable represents the type’s version.

public org.omg.CORBA.TypeCode **type**

This variable represents the type’s IDL type.
Methods

public TypeDescription()

This method is the default constructor for an TypeDescription.

public TypeDescription(String name, String id, String defined_in, org.omg.CORBA.VersionSpec version, org.omg.CORBA.TypeCode type)

This method constructs an 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.TypedefDef

The interface is used to represent an Union that is stored in the Interface Repository. This interface provides methods for setting and retrieving the union’s list of members and discriminator type.

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

Methods

public org.omg.CORBA.TypeCode discriminator_type()

This method returns the TypeCode of the discriminator for the Union.

public org.omg.CORBAIDLType 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         |
|-----------------|---------------------|
| discriminator_type_def | The list of members. |

public org.omg.CORBA.UnionMember[] members()

This method returns the union’s list of members.
UnionMember

```java
class UnionMember
```

The `UnionMember` class describes a union that is stored in the interface repository. Helper and Holder versions of this class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.

### Variables

```java
public String name
```

This variable represents name of the union.

```java
public org.omg.CORBA.Any label
```

This variable represents the label of the union.

```java
public org.omg.CORBA.TypeCode type
```

This variable represents the union’s type.

```java
public org.omg.CORBA.IDLType type_def
```

This variable represents the union’s IDL type.

### Methods

```java
public UnionMember()
```

This method is the default constructor for an `UnionMember`.

```java
public UnionMember(String name, org.omg.CORBA.Any label, org.omg.CORBA.TypeCode type, org.omg.CORBA.IDLType type_def)
```

This method constructs an `UnionMember`, using the supplied parameters.

| Parameter | Description |
|-----------|-------------|
| name      | The name of this union. |
| label     | The label for this union. |
| type      | The union’s type. |
| type_def  | The union’s IDL type code. |
public interface VersionSpec

The VersionSpec interface describes the version of an object that is stored in the interface repository. Helper and Holder versions of this class are also provided. See Chapter 4, “Generated classes,” for more information on these classes and the methods they offer.

Variables

public short major

This major version number.

public short minor

This minor version number.

Methods

public VersionSpec()

This method is the default constructor for an UnionMember.

public VersionSpec(short major, short minor)

This method constructs a VersionSpec, using the supplied parameters.

| Parameter | Description |
|-----------|-------------|
| major     | The major version number. |
| minor     | The minor version number. |

public interface WstringDef extends org.omg.CORBA.IDLType

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 classes,” for more information on these classes and the methods they offer.

Methods

public int bound()

This method returns the bounds of the WString.
public void members(int bound)

This method sets the bounds of the WString.

| Parameter | Description |
|-----------|-------------|
| members   | The list of members. |
This chapter describes the ActivationImplDef, Activator, CreationImplDef, ImplementationDef, and OAD—dynamic interfaces and classes in the Activation package. They are used with the Object Activation Daemon (OAD). This chapter includes the following major sections:

- ActivationImplDef: page 8-1
- Activator: page 8-2
- CreationImplDef: page 8-3
- ImplementationDef: page 8-7
- OAD: page 8-8

ActivationImplDef

Public interface ActivationImplDef

The ActivationImplDef interface provides a set of attributes for an Activator.

```java
interface ActivationImplDef : ::CORBA::ImplementationDef {
    readonly attribute string service_name;
    readonly attribute ::CORBA::ReferenceData id;
    readonly attribute ::extension::Activator activator_obj;
}
```

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.

class public abstract String service_name()  
This method retrieves the implementation’s service name.

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 benefit from greater performance. If you have thousands of objects on a server, you can save system resources by only activating the objects when clients request them. You can defer the 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);  
    };

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 under the responsibility of the Activator, 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 |
|-----------|-------------|
| impl      | The instance of ImplementationDef. |

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 |
|-----------|-------------|
| obj       | The object reference of the object to deactivate. |
| impl      | The instance of ImplementationDef. |
public interface CreationImplDef extends org.omg.CORBA.ImplementationDef

The `CreationImplDef` interface provides a set 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**

```idl
textarea

interface CreationImplDef : CORBA::ImplementationDef {
    attribute CORBA::RepositoryId repository_id;
    attribute string object_name;
    attribute CORBA::ReferenceData id;
    attribute string path_name;
    attribute CORBA::Policy activation_policy;
    attribute CORBA::StringSequence args;
    attribute 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 (and ORBELINE for compatibility with previous versions of VisiBroker)

**C++ example**
To activate the VisiBroker for C++ application “factory_r” (located in a developer’s local directory) with one argument “ReentrantServer” and the environment `LD_LIBRARY_PATH` set properly for the OAD’s environment, you would fill out a `CreationImplDef` with the following attributes:

```
path_name = "/home/developer/Project1/factory_r"
args = ["ReentrantServer"]
env = ["LD_LIBRARY_PATH=/usr/ucblib:/usr/local/VisiCpp/lib"]
```

It would correspond to the OAD spawning the following command:
```
/home/developer/Project1/factory_r ReentrantServer \
  -OAoad_uid=<unique_id> -OAactivateIOR=<oad's ior>
```

In addition, the environment variables listed in “Environment variables that are propagated or passed explicitly” on page 8-5 would be propagated from the OAD’s environment into that of the spawned server.
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 example, if spawning a C++ implementation on Solaris, you would need to explicitly register an LD_LIBRARY_PATH in the CreationImplDef's environment if the spawned executable requires shared libraries.

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 }
```

For spawned C++ applications, the registration maps to the following command:

```exec-path { args1 ... argsN }
-OAoad_uid=<unique_id> -OAactivateIOR=<oad's ior>
```

In both cases, 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 (or ORBELINE for backward compatibility)
• LD_LIBRARY_PATH as set in the 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               |
|-----------------|---------------------------|
| activation_policy | 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. For Java, you must specify the class name as the first argument. For more information, see “Examples” on page 8-4.

| Parameter | Description               |
|-----------|---------------------------|
| args      | 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 8-5.

| Parameter | Description               |
|-----------|---------------------------|
| env       | 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               |
|-----------|---------------------------|
| id        | 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                                      |
|----------------|--------------------------------------------------|
| object_name    | A string containing the implementation’s object name. |

public abstract String path_name()

For C++ programs, this method retrieves the exact path name of the executable program that implements the object.

For Java 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 C++ programs, you must set the full path name. For Java 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                                      |
|------------|--------------------------------------------------|
| path_name  | 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.
public interface OAD extends org.omg.CORBA.Object

The OAD interface provides access to the OAD (Object Activation Daemon). It is used by the following 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 CreationImplDef impl)
        raises(Activation::DuplicateEntry, Activation::InvalidPath);
    CreationImplDef get_implementation(in CORBA::RepositoryId repId,
        in string object_name)
        raises(Activation::NotRegistered);
    void change_implementation(in CreationImplDef old_info,
        in CreationImplDef new_info, in CORBA::RepositoryId repId,
        in string object_name)
        raises(Activation::NotRegistered, Activation::InvalidPath,
            Activation::IsActive);
    attribute boolean destroy_on_unregister;
    void unreg_implementation(in CORBA::RepositoryId repId, in string object_name)
        raises(Activation::NotRegistered);
    void unreg_interface(in CORBA::RepositoryId repId)
        raises(Activation::NotRegistered);
    void unregister_all();
    Activation::ImplementationStatus get_status(in CORBA::RepositoryId repId,
        in string object_name)
        raises(Activation::NotRegistered);
    Activation::ImplStatusList get_status_interface(in CORBA::RepositoryId repId)
        raises(Activation::NotRegistered);
    Activation::ImplStatusList get_status_all();
};

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;
};

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                      |
|-------------|----------------------------------|
| old_info    | The information you want to change. |
| new_info    | The information to replace the old info. |

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. |

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 create_CreationImplDef()

This method creates an instance of CreationImplDef. You can then set its attributes as explained in “CreationImplDef” starting on page 8–3.
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 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. |

public com.visigenic.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. |
This method throws the following exceptions.

| Exception      | Description                                                   |
|----------------|---------------------------------------------------------------|
| NotRegistered  | The object you specify is not registered. You must specify a  |
|                | registered object.                                            |

```java
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          |
|----------------|----------------------|
| impl           | The instance of CreationImplDef.                             |

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.                                  |

```java
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          |
|----------------|----------------------|
| 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.                                            |

```java
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.                  |
Methods

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.
This chapter describes the interfaces and classes that allow clients and servers to monitor the occurrence of specific ORB-related events. This chapter includes the following major sections:

- ClientEventHandler
- HandlerRegistry
- ImplEventHandler

### ClientEventHandler

The `ClientEventHandler` interface allows client applications to define methods that the ORB will invoke to handle events such as the success or failure of a bind request. If the `ClientEventHandler` is registered as a global event handler, its methods are invoked when events occur that are related to any object the client uses.

**IDL definition**

```idl
struct ConnectionInfo {
    string hostname;
    long port;
};

interface ClientEventHandler {
    void bind_succeeded(in Object obj, in ConnectionInfo info);
};
```
Methods

```java
void bind_failed(in Object obj);
void server_aborted(in Object obj);
void rebind_succeeded(in Object obj, in ConnectionInfo info);
void rebind_failed(in Object obj);
```

public abstract void `bind_succeeded`(Object `obj`, ConnectionInfo `info`)

This method is invoked by the ORB when the client’s request to bind to the ORB object has completed successfully.

| Parameter | Description |
|-----------|-------------|
| `obj`     | A reference to the ORB object to which the client is trying to bind. |
| `info`    | All the information associated with the client connection—host name and port number. |

public abstract void `bind_failed`(Object `obj`)

This method is invoked if the client’s bind request fails.

| Parameter | Description |
|-----------|-------------|
| `obj`     | A reference to the object for which the bind failed. |

public abstract void `server_aborted`(Object `obj`)

This method is invoked if the connection to the object implementation is lost.

| Parameter | Description |
|-----------|-------------|
| `obj`     | A reference to the server that aborted. |

public abstract void `rebind_succeeded`(Object `obj`, ConnectionInfo `info`)

This method is invoked when an attempt to reconnect to an object implementation succeeds. Clients that maintain state information can provide their own implementation of this method that restores the state of the client and object, once the client has been reconnected to the object.

| Parameter | Description |
|-----------|-------------|
| `obj`     | A reference to the ORB object to which the client has rebound. |
| `info`    | All the information associated with the new client connection—host name and port number. |
public abstract void rebind_failed(Object obj)

This method is invoked when an attempt to reconnect to an object implementation fails.

| Parameter | Description |
|-----------|-------------|
| obj       | A reference to the object to which the rebind has failed. |

**HandlerRegistry**

public interface HandlerRegistry extends Object

The HandlerRegistry interface is used to register and unregister the handlers for client and implementation events.

**IDL definition**

