

OpenFusion® CORBA Services Version 5 Naming Service



OpenFusion CORBA Services

NAMING SERVICE GUIDE



Part Number: OFCOR-NAMG-5

Doc Issue 20, 16 September 2012

Copyright Notice

© 2012 PrismTech Limited. All rights reserved.

This document may be reproduced in whole but not in part.

The information contained in this document is subject to change without notice and is made available in good faith without liability on the part of PrismTech Limited or PrismTech Corporation.

All trademarks acknowledged.



CONTENTS

Table of Contents

List of Figures	ix
------------------------	-----------

Preface

About the Naming Service Guide	xi
Contacts	xii

The Naming Service

Chapter 1	Description	2
	1.1 Overview	2
	1.1.1 OMG Standard Features	2
	1.1.2 OpenFusion Enhancements	3
	1.2 Concepts and Architecture	3
	1.2.1 OMG Standard	3
	1.2.1.1 Naming Contexts	3
	1.2.1.2 Federation	5
	1.2.1.3 Name Components	5
	1.2.1.4 Interoperable Naming Service (INS)	6
	1.2.1.5 Stringified Names	6
	1.2.2 OpenFusion Enhancements	8
	1.2.2.1 Java Naming and Directory Interface (JNDI)	8
	1.2.2.2 Multiple Forms of Persistence	9
	1.2.2.3 Caching	9
	1.2.2.4 Purging and Memory Management	10
	1.2.2.5 Load Balancing Concepts	11
	1.2.2.6 Load Balancing in OpenFusion	11
	1.2.2.7 Instrumentation	13
	1.2.2.8 Fail-over	14
	1.2.2.9 Replication	14
Chapter 2	Using Specific Features	16
	2.1 Obtaining the Root Context	17
	2.2 Naming Context Creation and Destruction	17
	2.3 Binding and Unbinding Operations	18
	2.4 Accessing Naming Context Contents	19
	2.5 BindingIterator Operations	20
	2.6 Naming Context Extension Operations	21
	2.7 Using the LoadBalancingFactory	23
	2.8 Manipulating Objects in the LoadBalancer	24

	2.9 Using the LoadBalancer with the Naming Service	24
	2.10 Customizing the LoadBalancer	25
Chapter 3	Worked Example	28
	3.1 Example Client	28
Chapter 4	API Definitions	30
	4.1 OMG Standard API Definitions	30
	4.1.1 NamingContext Interface	30
	4.1.2 NamingContextExt Interface	31
	4.1.3 BindingIterator Interface	32
	4.2 OpenFusion API Extensions	32
	4.2.1 LoadBalancingFactory Interface	32
	4.2.2 LoadBalancer Interface	32
	4.2.3 LoadBalancer Standard Policies	34
	4.2.4 LoadBalancerPlugin Interface	35
	4.2.5 JNDIObject Interface	35
Chapter 5	Supplemental Information	38
	5.1 Administration Properties and Instrumentation	38
	5.2 Java Naming & Directory Interface (JNDI)	38
	5.3 Lightweight Directory Access Protocol (LDAP)	39
	5.4 Purging Options	39
	5.5 Memory Management	40
	5.6 XML Export and Import	41
	5.6.1 Exporting and Importing Cyclics	42
	5.7 Exceptions	44
 Java Naming and Directory Interface		
Chapter 6	Description	48
	6.1 Overview	48
	6.1.1 Sun's JNDI Standard Features	48
	6.1.2 OpenFusion Enhancements	48
	6.2 Concepts and Architecture	49
	6.2.1 Standard JNDI	49
	6.2.2 The Initial Context	50
	6.2.3 Naming Systems	50
	6.2.4 References and Addresses	50
Chapter 7	OpenFusion SPI Implementation	52
	7.1 Names	52
	7.2 Java Objects	53

	7.3 Supplied Factories	54
	7.3.1 Storing CORBA Objects	54
	7.3.2 Storing RMI-IIOP Objects	54
	7.4 Federation	54
Chapter 8	Using Specific Features	56
	8.1 JDBC-based Persistence	56
	8.2 Accessing Data	57
Chapter 9	Supplemental Information	58
	9.1 Configuration Properties	58
	9.1.1 Standard Properties	58
	9.1.2 Provider-specific Properties	59
	9.1.2.1 General	59
	9.1.2.2 Persistence	59
	9.2 Exceptions	62
 Configuration and Management		
Chapter 10	Naming Service Configuration	66
	10.1 Common Properties	66
	10.2 NameSingleton Configuration	66
	10.2.1 CORBA PropertiesOR Name Service Entry	66
	10.2.2 Lightweight Directory Access Protocol (LDAP)	68
	10.2.3 Persistence Options	70
	10.2.4 Instrumentation Properties	73
	10.2.5 General Properties	75
	10.3 LoadBalancingFactorySingleton Configuration	79
Chapter 11	Naming Service Manager	82
	11.1 Running the Naming Service Manager	82
	11.2 Using the Naming Service Manager	82
	11.2.1 Object Icons	84
	11.2.2 Tool Bar Buttons	84
	11.2.3 Adding a Naming Context	85
	11.2.4 Binding OpenFusion Services	85
	11.2.5 Binding Objects	86
	11.2.6 Deleting a Naming Context or Object Binding	86
	11.2.7 Exporting XML	86
	11.2.8 Importing XML	87
	11.2.9 Launching Managers and Browsers	87
	11.2.9.1 CORBA Object Browser	88
	11.2.9.2 Naming Service Manager	88

Chapter 12	The Purgable Interface	90
	<i>12.1</i> Purge Class Plugin	90
	<i>12.2</i> Using the Purgable Interface.....	90
Appendix A	Command Line Management Tool	94
	Features	94
	Configuration.....	95
	Index	104

List of Figures

- Figure 1 Simple Naming Graph4**
- Figure 2 Load Balancing13**
- Figure 3 LoadBalancerPlugin Interface35**
- Figure 4 Naming Hierarchy Export and re-Import43**
- Figure 5 JNDI Architecture49**
- Figure 6 OFNamingConverter Interface53**
- Figure 7 Naming Service Manager83**
- Figure 8: Example Domains Hierarchy and Directories94**

List of Figures

Preface

About the Naming Service Guide

The *Naming Service Guide* is included with the OpenFusion CORBA Services' *Documentation Set*. The *Naming Service Guide* explains how to use the OpenFusion Naming Service.

The *Naming Service Guide* is intended to be used with the *System Guide* and other OpenFusion CORBA Services documents included with the product distribution; refer to the *Product Guide* for a complete list of documents.

Intended Audience

The *Naming Service Guide* is intended to be used by users and developers who wish to integrate the OpenFusion CORBA Services into products which comply with OMG or J2EE standards for object services. Readers who use this guide should have a good understanding of the relevant programming languages (e.g. Java, IDL) and of the relevant underlying technologies (e.g. J2EE, CORBA).

Organisation

The *Naming Service Guide* is organised into three main sections. The first two sections describe the OpenFusion Naming Service and JNDI, respectively. These sections provide

- a high level description and list of main features
- explanation of the architecture and concepts
- how to use specific features
- detailed explanations of the main interfaces and how to use them
- other information which is needed to use the component

The last section, *Configuration and Management*, provides information on configuring and managing the OpenFusion Naming Service using the OpenFusion Graphical Tools. This section includes detailed descriptions of properties specific to the service, plus instructions on using the OpenFusion Graphical Tools' Browsers and Managers. This section should be read in conjunction with the *System Guide*.