```java
interface HandlerRegistry {
    HandlerRegistry instance();

    void reg_obj_client_handler(in Object obj, in ClientEventHandler handler)
        raises(HandlerExists, InvalidObject);

    void reg_glob_client_handler(in ClientEventHandler handler)
        raises(HandlerExists);

    void reg_obj_impl_handler(in Object obj, in ImplEventHandler handler)
        raises(HandlerExists, InvalidObject);

    void reg_glob_impl_handler(in ImplEventHandler handler)
        raises(HandlerExists);

    void unreg_obj_client_handler(in Object obj)
        raises(NoHandler, InvalidObject);

    void unreg_global_client_handler() raises(NoHandler);

    void unreg_obj_impl_handler(in Object obj)
        raises(NoHandler, InvalidObject);

    void unreg_global_impl_handler()
        raises(NoHandler);
};
```
Methods

public abstract void reg_glob_client_handler(ClientEventHandler handler)

This method can be called by the client application to register an event handler for all objects the client uses.

**Note** If both an object event handler and a global event handler are registered, the object event handler will take precedence for events that occur which are related to its object. All other events will be handled by the global event handler.

| Parameter   | Description                                           |
|-------------|-------------------------------------------------------|
| handler     | A ClientEventHandler to be called for all client-side events. |

This method throws the following exceptions:

| Exception    | Description                                           |
|--------------|-------------------------------------------------------|
| HandlerExists| A global handler is already specified for client-side events. |

public abstract void reg_glob_impl_handler(ImplEventHandler handler)

This method can be called within a server application to register an event handler with the BOA for all implementations contained in that application.

**Note** If both an object event handler and a global event handler are registered, the object event handler will take precedence for events that occur which are related to its object. All other events will be handled by the global event handler.

| Parameter   | Description                                           |
|-------------|-------------------------------------------------------|
| handler     | A reference to the ImplEventHandler to be called for all server-side events. |

This method throws the following exceptions:

| Exception    | Description                                           |
|--------------|-------------------------------------------------------|
| HandlerExists| A global handler is already specified for client-side events. |

public abstract void reg_obj_client_handler(Object obj, ClientEventHandler handler)

This method can be called by your client application to register an event handler for a specific object.

| Parameter   | Description                                           |
|-------------|-------------------------------------------------------|
| obj         | A reference to the object.                           |
| handler     | A reference to the ClientEventHandler which will be notified of all client-side events for obj. |
**Methods**

This method throws the following exceptions:

| Exception      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| HandlerExists  | The object specified already has a ClientEventHandler. You must specify an unregistered object. |
| InvalidObject  | The object reference is not valid.                                           |

**public abstract void reg_obj_impl_handler(Object obj, ImplEventHandler handler)**

This method can be called by your object implementation to register an event handler with the BOA for a specific object.

**Parameter** | **Description**
|----------------|-----------------------------------------------------------------------------|
| obj            | The object’s reference.                                                     |
| handler        | A reference to the ClientEventHandler which will be notified of all client-side events for obj. |

This method throws the following exceptions:

| Exception      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| HandlerExists  | The object specified already has a ClientEventHandler. You must specify an unregistered object. |
| InvalidObject  | The object reference is not valid.                                           |

**public abstract void unreg_glob_client_handler()**

This method can be called by your client application to unregister its global event handler.

This method throws the following exception:

| Exception      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| NoHandler      | There is no such handler registered.                                        |

**public abstract void unreg_glob_impl_handler()**

This method can be called by your object implementation to unregister its global event handler.

This method throws the following exception:

| Exception      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| NoHandler      | There is no such handler registered.                                        |

**public abstract void unreg_obj_client_handler(Object obj)**

The unreg_obj_client_handler method can be called by your client application to unregister an event handler for a specific object.

**Parameter** | **Description**
|----------------|-----------------------------------------------------------------------------|
| obj            | A reference to the object whose event handler is to be unregistered.        |
**ImplEventHandler**

This method throws the following exceptions:

| Exception         | Description                                      |
|-------------------|--------------------------------------------------|
| NoHandler         | The object does not have a handler registered.   |
| InvalidObject     | The object reference is not valid.               |

```java
public abstract void unreg_obj_impl_handler(Object obj)
```

This method can be called by your object implementation to unregister an event handler for a specific object.

| Parameter | Description                                      |
|-----------|--------------------------------------------------|
| obj       | A reference to the object whose event handler is to be unregistered. |

This method throws the following exceptions:

| Exception         | Description                                      |
|-------------------|--------------------------------------------------|
| NoHandler         | The object does not have a handler registered.   |
| InvalidObject     | The object reference is not valid.               |

**ImplEventHandler**

```java
public interface ImplEventHandler extends Object
```

The `ImplEventHandler` enables object implementations to register an event handler with the ORB to handle events for a particular ORB object. The implementation can also globally register an event handler to handle events for all ORB objects implemented. Implementation-side event handling can be used for a variety of purposes. The implementation may refuse a client connection request, based on the caller’s identity.

**IDL definition**

```java
interface ImplEventHandler {
    void bind(in ConnectionInfo info, in Principal principal, in Object obj);
    void unbind(in ConnectionInfo info, in Principal principal, in Object obj);
    void client_aborted(in ConnectionInfo info, in Object obj);
    void pre_method(in ConnectionInfo info, in Principal principal, in string operation_name, in Object obj);
    void post_method(in ConnectionInfo info, in Principal principal, in string operation_name, in Object obj);
}```
Methods

public abstract void bind(ConnectionInfo info, Principal principal, Object obj)

This method is invoked every time a client requests a connection to the object. This method allows your object implementation to do any special processing before the bind is completed. Once this method returns, the BOA continues with the normal binding process. You may choose to reject the bind by raising a CORBA.NO_PERMISSION exception.

| Parameter | Description |
|-----------|-------------|
| info      | All the information associated with the client connection—host name and port number. |
| principal | An array of Principal values associated with the client. |
| obj       | A reference to the requested ORB object. |

public abstract void unbind(ConnectionInfo info, Principal principal, Object obj)

This method is invoked every time a client application closes its connection to a previously bound object. The BOA will pass control to this method before the unbind occurs. The connection information and the object reference are passed to this method.

| Parameter | Description |
|-----------|-------------|
| info      | All the information associated with the client connection—host name and port number. |
| principal | An array of Principal values associated with the client. |
| obj       | A reference to the bound ORB object. |

public abstract void client_aborted(ConnectionInfo info, Object obj)

This method is invoked if the connection to a client application is broken.

| Parameter | Description |
|-----------|-------------|
| info      | All the information associated with the client connection—host name and port number. |
| obj       | A reference to the previously connected ORB object. |

void post_method_exception(in ConnectionInfo info,
                        in Principal principal,
                        in string operation_name,
                        in string exception_rep_id,
                        in Object obj);}
public abstract void pre_method(ConnectionInfo info, Principal principal, 
String operation_name, Object obj)

This method is invoked every time a client application invokes a method on 
the object for which the handler is registered. After this method returns, the 
BOA will proceed with the method invocation.

| Parameter     | Description                                                                 |
|---------------|----------------------------------------------------------------------------|
| info          | All the information associated with the client connection—host name and port number. |
| principal     | An array of Principal values associated with the client.                    |
| operation_name| The method name.                                                            |
| obj           | A reference to the ORB object for which the handler is registered.          |

public abstract void post_method(ConnectionInfo info, Principal principal, 
String operation_name, Object obj)

This method is invoked after every invocation of a method by a client on the 
object being traced. After this method returns, the results of the method invocation will be returned to the client.

| Parameter     | Description                                                                 |
|---------------|----------------------------------------------------------------------------|
| info          | All the information associated with the client connection—host name and port number. |
| principal     | An array of Principal values associated with the client.                    |
| operation_name| The method name.                                                            |
| obj           | A reference to the ORB object being traced.                                |

public abstract void post_method_exception(ConnectionInfo info, Principal principal, 
String operation_name, String exception_rep_id, Object obj)

This method is invoked if an implementation method invoked by a client raises an exception. After this method returns, the exception will be returned to the client.

| Parameter          | Description                                                                 |
|--------------------|----------------------------------------------------------------------------|
| info               | All the information associated with the client connection—host name and port number. |
| principal          | An array of Principal values associated with the client.                    |
| operation_name     | The method name.                                                            |
| exception_rep_id   | The repository identifier of the exception that was raised.                  |
| obj                | A reference to the requested ORB object.                                     |
This chapter describes the exception classes used in VisiBroker. It includes the following major sections:

- **Introduction**: page 10-1
- **SystemException**: page 10-3
- **UserException**: page 10-3

### 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 inheritance hierarchy, the class `org.omg.CORBA.Exception` no longer exists.

```
org.omg.CORBA.UserException
    `java.lang.Exception`
```
```
UserEx1  UserEx2
    `org.omg.CORBA.COMM_FAILURE`, etc.
```
```
org.omg.CORBA.SystemException
    `java.lang.RuntimeException`
```
```
org.omg.CORBA.COMM_FAILURE, etc.
```
SystemException

abstract public class SystemException extends java.lang.RuntimeException

CORBA system exceptions are raised when the runtime encounters problems. They inherit from java.lang.RuntimeException. 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.

Attributes

public CompletionStatus completed

This attribute indicates whether the operation was completed or not.

Table 10.1  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.                           |
| 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_INDENT             | Invalid identifier syntax.                      |
| INV_OBJREF             | Invalid object reference specified.              |
| MARSHAL                | Error marshalling parameter or result.           |
| NOIMPLEMENT            | Operation implementation not available.          |
| NOMEMORY               | Dynamic memory allocation failure.               |
| NOPERMISSION           | No permission for attempted operation.           |
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 10.1.

Code sample 10.1 Catching system and user exceptions

```java
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);
}
```

Constructor

protected UserException()

This method creates a UserException exception.
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. This chapter includes the following major sections:

- BindInterceptor
- ChainUntypedObjectWrapperFactory
- ClientInterceptor
- Closure
- ServerInterceptor
- UntypedObjectWrapper
- UntypedObjectWrapperFactory

**BindInterceptor**

```java
public interface BindInterceptor extends Object
```

When using interceptors, a `BindInterceptor` is called by the ORB during each bind and rebind operation.

In addition, VisiBroker provides the `DefaultBindInterceptor` class that you can extend rather than implement. The default interceptor class offers the same methods as the `BindInterceptor` interface; however, when you extend the default interceptor class, you can choose which methods to implement or override. When you use the default class, you can accept the default behavior that it provides or change it.
Methods

interface BindInterceptor {
    boolean bind(
        inout IOR ior,
        in Object object,
        in Closure closure
    );
    boolean bind_failed(
        inout IOR ior,
        in Object object,
        in Closure closure
    );
    void bind_succeeded(
        in IOR ior,
        in Object object,
        in Closure closure
    );
    boolean rebind(
        inout IOR ior,
        in Object object,
        in Closure closure
    );
    boolean rebind_failed(
        inout IOR ior,
        in Object object,
        in Closure closure
    );
    void rebind_succeeded(
        in IOR ior,
        in Object object,
        in Closure closure
    );
    void exception_occurred(
        in IOR ior,
        in Object object,
        in Closure closure);
);

Methods

public abstract boolean bind(IORHolder ior, Object object, Closure closure)

This method is called during all ORB bind operations. Returns true if the ior argument was changed and should be used instead of the IOR provided by the ORB.

| Parameter | Description |
|-----------|-------------|
| ior       | The IOR of the server object to which the client is binding. |
Methods

Interceptor and object wrapper interfaces and classes

public abstract boolean bind_failed(IORHolder ior, Object object, Closure closure)

This method is called if a bind operation failed. Returns true if the ior argument was changed and the bind should be attempted again.

| Parameter | Description |
|-----------|-------------|
| ior       | The IOR of the server object on which the bind operation failed. |
| object    | The client object which is being bound to the server. |
| closure   | The closure object previously given in the bind call. |

public abstract void bind_succeeded(IOR ior, Object object, 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. |
| object    | The client object which is being bound to the server. |
| closure   | The closure object previously given in the bind call. |

public abstract void exception_occurred(IOR ior, Object object, Closure closure);

This method is called when an exception is thrown by one of the interceptors.

| Parameter | Description |
|-----------|-------------|
| ior       | The IOR of the server object. |
| object    | The client object which is being bound to the server. |
| closure   | The closure object previously given in the bind call. |

public abstract boolean rebind(IORHolder ior, Object object, Closure closure)

This method is called during all ORB rebind operations.

| Parameter | Description |
|-----------|-------------|
| ior       | The IOR of the server object to which the client is rebinding. |
| object    | The client object which is being bound to the server. The object will not be properly initialized at this time so no operation on the object should be attempted. It may, however, be stored in a data structure and used after the bind has completed. |
| closure   | A new closure object for the rebind operation. The closure will be used in corresponding call to either rebind_failed or rebind_succeeded. |
**ChainUntypedObjectWrapperFactory**

```java
public abstract boolean rebind_failed(IORHolder ior, Object object, Closure closure)
```

This method is called if a rebind operation failed. Returns true if the ior argument was changed and the bind should be attempted again.

| Parameter | Description                        |
|-----------|------------------------------------|
| ior       | The IOR of the server object on which the rebind operation failed. |
| object    | The client object which is being bound to the server. |
| closure   | The closure object previously given in the rebind call. |

```java
public abstract void rebind_succeeded(IOR ior, Object object, Closure closure)
```

This method is called if a rebind operation succeeded.

| Parameter | Description                        |
|-----------|------------------------------------|
| ior       | The IOR of the server object on which the rebind operation succeeded. |
| object    | The client object which is being bound to the server. |
| closure   | The closure object previously given in the rebind call. |

**ChainUntypedObjectWrapperFactory**

```java
public interface ChainUntypedObjectWrapperFactory extends com.visigenic.vbroker.interceptor.UntypedObjectWrapperFactory
```

This interface is used by a client or server application to add or remove an UntypedObjectWrapperFactory object. An UntypedObjectWrapperFactory, described on page 11–13, is used to create an UntypedObjectWrapper for each object a client application binds to or for each object implementation created by a server application.

See the VisiBroker for Java Programmer’s Guide for complete information using object wrappers.

```java
interface ChainUntypedObjectWrapperFactory {
    ::interceptor::UntypedObjectWrapperFactory factory
    
    void add(  
        in ::interceptor::UntypedObjectWrapperFactory factory
    );
    
    void remove(  
        in ::interceptor::UntypedObjectWrapperFactory factory
    );
}
```

**Methods**

```java
public void add(com.visigenic.vbroker.interceptor.UntypedObjectWrapperFactory factory);
```

This method adds the specified untyped object wrapper factory for a client or server application.
ClientInterceptor

public interface ClientInterceptor extends Object

You can create a client-side interceptor by implementing the `ClientInterceptor` interface. There is usually one client interceptor per object.

In addition, VisiBroker provides the `DefaultClientInterceptor` class that you can extend rather than implement. The default interceptor class offers the same methods as the `ClientInterceptor` interface; however, when you extend the default interceptor class, you can choose which methods to implement or override. When you use the default class, you can accept the default behavior that it provides or change it.

```java
interface ClientInterceptor {
    void prepare_request(  
        inout RequestHeader hdr,  
        in Closure closure  
    );
    OutputStream send_request(  
        in RequestHeader hdr,  
        in OutputStream buf,  
        in Closure closure  
    );
    void send_request_failed(  
        in RequestHeader hdr,  
        in Environment env,  
        in Closure closure  
    );
    void send_request_succeeded(  
        in RequestHeader hdr,  
        in Closure closure  
    );
}
```

### Parameters

| Parameter | Description |
|-----------|-------------|
| factory   | The object wrapper factory to be added. |
| factory   | The object wrapper factory to be removed. |
Methods

```java
public abstract void exception_occurred(int req_id, Environment env, Closure closure)
```

This method is called when an exception is thrown by one of the interceptors.

| Parameter | Description |
|-----------|-------------|
| req_id | The request id corresponding to the request for which no reply was received. |
| env | An environment object containing the exception indicating the exception which occurred. |
| closure | The closure object previously passed to `prepare_request`. |

```java
public abstract void prepare_request(RequestHeaderHolder hdr, Closure closure)
```

This method is called while the request is being prepared. Implementations can put service contexts into the GIOP request header as appropriate.

| Parameter | Description |
|-----------|-------------|
| hdr | The GIOP Request header associated with this request. The header may be changed by the interceptor during this call. |
| closure | A new closure object to be used for the processing of this request. |
public abstract void receive_reply(ReplyHeader hdr, InputStream buf, Environment env, Closure closure)

This method is called when a reply is received. If return value is not null, that value is passed to the upcall. This method can be used to implement message decryption.

| Parameter | Description |
|-----------|-------------|
| hdr       | The GIOP Request header associated with this request. |
| buf       | An input stream containing the marshalled values of the return value and any out parameters returned from the server. |
| env       | An environment containing the exception returned from the server, if any. |
| closure   | The closure object previously passed to prepare_request. |

public abstract void receive_reply_failed(int req_id, Environment env, Closure closure)

This method is called when an attempt to receive a reply fails. The failure is most likely a COMM_FAILURE. If there are user exceptions, they will be in CORBA::portable::InputStreams.

| Parameter | Description |
|-----------|-------------|
| req_id    | The request id corresponding to the request for which no reply was received. |
| env       | An environment object containing the exception indicating why the reply failed. |
| closure   | The closure object previously passed to prepare_request. |

public abstract OutputStream send_request(RequestHeader hdr, OutputStream buf, Closure closure)

This method is called when sending the request/message. If the return is “non null” the return buffer is sent instead of the input buffer. Returns a new output stream to replace the one given by the ORB, or null if the ORB should use the original output stream.

| Parameter | Description |
|-----------|-------------|
| hdr       | The GIOP Request header associated with this request. |
| buf       | This output stream contains all marshalled arguments associated with this request. |
| closure   | The closure object previously passed to prepare_request. |

public abstract void send_request_failed(RequestHeader hdr, Environment env, Closure closure)

This method is called when send failed. The failure is most likely a COMM_FAILURE.

| Parameter | Description |
|-----------|-------------|
| hdr       | The GIOP Request header associated with this request. |
| env       | An environment object containing the exception indicating why the request failed. |
| closure   | The closure object previously passed to prepare_request. |
public abstract void send_request_succeeded(RequestHeader hdr, Closure closure)

This method is invoked by the ORB if it was able to send the request.

| Parameter | Description |
|-----------|-------------|
| hdr       | The GIOP Request header associated with this request. |
| closure   | The closure object previously passed to prepare_request. |

**Closure**

public class 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 11.1 Closure class**

```java
class Closure {
    java.lang.Object object;
}
```

**ServerInterceptor**

public interface ServerInterceptor extends Object

You can create a server-side interceptor by implementing the ServerInterceptor interface. There is usually one server interceptor per client connection.