Conventions

The conventions listed below are used to guide and assist the reader in understanding the Naming Service Guide.



Item of special significance or where caution needs to be taken.



Item contains helpful hint or special information.



Information applies to Windows (e.g. NT, 2000, XP) only.



Information applies to Unix based systems (e.g. Solaris) only.

Hypertext links are shown as *blue italic underlined*.

On-Line (PDF) versions of this document: Cross-references, e.g. ‘see *Contacts* on page xii’, act as hypertext links; click on the reference to go to the item.

```
% Commands or input which the user enters on the
command line of their computer terminal
```

Courier fonts indicate programming code and file names.

Extended code fragments are shown in shaded boxes:

```
NameComponent newName[] = new NameComponent[1];

// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example", "");
```

Italics and ***Italic Bold*** are used to indicate new terms, or emphasise an item.

Arial Bold is used to indicate user related actions, e.g. **File | Save** from a menu.

Step 1: One of several steps required to complete a task.

Contacts

PrismTech can be reached at the following contact points for information and technical support.

USA Corporate Headquarters

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA

Tel: +1 781 569 5819

Web:

<http://www.prismtech.com>

Technical questions: crc@prismtech.com (Customer Response Center)

Sales enquiries: sales@prismtech.com

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900

Fax: +44 (0)191 497 9901

A close-up, low-angle view of a computer keyboard, showing several keys in detail. The keys are white and set against a dark background. A white grid pattern is overlaid on the entire image, creating a sense of structure and connectivity. The lighting is soft, highlighting the texture of the keys.

THE NAMING SERVICE

CHAPTER

1 Description

The OpenFusion Naming Service and OpenFusion JNDI are part of a range of services and interfaces included with the OpenFusion CORBA Services product. The OpenFusion Naming Service can be used stand-alone or with other OpenFusion CORBA Services' interfaces and services.

The OpenFusion Naming Service and OpenFusion JNDI are standards based and fully compliant with recognised industry standards and specifications, supporting portability and interoperability.

The OpenFusion Naming Service provides a straightforward way of finding and using objects, by associating meaningful names with them. The Naming Service can then be used like a white pages telephone directory to find an object and obtain its Object Reference, without complex programming or using proprietary ORB mechanisms.

The Naming Service can also be used in any CORBA-compliant distributed-object system to create and maintain a directory of other services.

1.1 Overview

1.1.1 OMG Standard Features

The OpenFusion Naming Service is wholly compliant with the OMG specification. The basic features of the OMG specification include the ability to:

- give meaningful names to objects (*name bindings*)
- find names which have been bound to objects (*resolve*)
- group names in logical hierarchies (*naming contexts*)
- group distributed naming hierarchies (*federation*)
- retrieve lists of names and step through them (*iteration*)

The OMG also specifies an Interoperable Naming Service (INS), which extends the Naming Service to add interoperability and portability across ORBs and applications. Features of the INS include:

- a way to find and use a common initial naming context
- support for URL-style names

1.1.2 OpenFusion Enhancements

The OpenFusion implementation of the Naming Service includes several enhancements. This extended service is layered on top of the OMG-defined Naming Service and INS, and does not affect the use of these standard services.

Enhancements include:

- multiple forms of persistence
- caching
- purging and memory management
- load balancing
- additional instrumentation (service monitoring functions)

1.2 Concepts and Architecture

1.2.1 OMG Standard

The Naming Service associates meaningful names with objects. An association between a name and an object's *Interoperable Object Reference* (IOR) is called a *binding* or *name binding*.

Name bindings are grouped in hierarchies called *naming contexts*. A naming context is an object containing zero or more name bindings. Each name binding within a naming context refers to either another naming context or a CORBA object.

There is no limit to the number of different names that can be bound to the same object or naming context, or to the number of bindings that a naming context can contain.

Resolving a name is the process of locating an object or naming context by reading a name binding and retrieving the associated object reference.

Iteration is the process of retrieving a list of bindings from a naming context, and looking at each binding in turn.

1.2.1.1 Naming Contexts

A naming context is a set of name bindings where each name is unique within that context; the same name may, however, appear in other naming contexts. Naming contexts can be bound to other naming contexts to create naming hierarchies.

A very simple hierarchy of naming contexts is shown in *Figure 1*. It illustrates the fact that a given binding within a naming context can point to either an object or another naming context, and that a single object can be referenced by more than one name. These hierarchies are known as *naming graphs*.

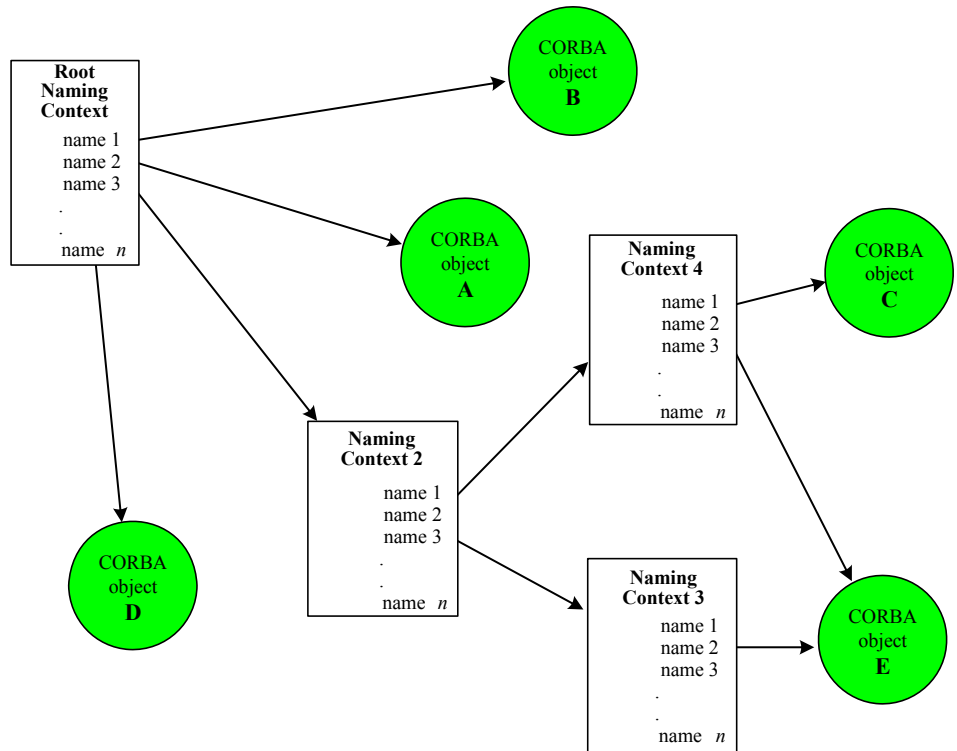


Figure 1 Simple Naming Graph

An object is referenced using an *initial* naming context, which is also referred to as the *root* context. This is followed by a sequence of one or more *name components*. Such a sequence is known as a *compound name*. Each name component resolves to the next naming context in a chain until the last name component resolves to the required object. In *Figure 1*, objects A, B and D are bound directly to the root context, so their names have only one component (these are *simple names*); objects C and E have names with three components. The full compound name for object C can be represented like this:

NamingContext2/NamingContext4/ObjectC

Object E can be accessed via two different names.

The service specification also permits a naming context to contain a binding which refers to a parent or grandparent further up the graph. For example, in *Figure 1* Naming Context 4 could contain a binding to Naming Context 2. This kind of reference is sometimes referred to as *cyclic*.

The root context is always implicit in a compound name; a special operation, `resolve_initial_references`, is performed once to obtain the root context, and all subsequent `resolve` operations depend on that.

Although it is not a requirement of the service specification, it is convenient and customary to have a single root naming context.

1.2.1.2 Federation

The OpenFusion Naming Service has the ability to link many distributed naming systems in a naming graph so that they appear as a single *namespace*. This is known as *federation*, and it enables large heterogeneous systems of names and naming contexts to be implemented. Clients using the Naming Service do not need to be aware of the physical location of a server, or of the way in which it is implemented; the link from a naming context to an object can cross several different ORBs running on different systems.

1.2.1.3 Name Components

Each name component has *id* and *kind* fields (sometimes referred to as *attributes*), represented by IDL strings. These strings are composed of ISO Latin-1 characters (excluding the ASCII *NUL*, 00h) and the combined length can be up to 255 characters.

The Naming Service always matches names using both fields, so it is acceptable for either field to be zero-length or to contain an empty string provided that uniqueness within a naming context is maintained. *Table 1* shows valid combinations of *id* and *kind* values.

Table 1 Name Component Fields

Id	Kind
name1	<empty>
name2	kind1
<empty>	<empty>
<empty>	kind2

Note that although it is technically possible for both fields to contain empty strings, this is not normally recommended, as it can be confusing to resolve to an empty name.

1.2.1.4 Interoperable Naming Service (INS)

The Interoperable Naming Service extends the basic Naming Service. It implements the `NamingContextExt` interface, which is derived from the standard `NamingContext` interface. This interface introduces an interoperable stringified form of the `CosNaming::Name` and other URL formats in order to facilitate the interpretation of object references.

1.2.1.5 Stringified Names

Names are sequences of name components, which are not human-readable and can be difficult for applications to deal conveniently with. A syntax for stringified names is therefore defined, and operations are provided to convert a name in sequence form to its equivalent stringified form and vice versa.

A stringified name has components separated by forward slashes; the id and kind fields within each component are separated by dots. The dot is omitted when the kind field is empty unless the id field is also empty, in which case the name component is comprised of a single dot. Similarly, if there is no dot in a stringified name component, then that component is taken to be an id field only (the associated kind field is empty).

For example, the stringified name `'name1/name2.kind1/./.kind2'` contains all the valid field combinations shown in *Table 1, Name Component Fields*.

A backslash must be used as an escape character if it is necessary for a name to contain a slash, backslash or dot.

1.2.1.5.1 Interoperable Object Reference (IOR)

A CORBA object is uniquely identified by its Interoperable Object Reference (IOR). The IOR is the CORBA 2.x compliant format for a standard representation of an object reference for all ORB vendors.

1.2.1.5.2 URLs

The exchange of IORs through non-electronic means is difficult because of their length and the way that binary information is encoded. The `corbaloc` URL scheme provides URLs that are familiar to people and are similar to FTP or HTTP URLs. A `corbaname` URL is similar to a `corbaloc` URL except that a `corbaname` URL also contains a stringified name that identifies a binding in a naming context. The `corbaloc` and `corbaname` schemes allow service addresses to be exchanged more easily throughout organizations. These schemes are also used to allow arbitrary object references to be specified for an initial service, although some ORBs do not

currently support these bootstrapping mechanisms. For example, the following line of code shows the OpenFusion Notification Service being referenced with a `corbaloc` URL:

```
-ORBInitRef
NotificationService=corbaloc::server.prismsystems.com/NotificationService
```

The available URL formats are: IOR, Corbaloc, Corbaname, file, FTP and HTTP.

IOR

The string form of an IOR (`IOR:<hex_octets>`) is a valid URL. The IOR URL is robust and insulates the client from the encapsulated transport information and object key used to reference the object. This URL format is independent of the Naming Service.

Corbaloc

The `corbaloc` URL scheme provides stringified object references that are more easily manipulated than IORs. This URL format is independent of the Naming Service.

A `corbaloc` URL contains:

- one or more protocol identifiers
- protocol-specific components

There are currently two protocols defined: Internet Inter-ORB protocol (IIOP) and `resolve_initial_references` (RIR). The RIR scheme allows for access to the ORB's configured initial references. The IIOP scheme is defined for use in TCP/IP and DNS centric environments such as the Internet. This protocol contains:

- one or more address(es) with an optional IIOP version number and an optional port
- an object key

For example:

```
corbaloc::10.1.1.123:14005/%00PMC%00%00%00%04%00%00%00%252cb9b780-7
803-11d3-a8ae-fef54d18874b%00%00%00%00%00%00%00%10-%9Er0x%03%11%D3%
A8%AE%FE%F5M%18%87K
```

This means that at host `10.1.1.123`, on port `14005`, it is possible to resolve the object reference denoted by the key. The key has been escaped to map away from octet values that cannot be directly part of a URL.

```
corbaloc::1.1@server.prismsystems.com,10.1.1.123:14005/%00PMC%
00%00%00%04%00%00%00%252cb9b780-7803-11d3-a8ae-fef54d18874b%00%00%0
0%00%00%00%00%10-%9Er0x%03%11%D3%A8%AE%FE%F5M%18%87K
```

This means that, at host `server.prismtechnologies.com` (using IIOP version 1.1) or, at the host denoted by the IP address `10.1.1.123` on port `14005` the key can be resolved as described above.

i

Port `2809` is used if a port is not specified.

Corbaname

A corbaname URL is similar to a corbaloc URL. However, a corbaname URL also contains a stringified name that identifies a binding in a naming context. For example:

```
corbaname::server.prismtechnologies.com/%00PMC%06%00%04%00%00#a/string/path/to/obj
corbaname:rir:#a/string/path/to/obj
```

The first URL specifies that an object (of type `NamingContext`) can be found at host `server.prismtechnologies.com` using the object key `00PMC%06%00%04%00%00`. The second URL uses the resolve initial references syntax to return a reference to a `NamingContext`. The stringified name `a/string/path/to/obj` is then used as the argument to a `resolve` operation on that `NamingContext`. The URL denotes the object reference that results from that lookup.

file

The file format (`file://`) should specify a file containing either a URL or an IOR.

FTP

The FTP format (`ftp://`) should, as above, specify a file containing a URL or an IOR. However, in this case, the file should be accessible via `ftp`.

HTTP

This format (`http://`) should specify an HTTP URL that returns an object URL or an IOR.

1.2.2 OpenFusion Enhancements

The OpenFusion Naming Service is implemented in Java for platform independence.

1.2.2.1 Java Naming and Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) is a generic API for accessing naming and directory services; the OpenFusion Naming Service is layered on top of JNDI. This enables it to access the OpenFusion service provider and also the Sun

Lightweight Directory Access Protocol (LDAP) provider. Clients may access either transparently, or use the OpenFusion JNDI SPI independently of the CORBA service.

The OpenFusion JNDI implementation is described in the *JNDI Guide*.

Please refer to the relevant Sun Microsystems documentation for details of JNDI and LDAP standard functionality.

1.2.2.2 Multiple Forms of Persistence

The OpenFusion Naming Service has been layered on top of the Java Naming and Directory Interface (JNDI). This enables it to store its persistent data in memory or databases. It can also utilize Sun's JNDI Lightweight Directory Access Protocol (LDAP) provider, using standard LDAP authentication mechanisms.