In addition, VisiBroker provides the DefaultServerInterceptor class that you can extend rather than implement. The default interceptor class offers the same methods as the ServerInterceptor interface; however, when you extend the default interceptor class, you can choose which methods to implement or override. When you use the default class, you can accept the default behavior that it provides or change it.

```java
interface ServerInterceptor {
    IOR locate(
        in unsigned long req_id,
        in OctetSequence object_key,
        in Closure closure
    );
    void locate_succeeded(
        in unsigned long req_id,
        in Closure closure
    );
    void locate_forwarded(
        in unsigned long req_id,
        inout IOR forward_iort,
        inout Closure closure
    );
}
```
IOR locate_failed(
    in unsigned long req_id,
    in OctetSequence object_key,
    in Closure closure
);
InputStream receive_request(
    in RequestHeader hdr,
    inout Object target,
    in InputStream buf,
    in Closure closure
);
void prepare_reply(
    in RequestHeader hdr,
    inout ReplyHeader reply,
    in Object target,
    in Closure closure
);
OutputStream send_reply(
    RequestHeader reqHdr,
    in ReplyHeader hdr,
    in CORBA::Object target,
    in OutputStream buf,
    in string exception_rep_id,
    in Closure closure
);
void send_reply_failed(
    in RequestHeader reqHdr,
    in ReplyHeader replyHdr,
    in Object target,
    in Environment env,
    in Closure closure
);
void request_completed(
    in RequestHeader reqHdr,
    in Object target,
    in Closure closure
);

enum ShutdownReason {
    CLIENT_ABORTED,
    SERVER_RESOURCES_EXCEEDED
};
void shutdown(
    in ServerInterceptor::ShutdownReason reason
);
void exception_occurred(
    in RequestHeader reqHdr,
    in Closure closure
);
public abstract void exception_occurred(RequestHeader reqHdr, Closure closure)

When chaining interceptors, some interceptor methods may raise a CORBA exception. When this occurs, the VisiBroker ORB does not invoke interceptor methods further down the chain. To inform interceptors that have already been invoked, the VisiBroker ORB invokes their exception_occurred methods. Implementations of this method should clean up any state they created in the previous invocation; otherwise, the VisiBroker ORB will raise a CORBA exception to the client or server rather than finish operations.

| Parameter  | Description                          |
|------------|--------------------------------------|
| reqHdr     | The request id corresponding to the request for which no reply was received. |
| closure    | The closure object previously passed to prepare_request. |

public abstract IOR locate(int req_id, byte object_key[], Closure closure)

The locate method is called whenever a LocateRequest message is received by the server. Returns an IOR to send in the LocateReply message or null if the ORB's default should be used.

| Parameter  | Description                          |
|------------|--------------------------------------|
| req_id     | The request id of the LocateRequest received by the server. |
| object_key | The object_key specified in the LocateRequest. |
| closure    | A new closure object to be used for all calls related to the processing of this locate request. |

public abstract IOR locate_failed(int req_id, byte object_key[], Closure closure)

This method is called if locate failed. Returns an IOR to return to the client or null to use the ORB's default behavior.

| Parameter  | Description                          |
|------------|--------------------------------------|
| req_id     | The request id of the LocateRequest received by the server. |
| object_key | The object_key specified in the LocateRequest. |
| closure    | The closure object originally passed to the locate call. |

public abstract void locate_forwarded(int req_id, IORHolder forward_ior, Closure closure)

This method is called if the locate method forwarded.

| Parameter  | Description                          |
|------------|--------------------------------------|
| req_id     | The request id of the LocateRequest received by the server. |
| forward_ior| An IOR indicating the forwarded object reference. |
| closure    | The closure object originally passed to the locate call. |
public abstract void locate_succeeded(int req_id, Closure closure)

This method is called if the locate method succeeded.

| Parameter | Description |
|-----------|-------------|
| req_id    | The request id of the LocateRequest received by the server. |
| closure   | The closure object originally passed to the locate call. |

public abstract void prepare_reply(RequestHeader hdr, ReplyHeaderHolder reply, Object target, Closure closure)

This method is called when the reply header is being prepared.

Note: The method request_failed can be called before prepare_reply.

public abstract InputStream receive_request(RequestHeader hdr, ObjectHolder target, InputStream buf, Closure closure)

This method is called when a request message is received from client. If return value is “not null”, that buffer is passed onto target. It also possible to change the hdr and forward the request to a different operation or to a different Object. Returns an input stream to replace the one received by the ORB or null if the ORB should use the original input stream.

| Parameter | Description |
|-----------|-------------|
| hdr       | The RequestHeader received by the server. |
| target    | An object specifying the target of the operation. The interceptor may change this value. |
| buf       | An input stream containing all of the marshalled data received from the client. |
| closure   | A new closure object to be used for all interceptor calls related to this request. |

public abstract void request_completed(RequestHeader reqHdr, Object target, Closure closure)

This method is called when the request completed. It gets called in the following cases:

- If it’s a oneway call and the request completed successfully. If the request failed exception_occurred gets called.
- If it’s a regular call, the method gets called after a reply is sent successfully to the client. If sending the reply to the client fails, send_reply_failed gets called.

public abstract OutputStream send_reply(RequestHeader reqHdr, ReplyHeader hdr, Object target, OutputStream buf, String exception_rep_id, Closure closure)

This method is called when reply is being sent. If return value is “not null,” that value is sent to the client. The env passed will contain any user exceptions, if they were thrown.

public abstract void send_reply_failed(RequestHeader reqHdr, ReplyHeader replyHdr, Object target, Environment env, Closure closure)

This method is called if send failed.
UntypedObjectWrapper

public abstract void shutdown(ShutdownReason reason)

The ORB invokes this method when the connection limit has been reached or the client connection is closed.

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 on page 11–13, that will create your UntypedObjectWrapper objects.

See the VisiBroker for Java Programmer’s Guide for complete information using object wrappers.

```java
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
    );
}
```

Methods

public void pre_method(java.lang.String operation, org.omg.CORBA.Object target,
com.visigenic.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 |
|-----------|-------------|
| operation | The name of the operation being requested. |
| target    | The object that is the target of the request. |
| closure   | 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.visigenic.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 Environment 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, described on page 11–4.

Your factory will be 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.

```java
interface UntypedObjectWrapperFactory {
    ::interceptor::UntypedObjectWrapper create(
        in Object object
    );
}
```

**Methods**

public com.visigenic.vbroker.interceptor.UntypedObjectWrapper create(org.omg.CORBA.Object object);

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 |
|-----------|-------------|
| object    | 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. |
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 12 of the OMG CORBA Specification.

**GIOP.ReplyHeader**

public class ReplyHeader

This class represents the reply header of a reply message that is sent to a client in response to a request message.

**IDL definition**

```idl
struct ReplyHeader {
    ::IOP::ServiceContextList service_context;
    unsigned long request_id;
    ::GIOP::ReplyStatusType reply_status;
};
```
**Variables**

public com.visigenic.vbroker.IOP.ServiceContext[] service_context;

A list of service context information that may be passed from the server to the client.

public int request_id;

Should be set to the same request_id as the request message for which this reply is associated.

public com.visigenic.vbroker.GIOP.ReplyStatusType reply_status;

Indicates the status of the reply and should be set to one of the values described in “GIOP.ReplyStatusType” on page 12-2.

**Constructors**

public ReplyHeader()

Creates an empty ReplyHeader.

public ReplyHeader(com.visigenic.vbroker.IOP.ServiceContext[] service_context,
int request_id, com.visigenic.vbroker.GIOP.ReplyStatusType reply_status)

Creates a ReplyHeader initialized with the specified service contexts, request identifier, and reply status.

| Parameter     | Description                                           |
|---------------|-------------------------------------------------------|
| service_context | The list of service context information that may be passed from the server to the client. |
| request_id    | The identifier of the request for which this reply is intended. |
| reply_status  | The status of the request.                           |

**GIOP.ReplyStatusType**

public class ReplyStatusType

This class contains values that are used to represent the status of a GIOP request.

**IDL definition**

```java
class ReplyStatusType {
    NO_EXCEPTION,
    USER_EXCEPTION,
    SYSTEM_EXCEPTION,
    LOCATION_FORWARD
};
```
**Methods**

```java
public int value()
```

This method returns an integral value representing the constant.

```java
public static com.visigenic.vbroker.GIOP.ReplyStatusType from_int(int value)
```

This method returns an enum instance for the value you specify. For more information about enum mapping, see “Enum” on page 3-12.

| Parameter   | Description                  |
|-------------|------------------------------|
| value       | The value for which an enum is returned. |

**GIOP.RequestHeader**

```java
public class RequestHeader
```

This class represents the request header of a request message that is sent to an object implementation.

**IDL definition**

```java
struct RequestHeader {
    ::IOP::ServiceContextList service_context;
    unsigned long request_id;
    boolean response_expected;
    ::CORBA::OctetSequence object_key;
    string operation;
    ::CORBA::OctetSequence requesting_principal;
};
```

**Variables**

```java
public com.visigenic.vbroker.IOP.ServiceContext[] service_context;
```

A list of service context information that may be passed from the server to the client.

```java
public int request_id;
```

A unique identifier used to associate a reply message with a particular request message.

```java
public boolean response_expected;
```

Set to `false` if the request is a oneway operation for which a reply is not expected. Set to `true` for operation requests and other requests that expect a reply.
Constructors

```java
public byte[] object_key;

Represents the object that is the target of the request. Object keys are stored in a vendor-specific format and are generated when an IOR is created.

```public java.lang.String operation;

Identifies the operation being requested on the target object. This member is the same as the operator member, except that it is a managed type.

```public byte[] requesting_principal;

A value that identifies the requesting principal, used to support the BOA.get_principal method, described on page 5-4.

Parameter Description
---|---
service_context The list of service context information that may be passed from the client to the server.
request_id The identifier of the request.
response_expected Set to false if this is a oneway operation request. Otherwise, set to true.
object_key Represents the object that is the target of the operation request.
operation Identifies the operation being requested.
requesting_principal The requestor's Principal.

GiOP.Version

```java
public class RequestHeader

This class represents the version of IIOP supported.

IDL definition

```java
struct Version {
    octet major;
    octet minor;
};
```
Variables

public byte major;
    The major number of the version.

public byte minor;
    The minor number of the version.

Constructors

public Version()
    Creates an empty Version.

public Version(byte major, byte minor)
    Creates a Version initialized with the specified major and minor numbers.

| Parameter | Description |
|-----------|-------------|
| major     | The major number of the version. |
| minor     | The minor number of the version. |

IIOP_1_1.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 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;
};

Variables

public com.visigenic.vbroker.GIOP.Version iiop_version;
    Represents the version of IIOP supported.
Constructors

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;

Object keys are stored in a vendor-specific format and are generated when an IOR is created.

public com.visigenic.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.