Persistent data in memory is provided by configured the JDBC hsqldb data source to perform memory based persistence.

Database persistence is implemented using Java Database Connectivity (JDBC). OpenFusion currently supports Oracle, Sybase and Informix on both Unix and Windows NT, plus Microsoft SQL Server on Windows NT only. Because the OpenFusion Naming Service supports persistence on enterprise quality, high-availability database systems, it is fully scalable.

The Naming Service can view non-CORBA objects found in JNDI and standard JNDI clients can access a persistent Naming Service hierarchy.

The persistence mechanism must be configured before the Naming Service is started; this is normally done with the Administration Manager. The OpenFusion Naming Service can create a *jndi.properties* file whenever it starts, which contains the minimum information required to allow another JNDI client to access the Naming Service hierarchy. JNDI properties can be configured by application resource files, environment parameters passed via a hashtable, system properties or applet parameters in JNDI, with those specified in the hashtable taking priority.

The OpenFusion JNDI implementation is described in the *JNDI Guide*.

1.2.2.3 Caching

Several tunable caching policies are supported by the OpenFusion Naming Service, to help optimise performance. Available policies are:

- No cache (Read through and Write through)
- Read cache and Write through
- Read cache and Timed write
- Read cache and Batched write
- Read cache and Timed Batched write

plus

- minimum, maximum and interval

The read cache is purged as necessary using a least-recently-used algorithm when it reaches a user-defined size limit.

The minimum policy sets the minimum number of objects which will be left in the cache when it is cleared. The default value is zero (0).

The maximum policy sets the maximum number of objects which a cache will be allowed to hold. The default value is five hundred (500)

The interval policy sets the length of time, in seconds, that the cache is cleared, subject to the minimum policy described above. The default value is zero, which disables the interval policy.

The caching options are dynamic, so they can be changed whilst the service is running. This is normally done with the Administration Manager. Purging and Memory Management options are also described in *Supplemental Information* on page 38.



Care must be taken when specifying caching properties to avoid values which could result in *thrashing* (objects being rapidly loaded, removed from memory, and reloaded).

1.2.2.4 Purging and Memory Management



It is important to be aware of the differences between *purging* and *memory management*. Memory management is related to caching, and is performed without reference to the status of an object. The purging mechanism is part of the OpenFusion Naming Service and its handling of objects depends explicitly on their status.

These features can be enabled and controlled with properties specified in the Administration Manager. Please refer to *Supplemental Information* on page 38 for more information.

1.2.2.4.1 Purging

Purging is the deletion of invalid object references and *purgable* objects from a service. Object references are regarded as invalid when they are not active and not persistent. The OpenFusion Naming Service can most easily determine whether an object is purgable if the `com.prismt.openfusion.plugin.Purgable` interface is implemented.

1.2.2.4.2 Memory Management

Memory Management is the removal of objects from memory. The objects can be naming contexts as well as client and server objects. They are re-loaded on demand.

The purging and memory management options must be configured through the Administration Manager before the Naming Service is started.



Note

- Care must be taken when specifying memory management properties to avoid values which could result in *thrashing* (objects being rapidly loaded, removed from memory, and reloaded).
- When the Naming Service is being used with purging enabled, clients must always perform operations such as `resolve` from the *root* context, to avoid problems arising from attempts to resolve naming contexts which have been removed from memory.

Details of purging and memory management options are given in *Supplemental Information* on page 38.

1.2.2.5 Load Balancing Concepts

The purpose of load balancing is to optimise the use of available resources in order to minimise the time between the issue of a request for a service and the performance of that service.

Frequent requests from many clients for a particular kind of service can be satisfied by any one of several servers which are capable of providing that service, without any client needing to know at the time of the request which servers are available to fulfil the request.

An example illustrates the principle: a printing service distributing print jobs to multiple printers. In order to provide the best service to users, the service allocates print jobs to the available printers according to predefined algorithms or *policies*.

The policies used may be simple or sophisticated. In the simplest case, where the available printers have identical capabilities, print jobs are allocated to each printer in turn as they are received (*a round robin* policy); the total number of printing requests is divided equally amongst the available printers. A sophisticated system would implement different policies to take account of the capabilities of individual printers and the characteristics of specific printing requests. It could, for example, allocate a print job based on the size of the job and the speeds of the available printers.

1.2.2.6 Load Balancing in OpenFusion



This section describes a proprietary load balancing solution which is specific to the OpenFusion Naming Service. As an alternative solution, OpenFusion also offers an implementation of the proposed OMG specification for Load Balancing. This is described in the *Load Balancing Service Guide*.

Load balancing is implemented in OpenFusion as a Quality of Service option which enables the service to bind multiple objects to the same name. It uses a *delegate* style, which means that the application interface can be separated from the control or management interface. The *alias* provides the application interface; it then makes local calls to methods on the load balancer object instead of implementing the interface itself.

When a new load balancer is required, the OpenFusion `LoadBalancingFactory` is used to create it. The `LoadBalancingFactory` is normally co-located with the OpenFusion Naming Service, and starts automatically with it. A policy is specified when the load balancer is created, but it can be changed dynamically if required.

The servers which are to be managed by the load balancer are then registered with it. The load balancer and the alias are both bound into the Naming Service. These bindings refer to the same object, but the Naming Service recognises the difference between them.

Details of the `LoadBalancingFactory` and `LoadBalancer` interfaces are described in *API Definitions* on page 30.

A load balancer can be applied to implementation of the printing service example mentioned earlier.

A client sends a print job requiring a laser printer to the printing service, and the printing service queries the laser printer load balancer via its alias. The load balancer uses its current policy to determine which printer the job should be sent to, and returns that printer to the printing service. The printing service then sends the job to the selected printer. This is illustrated in *Figure 2*.

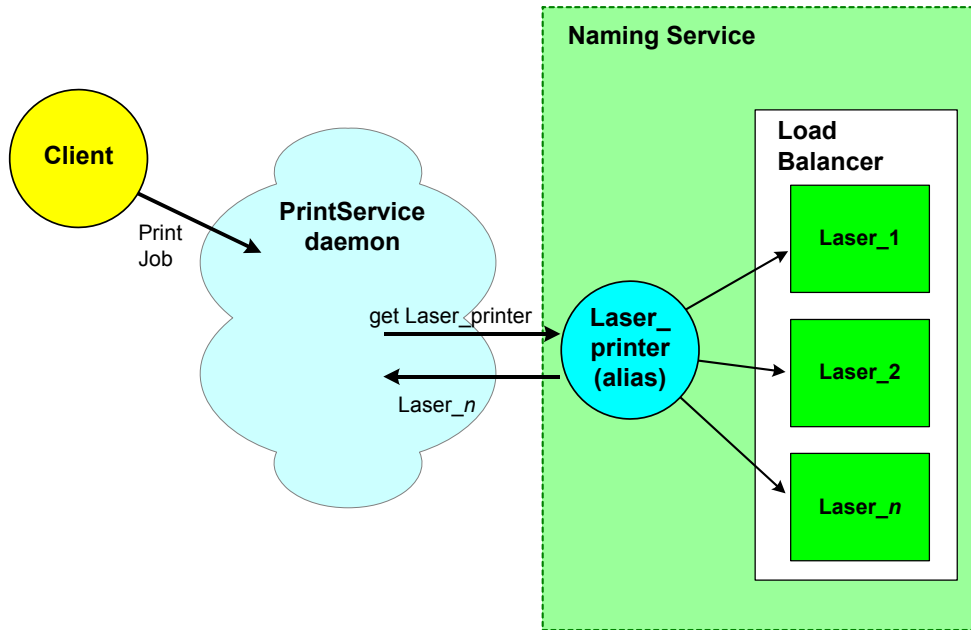


Figure 2 Load Balancing

OpenFusion load balancing is supplied with a number of standard policies for allocating requests to servers. These are designed to suit many common situations, but user-defined algorithms can be developed and plugged in if none of those is appropriate in a specific case. There is a complete list of the standard policies, together with details of the `LoadBalancerPlugin` interface, in *API Definitions* on page 30.

The policy used by a load balancer can also be changed through the Administration Manager.

It is easy to add objects to and remove them from a load balancer. In the example, a printer can be excluded if it goes off-line (when it runs out of toner, for example) and then reinstated (when the toner cartridge is replaced) or another printer can be added to the pool, without having to stop and re-start or otherwise affect the printing service.

1.2.2.7 Instrumentation

OpenFusion provides both general and service-specific instrumentation features which can be used for system monitoring, which in turn aids in problem identification, performance tuning, and so on. OpenFusion instrumentation consists of a set of properties that can be monitored either using the Administration Manager or remotely using SNMP.

In addition to properties that are read-only at runtime, OpenFusion provides some properties that can be set and reset at runtime as required, such as when a particular threshold value is reached or a time period has elapsed. Note that there is virtually no performance overhead involved in using any of the OpenFusion instrumentation features.

1.2.2.8 Fail-over

Fail-over is the ability of the OpenFusion Naming Service to activate a backup server if the master server fails, to improve reliability. Note that this functionality is currently only available when OpenFusion is running with the OpenFusion JacORB or VisiBroker ORB from Inprise.

To implement fail-over, the following Service configuration is required:

- Two Naming Services, each registered with the same process ID.
- Each Naming Service must be configured to see the same data.
- One Naming Service must be marked as the *System Master* (by setting the **System Master** property for the **NamingSingleton** in the Administration Manager).

The fail-over options must be configured through the Administration Manager before the Naming Service is started.

1.2.2.9 Replication

Replication is the duplication of data across two or more databases. The duplication and synchronisation is normally performed by the database itself, and is therefore transparent to the Naming Service. This enables two or more Naming Services to use the same data, but from physically distinct databases, which may help improve performance.

2 Using Specific Features

This section describes how to use the Naming Service with illustrative examples in Java.

It first shows how to create and destroy naming contexts and name bindings, how to retrieve the contents of a naming context, and how to resolve a binding to an object. The load balancing features of the OpenFusion Naming Service are demonstrated later.

The available operations are listed in API Definitions on page 30, which includes additional information which is useful in developing applications for the OpenFusion Naming Service.

The exceptions raised by the OpenFusion Naming Service and Load Balancer are listed in Supplemental Information on page 38.

An example application using the service, complete with source code and a description of how to compile and run it, is supplied elsewhere as part of the product distribution.

i Note

- No CORBA system exceptions are caught in any of these examples; code to deal with them has been omitted for the sake of clarity and brevity. These exceptions must of course be properly caught and handled in a working system.
- The following libraries must be imported into any application using the OpenFusion Naming Service:

```
import org.omg.CosNaming.*;  
import org.omg.CosNaming.NamingContextPackage.*;  
import org.omg.CosNaming.NamingContextExt.*;  
import org.omg.CosNaming.NamingContextExtPackage.*;
```

- The following import statements should also be added when load balancing is enabled:

```
import com.prismt.cos.CosNaming.NamingExtensions.*;  
import  
com.prismt.cos.CosNaming.NamingExtensions.LoadBalancerPackage.*;
```

2.1 Obtaining the Root Context

Before any objects or naming contexts can be added to (bound) or found (resolved) in the Naming Service, the root or initial context must be obtained. This is achieved by using `resolve_initial_references`:

```
org.omg.CORBA.Object obj = null;
org.omg.CORBA.ORB orb = null;
NamingContextExt rootContext = null;

orb = ObjectAdapter.init (args);

try
{
    obj = orb.resolve_initial_references ("NameService");
    rootContext = NamingContextExtHelper.narrow (obj);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex)
{
    System.err.println ("Failed to resolve NameService");
    System.exit (1);
}
```

2.2 Naming Context Creation and Destruction

The `NamingContext` interface provides two `NamingContext` creation operations and a single destroy operation, defined in IDL as:

```
NamingContext new_context ();

NamingContext bind_new_context (in Name n)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);

void destroy () raises (NotEmpty);
```

The `new_context` operation creates a new `NamingContext` object which is not bound to any other `NamingContext`:

```
NamingContext newContext = rootContext.new_context ();
```

The `bind_new_context` operation creates a new `NamingContext` and binds it using the supplied name.

The `destroy` operation requests the destruction of a `NamingContext`. The `NamingContext` must be empty. After `destroy` is invoked, no further operations can be invoked on the object reference of the `NamingContext`.

```
newContext.destroy ();
```



Bindings to a destroyed context are not removed. To do so would require a context to know about all of its parents as well as its children. An attempt to resolve a binding to a destroyed context will throw the `CORBA.INV_OBJREF` exception. Accordingly, bindings to a naming context should be removed before it is destroyed.

2.3 Binding and Unbinding Operations

The `NamingContext` interface provides five bind operations and a single unbind operation, defined in IDL as:

```
void bind (in Name n, in Object obj)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);

void rebind (in Name n, in Object obj)
    raises (NotFound, CannotProceed, InvalidName);

void bind_context (in Name n, in NamingContext nc)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);

void rebind_context (in Name n, in NamingContext nc)
    raises (NotFound, CannotProceed, InvalidName);

NamingContext bind_new_context (in Name n)
    raises (NotFound, CannotProceed, InvalidName, AlreadyBound);

void unbind (in Name n)
    raises (NotFound, CannotProceed, InvalidName);
```

The bind operations allow binding to occur between a name and either a generic CORBA object or a Naming Context. In order to bind a CORBA object, the name to bind against must be correctly constructed. Given a name with n components, the first $n - 1$ components must resolve to a bound `NamingContext`. However, the simplest case involves a name with only one component. The following code creates a new name with a single component and uses it to bind an object:

```
NameComponent newName[] = new NameComponent[1];

// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example name", "");

rootContext.bind (newName, demoObject);
```

The `rebind` operation is identical to the `bind` operation except that the `AlreadyBound` exception is not thrown; an existing binding with the same name is replaced by the new binding.

The `bind_context` operation adds a `NamingContext` object so that it becomes part of the graph of Naming Contexts used for resolving compound names. Note that a `NamingContext` can also be added using the `bind` operation but that the `NamingContext` will not become part of the graph of Naming Contexts and will not be used for resolving compound names.

```
NameComponent newName[] = new NameComponent[1];
newName[0] = new NameComponent ("example2", "");

rootContext.bind_context (newName, namingContextObject);
```

The `rebind_context` operation is identical to the `bind_context` operation except that the `AlreadyBound` exception is not thrown; an existing binding with the same name is replaced by the new binding.

The `bind_new_context` operation is equivalent to creating a new `NamingContext` and then adding it using `bind_context`:

```
NameComponent newName[] = new NameComponent[2];
newName[0] = new NameComponent ("example2", "");
newName[1] = new NameComponent ("example3", "context");

NamingContext newContext = rootContext.bind_new_context (newName);
```

The above examples use a compound name. The first component resolves to a `NamingContext` added with `bind_context`.