Constructors

public ProfileBody()

Creates an empty ProfileBody.

public ProfileBody(com.visigenic.vbroker.GIOP.Version iiop_version, java.lang.String host, short port, byte[] object_key, com.visigenic.vbroker.IOP.TaggedComponent[] components)

Creates a ProfileBody initialized with the specified IIOP version, host name, port number, object key, and components.

Parameter | Description
--- | ---
iiop_version | The version of IIOP supported.
host | The host where the object implemented.
port | The port number to use when establishing a connection to the object.
object_key | The object’s key.
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.

IOP.IOR

public class IOR

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 on page 5-23.

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

```c++
struct IOR {
    string type_id;
    sequence<::IOP::TaggedProfile> profiles;
};
```

Variables

```java
public java.lang.String type_id;
```

Describes the type of object reference that is represented by this IOR.

```java
public com.visigenic.vbroker.IOP.TaggedProfile[] profiles;
```

Represents a sequence of one or more TaggedProfile objects, which contain information about the protocols that are supported.

Constructors

```java
public IOR()
```

Creates an empty IOR.

```java
public IOR(java.lang.String type_id, com.visigenic.vbroker.IOP.TaggedProfile[] profiles)
```

Creates a IOR initialized with the specified type and profiles.

| Parameter | Description |
|-----------|-------------|
| type_id   | Describes the type of object reference that is represented by this IOR |
| profiles  | One or more TaggedProfile objects, which contain information about the protocols that are supported. |

IOP.ServiceContext

```java
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 classes," for more information on these classes and the methods they offer.

IDL definition

```c++
struct ServiceContext {
    ::IOP::ServiceID context_id;
    ::CORBA::OctetSequence context_data;
};
```
### Variables

**Variables**

```java
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.
```

### Constructors

**Constructors**

```java
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

| Parameter          | Description                                               |
|--------------------|-----------------------------------------------------------|
| context_id         | Identifies a particular service and data format.          |
| context_data       | Service-specific data, encapsulated as a sequence of octets |

### IOP.TaggedProfile

**IOP.TaggedProfile**

```java
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 classes,” for more information on these classes and the methods they offer.

### IDL definition

```idl
struct TaggedProfile {
    ::IOP::ProfileId tag;
    sequence<octet> profile_data;
};
```
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_MULTIPLE_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.

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_MULTIPLE_COMPONENTS                                                     |
|             | TAG_VSGN_LOCATOR                                                           |
|             | TAG_LOCAL_IPC_IOP                                                          |
| profile_data| The protocol information needed to invoke an operation on an IOR            |
Chapter 13

URL Naming interface

This chapter describes the Resolver interface and classes used in VisiBroker’s URL Naming Service.

Note

In VisiBroker for Java version 2.5, the URL Naming Service was called the Web Naming Service.

It includes the following major section:

Resolver page 13-1

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 13, “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);

};
**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 |
|-----------|-------------|
| url_s     | 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 will replace the old binding.

| Parameter | Description |
|-----------|-------------|
| url_s     | The URL's string. |
| obj       | 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 will be raised.

| Parameter | Description |
|-----------|-------------|
| url_s     | The URL's string. |
| obj       | The object whose IOR will be associated with the URL. |
Location Service interfaces

This chapter provides detailed information about the Location Service’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 15, “Discovering object instances using the Location Service,” in the VisiBroker for Java Programmer’s Guide. This chapter includes the following major sections:

- Agent page 14-1
- Desc page 14-4
- Fail page 14-6
- TriggerDesc page 14-6
- TriggerHandler page 14-7

Agent

public interface Agent extends 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.

```java
interface Agent {
    HostnameSeq all_agent_locations()
    raises(Fail);
}
```
Methods

public abstract String[] all_agent_locations()

This method retrieves a sequence of host names on which osagents reside.

This method throws the following exceptions.

| Exception | Description |
|-----------|-------------|
| Fail      | Either there is no agent available or there is unsuccessful communications with the osagent. |

public abstract Object[] all_instances(String repository_id)

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 exceptions.

| Exception | Description |
|-----------|-------------|
| Fail      | The repository id is invalid. |

public abstract Desc[] all_instances_descs(String repository_id)

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. |
Methods

This method throws the following exceptions.

| Exception  | Description                        |
|------------|------------------------------------|
| Fail       | The repository id is invalid.      |

public abstract Object[] all_replica(String repository_id, String instance_name)

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 exceptions.

| Exception  | Description                        |
|------------|------------------------------------|
| Fail       | Either the repository id or object name is invalid. |

public abstract Desc[] all_replica_descs(String repository_id, String instance_name)

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 exceptions.

| Exception  | Description                        |
|------------|------------------------------------|
| Fail       | Either the repository id or object name is invalid. |

public abstract String[] all_repository_ids()

This method retrieves all interfaces known to any osagent. This method throws the following exceptions.

| Exception  | Description                        |
|------------|------------------------------------|
| Fail       | The repository id is invalid.      |
public abstract void reg_trigger(TriggerDesc desc, TriggerHandler handler)

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 exceptions.

| Exception | Description |
|-----------|-------------|
| Fail      | There is no such trigger. |

public abstract void unreg_trigger(TriggerDesc desc, TriggerHandler handler)

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 exceptions.

| Exception | Description |
|-----------|-------------|
| Fail      | There is no such trigger. |

---

Desc

desc

public final class Desc

This class 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.

```java
struct Desc {
    CORBA::Object ref;
    ::IIOP_1_1::ProfileBody iiop_locator;
    string repository_id;
    string instance_name;
}
```
Variables

```java
public boolean activable;
public java.lang.String agent_hostname;
public com.visigenic.vbroker.IIOP_1_1.ProfileBody iiop_locator;
public java.lang.String instance_name;
public org.omg.CORBA.Object ref;
public java.lang.String repository_id;
```

**Variables**

- **activable**: A boolean indicating whether the object is registered with the Object Activation Daemon. It is set to `true` to indicate registration and `false` to indicate manual start.
- **agent_hostname**: The name of the host running the Smart Agent.
- **iiop_locator**: An `IIOP_1_1.ProfileBody` object, as described on page 12-5.
- **instance_name**: The object’s instance name.
- **ref**: A reference to the object being described.
- **repository_id**: The object’s repository identifier.

**Constructor**

```java
public Desc( org.omg.CORBA.Object ref, com.visigenic.vbroker.IIOP_1_1.ProfileBody iiop_locator,
            java.lang.String repository_id, java.lang.String instance_name, boolean activable,
            java.lang.String agent_hostname)
```

This constructor initializes a `Desc` object with the specified parameters.

| Parameter          | Description                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| ref                | A reference to the object being described.                                  |
| iiop_locator       | A reference to the object being described.                                  |
| repository_id      | The object’s repository identifier.                                         |
| instance_name      | The object’s instance name.                                                 |
| activable          | Set to `true` to indicate registration, `false` for manual start.           |
| agent_hostname     | The name of the host running the Smart Agent.                              |
Methods

public java.lang.String toString()

Returns a string containing the contents of this object.

Fail

public class Fail extends org.omg.CORBA.UserException

This exception class may be thrown by the Agent class to indicate various errors. The data member FailReason is used to indicate the nature of the failure.

Variables

com.visigenic.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

This class contains information you use to describe the characteristics of one or more objects for which you wish to register a TriggerHandler, described on page 14-7. These members may be set to null to monitor the widest possible set of objects. The more information that is specified, the smaller the resulting set of objects.

struct TriggerDesc {
    string repository_id;
    string instance_name;
    string host_name;
};

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.
Constructor

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.

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                                                                 |
|-------------|----------------------------------------------------------------------------|
| repository_id | The repository identifiers of the objects to be monitored by the TriggerHandler. May be set to null to include all possible repository identifiers. |
| instance_name | The instance name of the object to be monitored by the TriggerHandler. May be set to null to include all possible instance names. |
| hostname     | 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. |

Methods

public java.lang.String toString()

Returns a string containing the contents of this object.

TriggerHandler

public interface TriggerHandler

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 class and implementing its impl_is_ready and impl_is_down methods.

interface TriggerHandler {
    void impl_is_ready(in Desc desc);
    void impl_is_down(in Desc desc);
};
**Methods**

**public abstract void impl_is_ready(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 abstract void impl_is_down(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. |
Chapter 15

ORB Manager interfaces

This chapter describes the ORB Manager interfaces, which allow a client application to query and set the attributes of a server application. This chapter includes the following major sections:

- Adapter page 15-1
- AttributeSet page 15-3
- Server page 15-5

Adapter

public interface Adapter extends com.visigenic.vbroker.services.ORBManager.AttributeSet

This interface allows you to obtain and set the attributes for a particular type of Object Adapter. The common attributes for each type of Object Adapter are described in Table 15.2. Attributes that are specific to the TPool adapter are described Table 15.2. You obtain an Adapter object reference by invoking the Server.get_adapter method, described on page 15–7.

In addition to the methods described here, this interface inherits the methods defined for the base interface AttributeSet, described on page 15–3.

Note For client applications to use this interface, your server application must have been started with the -DORBservices=ORBManager command-line option.
Adapter attributes

Table 15.1 Common attributes for both TPool and TSession adapters

| Attribute              | Access  | Description                                                                 |
|------------------------|---------|----------------------------------------------------------------------------|
| OAactivatedConnections | Read-Only | Current number of active client connections                                 |
| OAactivatedRequests    | Read-Only | Current number of outstanding operations requests.                        |
| OAconnectionMax        | Read-Write | Maximum number of incoming connections allowed.                          |
| OAconnectionMaxIdle    | Read-Write | Number of seconds that connections are allowed to be idle before being shutdown. A value of 0 means that connections will never time-out. |

Table 15.2 Attributes for TPool adapters

| Attribute             | Access  | Description                                                                 |
|-----------------------|---------|----------------------------------------------------------------------------|
| OAallocatedThreads    | Read-Only | Current number of allocated threads.                                       |
| OAThreadMax           | Read-Write | Maximum number of threads allowed.                                         |
| OAThreadMaxIdle       | Read-Write | Number of seconds a thread can exist without servicing any requests before it is returned to the system. |
| OAThreadMin           | Read-write | Minimum number of threads allowed.                                         |

```java
interface Adapter : ::ORBManager::AttributeSet {
    typedef sequence< ::Object > ObjectSeq;
    readonly attribute string adapter_id;
    readonly attribute ::ORBManager::Adapter::ObjectSeq persistent_objects;
};
```

Methods

```java
public java.lang.String adapter_id();
```

Returns the identifier for this adapter. The returned string will contain one of the following values:

- TPool
- TSession

```java
public org.omg.CORBA.Object[] persistent_objects();
```

Returns a sequence of object references to the persistent objects offered by this adapter. You can use this method to discover all the persistent objects currently active for the adapter.
public interface AttributeSet extends org.omg.CORBA.Object

This base interface is used to derive the Server and Adapter interfaces which are, in turn, used to manage ORB and Object Adapter attributes.