The `unbind` operation removes a name binding. It does not matter which of the `bind` operations was used to create the binding. The following example destroys bindings created with the previous example:

```
NameComponent comp[] = new NameComponent[2];
comp[0] = new NameComponent ("example2", "");
comp[1] = new NameComponent ("example3", "context");

rootContext.unbind (comp);
```

2.4 Accessing Naming Context Contents

Two operations are available for accessing the contents of Naming Contexts, defined in IDL as:

```
Object resolve (in Name n)
    raises (NotFound, CannotProceed, InvalidName);
void list (in unsigned long how_many, out BindingList bl,
    out BindingIterator bi);
```

The `resolve` operation takes a name and returns the object, if any, bound to that name.

```
NameComponent comp[] = new NameComponent[1];
comp[0] = new NameComponent ("example", "");
obj = rootContext.resolve (comp);
```

The `list` operation provides a means of accessing the entire content of a Naming Context. The `list` operation is the only means of determining the name bindings held by an arbitrary context. This operation returns results using two mechanisms: a `BindingList`, which is a sequence of bindings, and a `BindingIterator` which provides an iterator object to access the bindings.

The following example has two parts. The first part retrieves only the first five objects in a naming context using `BindingList`; the second part continues retrieving objects until the end of the list is reached using `BindingIterator`:

```
BindingIteratorHolder iter = new BindingIteratorHolder ();
```

```

BindingListHolder list = new BindingListHolder ();
rootContext.list (5, list, iter);

for (int i = 0; i < list.value.length; i++)
{
    System.out.println ("list entry " + i);
    System.out.print  (" name length: ");
    System.out.println (list.value[i].binding_name.length);
    System.out.print  (" name id: ");
    System.out.println (list.value[i].binding_name[0].id);
    System.out.print  (" name kind: ");
    System.out.println (list.value[i].binding_name[0].kind);
    System.out.print  (" bind type: ");
    System.out.println (list.value[i].binding_type);
}

BindingHolder binding = new BindingHolder ();

while (iter.value != null && iter.value.next_one (binding))
{
    obj = rootContext.resolve (binding.value.binding_name);
}

```

2.5 BindingIterator Operations

The `BindingIterator` interface provides two operations to access bindings and one destroy operation, defined in IDL as:

```

boolean next_one (out Binding b);
boolean next_n (in unsigned long how_many, out BindingList bl);
void destroy ();

```

The previous example showed the use of the `next_one` operation. This operation returns `true` when the binding argument contains a valid binding.

The `next_n` operation returns the number of bindings specified by the `how_many` variable in a `BindingList` sequence. The sequence is then accessed in the same way as the `BindingList` returned from a `NamingContext` `list` operation.

The following code fragment repeats the example of the `list` operation using the `next_one` operation to iterate through the contents:

```

BindingIteratorHolder iter = new BindingIteratorHolder ();
BindingListHolder list = new BindingListHolder ();
rootContext.list (5, list, iter);

for (int i = 0; i < list.value.length; i++)
{
    System.out.println ("list entry " + i);
    System.out.print  (" name length: ");
    System.out.println (list.value[i].binding_name.length);
    System.out.print  (" name id: ");
    System.out.println (list.value[i].binding_name[0].id);
    System.out.print  (" name kind: ");
    System.out.println (list.value[i].binding_name[0].kind);
    System.out.print  (" bind type: ");
    System.out.println (list.value[i].binding_type);
}

```

```

BindingHolder binding = new BindingHolder ();

while (iter.value != null && iter.value.next_one (binding))
{
    obj = rootContext.resolve (binding.value.binding_name);
}

```

2.6 Naming Context Extension Operations

The following examples show the use of the Interoperable Naming Service extension.

In a similar manner to the above, the initial NamingContextExt object is obtained by using the `resolve_initial_references` operation.

```

NamingContextExt rootExtContext = null;

try
{
    obj = orb.resolve_initial_references ("NameService");
    rootExtContext = NamingContextExtHelper.narrow (obj);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex)
{
    System.err.println ("Failed to resolve NameService");
    System.exit (1);
}

```

The name component is transformed into a stringified name. The extension provides the convenience operation `resolve_str` to resolve the stringified object.

```

org.omg.CORBA.Object res;
NameComponent newName[] = new NameComponent[2];
newName[0] = new NameComponent ("example2", "");
newName[1] = new NameComponent ("example2", "");

String stringified = new String (rootExtContext.to_string (newName));
System.out.println ("Stringified name is: " + stringified);

try
{
    res = rootExtContext.resolve_str (stringified);

    if (res != null)
    {
        System.out.println ("Object: " + res.toString ());
    }
}
catch (org.omg.CORBA.UserException ex)
{
    System.out.println ("Resolve Exception: " + ex);
}

```

It is also possible to convert back to a CORBA NameComponent and use that to resolve the object.

```

NameComponent copy[] = rootExtContext.to_name (stringified);

```

```
org.omg.CORBA.Object copyobj = rootExtContext.resolve (copy);
```

It is also possible to form a URL with a stringified name as shown below. This is an aid to portability and allows access to `CosNaming` via a standard URL naming scheme.

```
// The resulting URL address can then be used to resolve within
// a naming service.
System.out.println
(
    "to_url: "
    + rootExtContext.to_url ("rir:", stringified)
);
```

The following example shows how a `corbaloc` string is generated. The IOR key is then used in a narrow operation to resolve the service.

```
// The Corbaloc string that is generated can be used to resolve the
// service.
System.out.println ("Root IOR: " + orb.object_to_string
(rootContext));

// These operations are OpenFusion specific.

NamingContextExt newCtx = null;
IORDecoder decoder = new IORDecoder (rootContext);
StringBuffer locstr = new StringBuffer ("corbaloc::");

locstr.append (decoder.getHost ());
locstr.append (":");
locstr.append (decoder.getPort ());
locstr.append ("/");
locstr.append
    (StringUtil.encode (StringUtil.byteToString (decoder.getKey ())));

// StringBuffer locstr now contains the address. Attempt to resolve
// to check.
// Cannot use orb.string_to_object as no hooks are available to
// add support for INS extensions.
newCtx = NamingContextExtHelper.narrow
    (ORBAdapter.stringToObject (locstr.toString ()));

if (newCtx != null)
{
    System.out.println
    (
        "Successfully resolved context: " +
        ORBAdapter.objectToString (newCtx)
    );
}
else
{
    System.err.println ("Failed to resolve NameService");
}
```

The following example shows how a `corbaname` string may be used, for example
`corbaname:rir:#name/in/name.service`

or

```
corbaname:iiop:server.prismtechnologies.com:14005/
escaped_octal_key_string#name/in/name.service
```

The URL denotes an object bound into the Name Service at host `server.prismtechnologies.com` on port 14005. The key string would be used to resolve to the `NamingContext` and then the stringified name is resolved against that to yield an object reference.

```
// The Corbaname string that is generated can be used to resolve the
// service.
System.out.println ("Root IOR: " + orb.object_to_string
(rootContext));

// These operations are OpenFusion specific.

NamingContextExt newCtx = null;
IORDecoder decoder = new IORDecoder (rootContext);
StringBuffer corbaname = new StringBuffer ("corbaname:iiop:");

corbaname.append (decoder.getHost ());
corbaname.append (":");
corbaname.append (decoder.getPort ());
corbaname.append ("/");
corbaname.append
(StringUtil.encode (StringUtil.byteToString (decoder.getKey ()))));
corbaname.append ("#example name");

// StringBuffer corbaname now contains the NamingContext. Attempt to
// resolve to check.
// Cannot use orb.string_to_object as no hooks are available to
// add support for INS extensions.
newCtx = NamingContextExtHelper.narrow
(ORBAdapter.stringToObject (corbaname.toString ()));

if (newCtx != null)
{
    System.out.println
        ("Context IOR: " + ORBAdapter.objectToString (newCtx));
}
else
{
    System.err.println ("Failed to resolve NameService");
}
}
```

2.7 Using the LoadBalancingFactory

The initial `LoadBalancingFactory` is retrieved using the `resolve_initial_references` operation:

```
obj = orb.resolve_initial_references ("LoadBalancingFactory");
lbfactory = LoadBalancingFactoryHelper.narrow (obj);
```

It is used to create the `LoadBalancer` as shown below. The policy that the load balancer will use initially is specified when it is created, but this can be changed dynamically if required. The objects that are being added to the `LoadBalancer` are `CORBA NamingContext` objects.

```
lb = lbfactory.createLoadBalancer ("Roundrobin");
lb.add (ctx1);
lb.add (ctx2);
lb.add (ctx3);
```

2.8 Manipulating Objects in the LoadBalancer

Objects may be added as shown above, or they can be directly retrieved via the `get` operation of the `LoadBalancer` interface.

```
System.out.println ("LoadBalancer retrieved: " + lb.get ());
```

The `list` operation displays all objects currently bound into the `LoadBalancer`. Objects may be also be removed from the `LoadBalancer`.

```
org.omg.CORBA.Object elements[] = lb.list ();
```

The `remove` operation allows objects to be removed from the `LoadBalancer`. The required parameter is the `CORBA` object that is to be removed. For instance, the client may use the `list` operation and then iterate over those results to remove all the elements from the `LoadBalancer`.

```
lb.remove (anobject);
```

2.9 Using the LoadBalancer with the Naming Service

Remember that the client must perform two binds: one for the `LoadBalancer` and one for the `LoadBalancerAlias`. They refer to the same object but this separation allows the Load Balancing object to be dynamically changed even after it has been bound into the Naming Service because the Naming Service can distinguish between them.

```
// lboj and lbalias are NameComponents.
rootContext.bind (lboj, lb);
rootContext.bind (lbalias, lb.getAlias ());
```

Then, the client may either resolve the `lboj` to get the `LoadBalancer`, or the `lbalias` to perform the actual load balancing. For example, the alias below is retrieved. This code simply prints the objects it resolves. Contexts 1, 2, and 3 are returned when Round Robin has been selected. It then loops and returns context 1 again.

```
for (int i=0; i<4; i++)
{
    obj = rootContext.resolve (lbalias);

    System.out.println ("Resolved: " + obj);
    try
```

```

{
    NamingContextExt ctx = NamingContextExtHelper.narrow (obj);
    System.out.println ("Resolved name context: " + ctx);
}
catch (org.omg.CORBA.BAD_PARAM e)
{
    System.err.println
    (
        "Unable to narrow the object. Maybe LoadBalancing " +
        "is not enabled in the name service?"
    );
    break;
}
}

```

In contrast, in the following code `LoadBalancer` is retrieved and the objects bound into it are listed.

```

org.omg.CORBA.Object newlb = rootContext.resolve (lboobj);
lb = LoadBalancerHelper.narrow (newlb);

// List
org.omg.CORBA.Object initiallist[] = lb.list ();

```

2.10 Customizing the LoadBalancer

The `LoadBalancer` enables the client to use different algorithms (*policies*) when returning objects. A standard set of policies is supplied and automatically loaded. It is possible to design further plugins and either add these dynamically or configure them to be loaded at runtime. The alternative is to pass the class name to the `addPlugin` method. The plugin should implement the `LoadBalancerPlugin` interface as the example below shows.

```

public class LoadBalancingTestPlugin implements LoadBalancerPlugin
{
    // Must have a public no-args constructor.
    public LoadBalancingTestPlugin () throws PluginFailure
    {
    }

    public org.omg.CORBA.Object get
    (
        LoadBalancer reference,
        org.omg.CORBA.Object[] objects,
        java.lang.String policy
    )
    throws PluginFailure
    {
        // Always return second object bound
        if (objects.length < 2)
        {
            // Not enough objects bound to get the second object
            throw new PluginFailure ();
        }
        else
        {
            return objects[1];
        }
    }
}

```



```

    }

    public java.lang.String[] getSupportedPolicies ()
    {
        return new String [] { "TEST_POLICY" };
    }
}

```

The plugin may be added dynamically to the `LoadBalancer`. The policies of that plugin are then available for use, and can be selected dynamically.

```

System.out.println ("Adding the TestPlugin");
try
{
    lb.addPlugin
        ("com.prismt.cos.CosNaming.examples.LoadBalancingTestPlugin");
}
catch (InvalidPlugin e)
{
    System.out.println ("Caught exception: " + e);
    System.exit (1);
}

// Set the policy to that of the new plugin.
System.out.println ("Setting a new policy");
try
{
    lb.setPolicy ("TEST_POLICY");
}

```


3

Worked Example

This section contains a simple example application which demonstrates the way in which various features of the OpenFusion Naming Service are used together.

- i** No CORBA system exceptions are caught in any of the following examples: code to deal with exceptions has been omitted for the sake of clarity and brevity. These exceptions must, of course, be properly caught and handled in a working system.

The exceptions raised by the OpenFusion Naming Service and Load Balancer are listed in Supplemental Information on page 38.

3.1 Example Client

Step 1: Obtain the Naming Service Root Context

The initial Naming Context object is obtained by using the `resolve_initial_references` operation:

```
org.omg.CORBA.Object obj = null;
org.omg.CORBA.ORB orb = null;
NamingContextExt rootContext = null;

orb = ObjectAdapter.init (args);

try
{
    obj = orb.resolve_initial_references ("NameService");
    rootContext = NamingContextExtHelper.narrow (obj);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex)
{
    System.err.println ("Failed to resolve NameService");
    System.exit (1);
}
```

Step 2: Add a new binding

The addition of a new binding requires a name to identify the binding. In this example, there is only one name context, so the name consists of only one component. The following code allocates a name with a maximum sequence length of one:

```
NameComponent newName[] = new NameComponent[1];
```

The first component of the name sequence must now be set:

```
// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example", "");
```

Note that both the `id` and `kind` fields are always used when matching names. The `kind` field has no defined meaning within the Naming Service, so it is available for use by applications running on top of the Naming Service.

Assuming the existence of an object reference, `demoObject`, the object can now be bound:

```
// obtain demoObject reference ...
rootContext.bind (newName, demoObject);
```

Step 3: List the contents of a naming context

The `list` operation allows the contents of a Naming Context to be examined, with resulting name bindings returned via either a `BindingList` CORBA sequence or a `BindingIterator` object. In the following example, the `BindingList` is not used and all of the contents are returned using the iterator. Note that even though a zero length list is specified in the first argument of the `list` command, a valid (empty) sequence is still returned.