```java
interface AttributeSet {
    exception InvalidAttributeId {
    };
    exception InvalidAttributeType {
    };
    exception InvalidAttributeValue {
    };
    exception AttributeReadOnly {
    };
    exception ShutdownTimeout {
    };
    any get_attribute( in string attribute_id )
    raises(::ORBManager::AttributeSet::InvalidAttributeId);
    void set_attribute( in string attribute_id, in any attribute_value)
    raises(::ORBManager::AttributeSet::InvalidAttributeId,
    ::ORBManager::AttributeSet::InvalidAttributeType,
    ::ORBManager::AttributeSet::InvalidAttributeValue,
    ::ORBManager::AttributeSet::AttributeReadOnly
    );
    struct Attribute {
        string id;
        any value;
        boolean is_readonly;
    };
    typedef sequence<::ORBManager::AttributeSet::Attribute> AttributeSeq;
    ::ORBManager::AttributeSet::AttributeSeq get_all_attributes();
};
```

See also “Adapter” on page 15-1 and “Server” on page 15-5

Methods

```java
public OBManager.AttributeSetPackage.Attribute[] get_all_attributes();
```

Returns a sequence of all of the Attribute objects available for this object. Each Attribute object contains the attribute’s identifier as a string, an Any object containing the attribute’s type and value, and an indication of whether or not the attribute is read-only or may be altered using the set_attribute method.
public org.omg.CORBA.Any get_attribute( java.lang.String attribute_id)
throws OBManager.AttributeSetPackage.InvalidAttributeId;

Returns an Any object containing the value of the attribute with the specified attribute_id.

| Parameter      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| attribute_id   | A string containing the identifier of the server attribute whose value is to be returned. The permitted values for this parameter depend on the object’s type. For a list of ORB attributes, see page 15–5. For a list of Object Adapter attributes, see page 15–2. |

This method may throw the following exceptions.

| Exception                  | Description                                           |
|----------------------------|-------------------------------------------------------|
| InvalidAttributeId         | The attribute_id is not invalid.                      |

public void set_attribute( java.lang.String attribute_id, org.omg.CORBA.Any attribute_value)
throws ORBManager.AttributeSetPackage.InvalidAttributeId,
ORBManager.AttributeSetPackage.InvalidAttributeType,
ORBManager.AttributeSetPackage.InvalidAttributeValue,
ORBManager.AttributeSetPackage.AttributeReadOnly;

Sets the value of the attribute with the specified attribute_id.

**Note** When setting an attribute, the value is always specified as a string. Boolean attribute values are specified as “true” or “false”. Integral values are specified as a string of numeric characters, such as “123”.

| Parameter      | Description                                                                 |
|----------------|-----------------------------------------------------------------------------|
| attribute_id   | A string containing the identifier of the server attribute to be set. The permitted values for this parameter depend on the object’s type. For a list of ORB attributes, see page 15–5. For a list of Object Adapter attributes, see page 15–2. |
| attribute_value| An Any object containing the value to set for the attribute. The permitted values depend on the type of attribute. For a list of ORB attribute values, see page 15–5. For a list of Object Adapter attribute values, see page 15–2. |

This method may throw the following exceptions.

| Exception                  | Description                                                                 |
|----------------------------|-----------------------------------------------------------------------------|
| InvalidAttributeId         | The attribute_id is not invalid.                                           |
| InvalidAttributeType       | The attribute_value object’s TypeCode does not match the data type of the attribute being set. |
| InvalidAttributeValue      | The attribute_value is invalid or out of range.                            |
| AttributeReadOnly          | The attribute with the specified attribute_id is read-only and cannot be set. |
public interface Server extends com.visigenic.vbroker.services.ORBManager.AttributeSet

This interface allows you to obtain and set the ORB attributes described in Table 15.3 and Table 15.4. You obtain a Server object reference by binding to an object and then invoking the Object._resolve_reference method, specifying ORBManager as the value of the id parameter.

Note: For client applications to use this interface, your server application must have been started with the -DORBservices=ORBManager command-line option.

interface Server : ::ORBManager::AttributeSet {
    readonly attribute unsigned long process_id;
    readonly attribute long activation_policy;
    ::ORBManager::Adapter get_adapter( in string oa_id);
    ::ORBManager::AdapterSeq get_all_adapters();
    void shutdown();
};

Read-only ORB attributes

Table 15.3  Read-only attributes for the Server interface

| Attribute         | Description                                                                 |
|-------------------|-----------------------------------------------------------------------------|
| ORBagentAddr      | IP address or host name of the host running the Smart Agent that the server is using. Zero will be returned if this was not specified by the server when it was started. |
| ORBagentPort      | Smart Agent port number being used by the server.                           |
| ORBbackCompat     | If “true”, the server is operating in backward compatibility mode, as described in Appendix A. |
| ORBdisableAgentCache | If “true”, Smart Agent caching is turned off.                              |
| ORBdisableLocator | If “true”, Smart Agent and Gatekeeper are disabled.                         |
| ORBgatekeeperIOR  | The URL associated with the Gatekeeper.                                    |
| ORBnullString     | If “true”, passing a null string will cause a BAD_PARAM exception to be thrown. The passing of zero length strings is allowed if ORBbackCompat is true. |
| ORBpatch          | For internal use.                                                           |
| ORBprocId         | The process identifier of the server or the vbj process used to start the server. |
| ORBsecureShutdown | If “true”, the server cannot be stopped using the shutdown command or using the ORB Manager interface. |
| ORBsyncGC         | If “true”, the server performs synchronous garbage collection.              |
### Read-write ORB attributes

**Table 15.4** Read-write attributes for the Server interface

| Attribute          | Description                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| ORBagentAddrFile   | The name of the file on the server’s host containing the IP addresses of Smart Agents on other networks.                                    |
| ORBalwaysProxy     | If “true”, clients must always connect to the server using the Gatekeeper.                                                                    |
| ORBagentNoFailOver | If “true”, the server will not automatically switch to another Smart Agent if the current one becomes unavailable.                             |
| ORBconnectionCacheMax | The maximum number of connections that this server can cache.                                                                               |
| ORBconnectionMax   | The maximum number of outgoing connections allowed by the server.                                                                           |
| ORBconnectionMaxIdle | The maximum number of seconds a connection can be idle before it will be shutdown by the server. A value of zero means the connection will never time-out. |
| ORBdebug           | If “true”, debugging is turned on for the server.                                                                                           |
| ORBdebugDir        | Specifies the directory on the server’s host where thread debugging information is written.                                                 |
| ORBdebugThreads    | If “true”, thread debugging is turned on for the server.                                                                                     |
| ORBdisableGateKeeperCallbacks | If “true”, Gatekeeper callbacks are disabled.                                                                                       |
| ORBgcTimeout       | The interval, in seconds, at which ORB garbage collection will occur on the server.                                                          |
| ORBmbufSize        | The intermediate buffer size set on the server for use by VisiBroker when processing operation requests.                                    |
| ORBtcpNoDelay      | If “true”, the server’s sockets are configured to send any data written to them immediately instead of batching data as the buffer fills.     |
| ORBtcpTimeout      | The number of milliseconds a socket will wait to send data before timing out. A value of zero means that no time-out will ever occur.         |
| ORBsendBindfalse   | If “false”, the server provides backward compatibility for bind calls and VisiBroker for Java 2.5.                                            |
| ORBservices        | The identifier of a service to be installed on the server. See Appendix A for a description of possible services?                              |
| ORBwarn            | A value of “0”, “1”, or “2”, which indicates the level of warning messages to be printed. See Appendix A for a description of each level.      |
Methods

public Manager.Adapter get_adapter(java.lang.String oa_id);

Returns the an Adapter object for the Object Adapter with the specified identifier, which can then be used to set or obtain various attributes.

| Parameter | Description |
|-----------|-------------|
| oa_id     | Identifier of the Object Adapter for which you wish to obtain a management interface. The string must contain one of the following values:—TPool —TSession |

public Manager.Adapter[] get_all_adapters();

This method returns a sequence of Adapter object for all of the Object Adapters that the server is using.

public int process_id();

Returns the process identifier belonging to the server or the vbj process used to start the server.

public void shutdown();

This method causes the server to shutdown and any object references associate with that server will no longer be valid.

Note If a client application attempts to invoke this method for a server that was started with the -DORBsecureShutdown=true option, a CORBA::NO_PERMISSION exception will be thrown.
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. It includes the following sections:

- How to set ORB and BOA options
- BOA_init() method
- BOA options
- ORB.init() method
- ORB options
- Location Service options

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,
Using vbj and starting your executable

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 0 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 Code sample A.3 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 policy 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 will return you 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.

```java
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.

```java
... 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.

| Options       | Description                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| `OAconnectionMax` `<#>` | Specifies the maximum number of allowable incoming connection to the adapter. If you do not specify, the default is unlimited.                   |
| `OAconnectionMaxIdle` `<#>` | 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. |
| `OAid` `<TPool | TSession>` | 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`. |
**ORB.init() method**

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 will ignore any arguments it does not recognize.

In Code sample A.3, the `ORB.init` is passing arguments specifying a port for the Smart Agent.

**Code sample A.3** Example using `ORB.init()` with arguments

```java
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);
    ...
}
```

---

**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. |
| OAprt <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 options

The following table summarizes the ORB.init() options.

Table A.2  ORB.init options

| Options                  | Description                                                                                                                                 |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| ORBagentAddr <hostname | 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 will be used to locate a Smart Agent. |
| ORBagentAddrFile <file_name> | Specifies a file to be used in place of the default file, agentaddr.                                                                 |
| ORBagentNoFailOver <false | Enables fail-over of this Smart Agent to another Smart Agent. If you do not specify, the default value is false.                               |
| ORBagentPort <port_number> | 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 will be used. |
| ORBalwaysProxy <false | Specifies whether or not clients must always connect using the Gatekeeper. The default is false. If set to true, ORBgatekeeperIOR must also be set.        |
| ORBalwaysTunnel <false | Specifies whether or not clients must always connect to the Gatekeeper using HTTP. The default is false. If set to true, ORBgatekeeperIOR must also be set. |
| ORBbackCompat <false | 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 ORBbackCompat true 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. |
| ORBconnectionCacheMax <#> | 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. |
| ORBconnectionMax <#> | 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. |
| 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 will never time-out. This option should be set for Internet applications. |
| ORBdebug <false | Turns debugging on.                                                                                                                        |
## ORB options

**Table A.2** ORB.init options (continued)

| Options                          | Description                                                                                                                                 |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 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.                                                                                                                                 |
| 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, ORBsGatekeeperIOR must also be set. |
| ORBdisableLocator <false | true>          | Disables the Smart Agent and Gatekeeper.                                                                                                                                 |
| ORBsGatekeeperIOR <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.                                                                     |
| 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 will be 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. |
| ORBnullString <false | true>          | NULL string throws BAD_PARAM. If passing a zero length string is desired, use one of the following: ORBbackcompat=true or ORBnullstring=true. |
| ORBsecureShutdown <false | true>          | If set to false, a user can use the shutdown command to the ORB management interface to shutdown 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. |
| ORBsendBind <false | true>          | Provides backward compatibility between the bind call and VisiBroker for Java 2.5. If you do not specify, the default value is false.  |
| 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 of not to perform synchronous garbage collection. The default value is true.  |
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 interface name to locate 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.
- Sets **ORBnullString** to true, which allows NULL strings to be passed on the wire.

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 will be initialized and will receive any command-line arguments you have specified. Command-line options for the Location Service will be processed and stripped from the argument list. All unrecognized options will be ignored.

### 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. | |
Location Service options

As with the command-line options for the BOA and ORB, the Location Service options take the form of type-value pairs.

```bash
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 12. If this option is not specified, debugging output is disabled. |
| LOCtimeout <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 will verify the existence of an object before returning an object reference to the client application. If set to 0, the Location Service will offer faster performance, but it may not return the most current information. The default value for this option is 0. |
Symbols

... ellipsis 1-3
[] brackets 1-3
| vertical bar 1-3

A

activation
  interfaces 8-1
  policies 8-3
ActivationImplDef interface 8-1
Activator
  impl_is_ready 5-4
Activator interface 8-2
Agent interface 14-1
AliasDef interface 7-2
Any class 6-2
applications
  locating osagent via vbj
  command 2-7
ARG_IN class 6-4, 14-4
ARG_INOUT class 6-5
ARG_OUT class 6-5
arguments
  -ORBshmsize 2-8
ArrayDef interface 7-3
arrays
  mapping 3-11
AttributeDef interface 7-4
AttributeDescription class 7-4
AttributeMode class 7-5

B

backward compatibility A-5, A-6
binding
  BindOptions class 5-1
  BindInterceptor 11-1, 12-1
  BindOptions class 5-1
BOA
  initialization options A-1, A-2
    OAConnectionMax A-3
    OAiipaddr A-4
    OApport A-4
    OAThreadMax A-4
BOA interface 5-2
boolean type
  mapping 3-8
buffer size
  setting A-6
C

Caffeine compiler
description 2-5
char type
  mapping 3-8
classes
  Any 6-2
  ARG_IN 6-4, 14-4
  ARG_INOUT 6-5
  ARG_OUT 6-5
  AttributeDescription 7-4
  AttributeMode 7-5
  BindOptions 5-1
  BOA 5-2
  Closure 11-8
  CompletionStatus 5-5
  ConstantDescription 7-7
  ContainedPackage.
    Description 7-10
  ContainerPackage.
    Description 7-16
  ContextList 6-5, 6-6
  DefinitionKind 7-17
  Environment 6-16
  ExceptionDescription 7-20
  ExceptionList 6-16
  FullInterface-
    Description 7-24
generated 4-1
  Helper 4-1, 4-3
  Holder 4-1, 4-6
  ImplBase 4-8
  InterfaceDescription 7-25
  InvalidName 5-8
  java.lang.Runtime 10-1
  ModuleDescription 7-27
  NamedValue 6-19
  OperationDescription 7-29
  OperationMode 7-31
  Operations 4-7
  operations 4-2
  ORB 5-13
  ParameterDescription 7-31
  ParameterMode 7-32
  PrimitiveKind 7-33
  Principal 5-25
  skeleton 4-2, 4-8
stub 4-2, 4-8
SystemException 10-2
TCKind 6-28
tie 4-2, 4-8
TypeDescription 7-38
UnionMember 7-40
UnknownUser-
  Exception 6-32
UserException 10-3
client
  event handlers 11-8
  locating osagent via vbj
    command 2-7
  property list 5-6
ClientEventHandler
  interface 9-1
ClientInterceptor 11-5
Closure 11-8
commands
  idl2ir 2-1
  idl2java 2-2
  java2idl 2-5
  java2iiop 2-5
  options 1-3
  vbj 2-6
CompletionStatus class 5-5
connections
  idle time A-3, A-5
  maximum number A-3, A-5
ConstantDef interface 7-6
ConstantDescription class 7-7
constants
  mapping 3-10
constructed types
  mapping 3-11
Contained interface 7-8
ContainedPackage.Description
  class 7-10
Container interface 7-11
ContainerPackage.Description
  class 7-16
Context interface 5-6
ContextList class 6-5, 6-6
conventions
  platform icons 1-3
  syntax 1-3
  typographic 1-2
CreationImplDef interface 8-3
D
DefinitionKind class 7-17
delegation
  with server
    implementations 3-21
DII
generating portable
  stubs 2-6
documentation
  more info on this subject 1-3

E
EnumDef interface 7-18
eenums
  mapping 3-11
Environment class 6-16
  environment variables 8-5
EstructDef interface 7-18
  event handlers
    for clients 11-8
    interfaces 9-1
ExceptionDef interface 7-19
  ExceptionDescription class 7-20
  ExceptionList class 6-16
  exceptions
    class hierarchy 10-1
    mapping 3-22
    extract method
      in Helper classes 4-3

F
floating point
  mapping 3-9
FullInterfaceDescription
class 7-24

G
generated classes 4-1

H
HandlerRegistry interface 9-3
  Helper classes 4-3
    mapping 3-9
    overview 4-1
  Holder classes 4-6
    mapping 3-5
    overview 4-1
  hostname A-4
  interfaces
    ActivationImplDef 8-1
    Activator 8-2
    Agent 14-1
    AliasDef 7-2
    ArrayDef 7-3
    AttributeDef 7-4
    BindInterceptor 11-1, 12-1
    BOA 5-2
    ClientEventHandler 9-1
    ClientInterceptor 11-5
    ConstantDef 7-6
    Contained 7-8
    Container 7-11
    Context 5-6
    CreationImplDef 8-3
    EnumDef 7-18
    EstructDef 7-18
    ExceptionDef 7-19
    HandlerRegistry 9-3
    IDLType 7-21
    ImplEventHandler 9-6
    InputStream 6-17
    InterfaceDef 7-22
    mapping 3-18, 3-21
    ModuleDef 7-27
    NVList 6-20
    OAD 8-8
    Object 5-8
    OperationDef 7-28
    OutputStream 6-22
    PrimitiveDef 7-33
    Repository 7-34
    Request 6-23
    Resolver 13-1
    SequenceDef 7-36
    ServerInterceptor 11-8
    ServerRequest 6-27
    StringIO 7-37
    StructDef 7-37
    TriggerHandler 14-7
    TypeCode 6-29
    TypedefDef 7-38
    UnionDef 7-39
    VersionSpec 7-41
    WstringDef 7-41

Inter-operable Object Reference
(see IOR) 12-6
InvalidName class 5-8
IOR
  associating with a URL 13-2
  IP address A-4, A-5
  IRObjec 7-26
Java
creating an IDL file from Java 2-5
defining IDL interfaces 2-5
mapping from IDL 3-1
starting the interpreter via vbj 2-6
Java code
generating from IDL file 2-2
java2idl
command info 2-5
description 2-5
java2iop
command info 2-5
generating portable stubs for DII 2-6

Location Service
query a Smart Agent 14-1, 15-1
registering triggers 14-1, 15-1
TriggerDesc 14-6

mapping
arrays 3-11
boolean type 3-8
char type 3-8
constants 3-10
constructed types 3-11
enums 3-11
exceptions 3-22
floating point 3-9
Helper classes 3-9
Holder classes 3-5
IDL names 3-2
IDL types 3-4
integer 3-9
interface scope 3-22
interfaces 3-18
modules 3-3
nested types 3-24
octet 3-8
passing parameters 3-19
reserved names 3-2
reserved words 3-3
sequences 3-11
string 3-9
structs 3-11
typedefs 3-25
unions 3-11
messages
broadcasting via vbj command 2-7
methods 4-6
_bind_options in Object 5-9
_boa in Object 5-10
_clone in Object 5-10
_create_request in
Object 5-10
_get_interface in Object 5-10
_hash in Object 5-11
_is_a in Object 5-11
_is_bound in Object 5-11
_is_equivalent in Object 5-11
_is_local in Object 5-12
_is_persistent in Object 5-12
_non_existent in Object 5-12
_object_name in Object 5-12
_principal in Object 5-12
_repository_id in Object 5-12
_request in Object 5-13
_value in AttributeMode 7-6
_value in OperationMode 7-31
_value in ParameterMode 7-33
_absolute_name in
Contained 7-9
activate in Activator 8-2
activation_policy in
CreationImplDef 8-6
activator_obj in
ActivationImplDef 8-1
add in ContextList 6-6
add in ExceptionList 6-17
add in NVList 6-21
add_in_arg in Request 6-24
add_inout_arg in
Request 6-24
add_item in NVList 6-21
add_named_in_arg in
Request 6-24
add_named_inout_arg in
Request 6-24
add_out_arg in Request 6-25
add_value in NVList 6-21
addClassObjectWrapperClass in Helper class 4-5
addServerObjectWrapperClass in Helper class 4-5
all_agent_locations in
Agent 14-2
all_instances in Agent 14-2
all_instances_descs in
Agent 14-2
all_replica in Agent 14-3
all_replica_descs in
Agent 14-3
all_repository_ids in
Agent 14-3
args in CreationImplDef 8-6
arguments in Request 6-25
base in EstructDef 7-19
base_def in EstructDef 7-19
base_interfaces in
InterfaceDef 7-23
bind in BindInterceptor 11-2
bind in
ImplEventHandler 9-7
bind in
ORB 5-15
bind_failed in
BindInterceptor 11-3
bind_failed in
ClientInterceptor 9-2
bind_succeeded in
BindInterceptor 11-3, 11-4
bind_succeeded in
ClientInterceptor 9-2
BindOptions in
BindOptions 5-2
BOA_init in ORB 5-15
bound in SequenceDef 7-36
bound in StringDef 7-37
bound in WstringDef 7-41
change_implementation in
OAD 8-9
clear in Environment 6-16
client_aborted in
ImplEventHandler 9-7
completion status
methods 5-5
containing_repository in
Contained 7-9
content_type in
TypeCode 6-29
contents in Container 7-12
context_name in Context 5-6
dictories in
OperationDef 7-28
dictories in Request 6-25
count in ContextList 6-6
count in ExceptionList 6-6
count in NVList 6-21
create_alias in
  Container 7-13
create_alias_tc in ORB 5-16
create_any in ORB 5-16
create_array in
  Repository 7-35
create_array_tc in ORB 5-16
create_child in Context 5-6
create_constant in
  Container 7-13
create_context_list in ORB 5-16
create_enum in
  Container 7-13
create_enum_tc in ORB 5-18
create_environment in
  ORB 5-19
create_estruct in
  Container 7-13
create_estruct_tc in ORB 5-19
create_exception in
  Container 7-14
create_exception_tc in ORB 5-19
create_input_stream in
  Any 6-2
create_input_stream in ORB 5-19
create_interface in
  Container 7-14
create_interface_tc in ORB 5-19
create_list in ORB 5-20
create_module in
  Container 7-14
create_named_value in ORB 5-20
create_operation_list in ORB 5-20
create_output_stream in
  Any 6-2
create_output_stream in ORB 5-20
create_recursive_sequence_tc in ORB 5-20
create_sequence in
  Repository 7-35
create_sequence_tc in ORB 5-21
create_string in
  Repository 7-35
create_string_tc in ORB 5-21
create_struct in
  Container 7-14
create_struct_tc in ORB 5-21
create_union in
  Container 7-15
create_union_tc in ORB 5-21
create_wstring in
  Repository 7-35
create_wstring_tc in
  ORB 5-21
ctx in Request 6-25
ctx in ServerRequest 6-27
deactivate in Activator 8-2
deactivate_obj in BOA 5-4
def_kind in IObject 7-26
default_bind_options in
  ORB 5-22
default_index in
  TypeCode 6-30
default_principal in
  ORB 5-22
defined_in in Contained 7-9
delete_values in Context 5-7
describe in Contained 7-9
describe_contexts in
  Container 7-15
describe_interface in
  InterfaceDef 7-23
destroy in IObject 7-26
destroy_on_unregister in
  OAD 8-10
discriminator_type in
  TypeCode 6-30
discriminator_type in
  UnionDef 7-39
discriminator_type_def in
  UnionDef 7-39
element_type in
  ArrayDef 7-3
element_type in
  SequenceDef 7-36
element_type_def in
  ArrayDef 7-3, 7-4
element_type_def in
  SequenceDef 7-36
eval in CreationImplDef 8-6
eval in Request 6-25
equal in Any 6-2
equal in TypeCode 6-30
except in ServerRequest 6-27
exception in
  Environment 6-16
exception_occurred in
  BindInterceptor 11-3
exception_occurred in
  ClientInterceptor 11-6
exception_occurred in
  ServerInterceptor 11-10
exceptions in
  OperationDef 7-28
exceptions in Request 6-25
extract in Helper 4-3
extraction methods in
  Any 6-3, 6-8
flags in NamedValue 6-19
force_register_url in
  Resolver 13-2
from_int in TCKind 6-28, 12-3
get_default_context in
  ORB 5-22
get_implementation in
  OAD 8-10
get_next_response in
  ORB 5-22
generic in
  Repository 7-35
generic in ORB 5-23
get_principal in BOA 5-4
get_response in Request 6-25
get_status in OAD 8-10
get_status_all in OAD 8-10
get_status_interface in
  OAD 8-10
get_values in Context 5-7
getStubClass in Helper 4-4, 4-5
Holder in Holder 4-7
id in ActivationImplDef 8-2
id in Contained 7-9
id in CreationImplDef 8-6
id in Helper 4-3
id in TypeCode 6-30
impl_is_down in
  TriggerHandler 14-8
impl_is_ready in BOA 5-4
impl_is_ready in
  TriggerHandler 14-8
init in ORB 5-23
insert in Helper 4-3
insertion methods in
  Any 6-4, 6-9
invoke in DynamicImplementation 6-15
invoke in Request 6-26
is_a in InterfaceDef 7-24
item in ContextList 6-6
item in ExceptionList 6-17
item in NVList 6-22
kind in PrimitiveDef 7-33
kind in TypeCode 6-30
unbind in
  ImplEventHandler 9-7
unreg_glob_client_handler in
  HandlerRegistry 9-5
unreg_glob_impl_handler in
  HandlerRegistry 9-5
unreg_implementation in
  OAD 8-11
unreg_interface in OAD 8-11
unreg_obj_client_handler in
  HandlerRegistry 9-5
unreg_obj_impl_handler in
  HandlerRegistry 9-6
unreg_trigger in Agent 14-4
unregister_all in OAD 8-12
value in ConstantDef 7-6
value in DefinitionKind 7-17
value in NamedValue 6-19
value in PrimitiveKind 7-34
value in TCKind 6-28, 12-3
write in Helper 4-4
write methods in
  OutputStream 6-22
write_estruct in
  OutputStream 6-23
write_value in Any 6-3
ModuleDef interface 7-27
ModuleDescription class 7-27
modules
  mapping 3-3

N
NamedValue class 6-19
nested types
  mapping 3-24
NVList interface 6-20

O
OACollectionMax
  ORB initialization
    option A-3
OACollectionMaxIdle
  ORB initialization
    option A-3, A-5
OAD interface 8-8
OAId
  BOA initialization
    option A-3
OAipaddr A-4
OAport
  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 interface 5-8
object server
  activation policies 8-3
Object Wrappers
  generated interfaces 4-5
  object wrappers
    adding typed object
      wrappers for clients 4-5
    adding typed object
      wrappers for servers 4-5
    removing typed object
      wrappers for clients 4-6
    removing typed object
      wrappers for servers 4-6
  objects
    activating 5-2
    deactivating 5-2
octet
  mapping 3-8
OperationDef interface 7-28
OperationDescription class 7-29
OperationMode class 7-31
operations
  reporting status 5-5
operations classes
  description 4-2
options
  when using commands 1-3
ORB
  backward compatibility A-5,
    A-6
  initialization options A-1,
    A-4
  registering event
    handlers 11-8
ORB class 5-13
ORB_init() method
  -ORBshmsize 2-8
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-5, A-6
ORBdisableLocator
  ORB initialization
    option A-6
ORBgatekeeperIOR
  ORB initialization
    option A-6
ORBmbufSize A-6
ORBmbufsize
  ORB initialization
    option A-6
ORBnullstring
  ORB initialization
    option A-6
ORBservers
  ORB initialization
    option A-6
ORBtcpTimeout
  ORB initialization
    option A-7
osagent
  locating via vbj
    command 2-7
OutputStream interface 6-22

P
ParameterDescription class 7-31
ParameterMode class 7-32
parameters
  mapping 3-19
platform designation with
  icons 1-3
port number A-4, A-5
preprocessing
  using idl2ir tool 2-2
  using idl2java compiler 2-4
PrimitiveDef interface 7-33
PrimitiveKind class 7-33
Principal class 5-25
Index

R
removeClientObjectWrapper-Class in Helper class 4-6
Repository interface 7-34
Request interface 6-23
RequestHeader 12-3, 12-4, 12-5
reserved names
 mapping 3-2
reserved words
 mapping 3-3
Resolver interface 13-1

S
SequenceDef interface 7-36
sequences
 mapping 3-11
ServerInterceptor 11-8
ServerRequest interface 6-27
servers
 using inheritance 3-20
skeleton classes 4-2
smart stubs 4-4, 4-5
sockets
 batching requests A-7
string
 mapping 3-9
StringDef interface 7-37
StructDef interface 7-37
structs
 mapping 3-11
structure
GIOP::RequestHeader 12-3, 12-4, 12-5
TriggerDesc 14-6
stubs 4-4, 4-5
classes 4-2
generating portable for
DII 2-2, 2-6
syntax 1-3
syntax conventions 1-3
SystemException class 10-2
T
TCKind class 6-28
threading A-4
 idle time A-4
 thread policy A-3
tie classes 4-2
tools
 idl2ir 2-1
 idl2java 2-2
 Java2idl 2-5
 java2iiop 2-5
 options 1-3
 syntax 1-3
 vbj 2-6
TriggerDesc 14-6
TriggerHandler interface 14-7
TSession A-4
type extensions 3-4
TypeCode interface 6-29
TypedefDef interface 7-38
typedef
 mapping 3-25
TypeDescription class 7-38
types
 mapping 3-4
typographic conventions 1-2

U
UnionDef interface 7-39
UnionMember class 7-40
unions
 mapping 3-11
UnknownUserException class 6-32
URL associating with IOR 13-2
URL Naming
 Resolver interface 13-1
UserException class 10-3

V
vbj command
 description 2-6
version of product 2-1, 2-2, 2-5, 2-6
VersionSpec interface 7-41

W
web naming
 Resolver interface 13-1
words
 reserved 3-3
wstring
 mapping 3-9
WstringDef interface 7-41