Once a name binding is obtained, the `resolve` operation returns the object associated with the binding. The resulting object may be a `NamingContext` which, if it was bound using `bind_context` or `bind_new_context`, will have a `bind_type` of `ncontext`.

```
BindingIteratorHolder bi = new BindingIteratorHolder ();
BindingListHolder bl = new BindingListHolder ();
NamingContextExt childContext = null;

rootContext.list (0, bl, bi);

BindingHolder binding = new BindingHolder ();
while (bi.value != null && bi.value.next_one (binding))
{
    try
    {
        obj = rootContext.resolve (binding.value.binding_name);
        if (binding.value.binding_type == BindingType.ncontext)
        {
            childContext = NamingContextExtHelper.narrow (obj);
            // do something with childContext
        }
        else
        {
            // do something with obj
        }
    }
    catch (org.omg.CORBA.UserException ex)
    {
        System.err.println ("resolve exception " + ex);
    }
}
```

4 API Definitions

This section describes selected interfaces and related aspects of the service. The complete IDL API is provided elsewhere as part of the product distribution.

The OpenFusion Naming Service provides most of its functionality through a single interface called `NamingContext`. A second interface, `BindingIterator`, provides support for enumerating the contents of Naming Contexts.

4.1 OMG Standard API Definitions

4.1.1 NamingContext Interface

The `NamingContext` interface provides operations to create, modify and examine name bindings within a naming context. The interface also provides operations to create and destroy naming contexts.

A compound name can be supplied when `NamingContext` operations take a name as a parameter. When a compound name is supplied, the operation is applied to the Naming Context identified by the compound name's components, excluding the last component. The last component identifies the binding within the selected Naming Context.

Table 2 Binding and Unbinding Operations

Operation	Description
bind	Creates a binding between a name and an object.
rebind	Creates a binding between a name and an object, replacing any existing binding with the same name.
bind_context	Creates a binding between a name and a Naming Context.
rebind_context	Creates a binding between a name and a Naming Context, replacing any existing binding with the same name.
bind_new_context	Creates and binds a new Naming Context.
unbind	Removes a name binding from a context.

Three operations support the creation and destruction of Naming Contexts:

Table 3 Naming Context Creation and Destruction

Operation	Description
new_context	Creates a new NamingContext object. This context is not bound to any other context.
bind_new_context	Creates a new NamingContext object and binds it using the supplied name.
destroy	Requests the destruction of the NamingContext.

Two operations access the contents of a Naming Context:

Table 4 Accessing Naming Context Contents

Operation	Description
resolve	Retrieves the object bound to a particular name.
list	Returns a list of name bindings associated with the Naming Context in the form of a sequence and a BindingIterator.

4.1.2 NamingContextExt Interface

The NamingContextExt interface provides operations to use URLs and stringified names.

Table 5 NamingContextExt Operations

Operation	Description
to_string	Accepts a compound name and returns a stringified name.
to_name	Accepts a stringified name and returns a compound name.
to_url	Accepts a URL address component and a stringified name and returns a URL.
resolve_str	A convenience operation that accepts a stringified name and performs a <code>resolve</code> in the same manner as <code>NamingContext::resolve</code> .
insToComponent	Converts an INS stringified name to a CORBA Name Component array.
componentToIns	Converts a CORBA Name Component array to an INS stringified name.

Conversions from URLs in the `corbaloc` and `corbaname` formats to objects are handled by `CORBA::ORB::string_to_object` but most ORBs currently do not support this functionality. However, the OpenFusion `ORBAdapter::stringToObject` operation does support this, and may be used instead. It is part of the `com.prismt.orb` package.

4.1.3 BindingIterator Interface

The `BindingIterator` interface provides two operations to access name bindings, and one destroy operation.

Table 6 BindingIterator Operations

Operation	Description
next_one	This operation returns the next binding. If there are no more bindings, <code>false</code> is returned.
next_n	This operation returns at most the requested number of bindings.
destroy	This operation destroys the iterator.

4.2 OpenFusion API Extensions

4.2.1 LoadBalancingFactory Interface

The `LoadBalancingFactory` is colocated with the Naming Service and is therefore automatically started with the Naming Service. One operation creates a `LoadBalancer` object.

Table 7 LoadBalancingFactory Operations

Operation	Description
createLoadBalancer	Creates and returns a new <code>LoadBalancer</code> object. The policy parameter is used to choose the initial policy for the <code>LoadBalancer</code> .

4.2.2 LoadBalancer Interface

A `LoadBalancer` is an object that may be bound into the Naming Service. This may have zero or more CORBA Objects placed inside it. The `LoadBalancer` is defined in IDL by:

```
interface LoadBalancer
{
...
    /**
     * This operation allows the LoadBalancer to retrieve
     * the alias object
     */
    LoadBalancerAlias getAlias ();
};
```

```
interface LoadBalancerAlias : LoadBalancer
{
};
```

The `LoadBalancer` implementation is known as *delegate style*.

The `LoadBalancer` should be bound when the client wishes to bind a `LoadBalancer` object that may be directly retrieved from the `NameService`. The `LoadBalancer` interface is used for control operations (such as adding objects or changing policies within the load balancer itself).

This applies whether or not Load Balancing has been enabled in the service.

Alternatively, the OpenFusion Naming Service will attempt to return an object bound into the `LoadBalancer` when a `LoadBalancerAlias` is bound and Load Balancing is enabled. The alias may be retrieved by the `getAlias` function shown above. The `LoadBalancerAlias` interface is used by applications to retrieve an object to perform a specific task.

The client therefore performs two binds: one for the `LoadBalancerAlias` and one for the `LoadBalancer`. Both refer to the same object but the Naming Service can distinguish between them. This separation allows the `LoadBalancer` object to be dynamically changed even after it has been bound into the Naming Service. For instance, `LoadBalancer` objects may be removed, added or their policy changed without the need for creating new `LoadBalancerAlias` objects.

Table 8 LoadBalancer Operations

Operation	Description
add	Adds an object to the <code>LoadBalancer</code> .
get	Retrieves an object from the <code>LoadBalancer</code> according to the specified policy.
remove	Removes the matching object from the <code>LoadBalancer</code> .
list	Returns a list of all the objects within the <code>LoadBalancer</code> .
setPolicy	Resets the current policy.
addPlugin	Adds a new plugin. The parameter should be a fully specified Java class name.
getAlias	Returns the delegate <code>LoadBalancerAlias</code> .

4.2.3 LoadBalancer Standard Policies

The Load Balancing interfaces have been exposed as configurable plugins, thereby allowing developers to write their own load balancing mechanisms should the default policies not be sufficient. The standard OpenFusion plugin contains the following policies:

Table 9 LoadBalancer Standard Policies

Policy Name	Description
Random	Returns the object references in a random order.
RoundRobin	Returns the object references in a sequential loop.
FirstBound	Returns the object reference that was first bound to the name.
Random_Active	Returns a random active object.
RoundRobin_Active	Returns only active objects sequentially.
FirstBound_Active	Returns the first bound active object.
Random_RemoveCurrent	Returns objects in a random order removing each as it does so.
FirstBound_RemoveCurrent	Returns the first bound object and removes it.
Random_Active_RemoveCurrent	Returns a random active object and removes it.
FirstBound_Active_RemoveCurrent	Returns a first bound active object and removes it.

RemoveCurrent

The `RemoveCurrent` version of each policy unbinds each object from the load balancer after it has been returned. This means that the load balancer contains a diminishing number of objects; calls made after the last object has been returned cause the `NoneBound` exception to be thrown.

These policies are useful when resources (objects to return) cannot be re-used once allocated or committed (returned by the load balancer), or require special processing before being re-used (triggered by the `NoneBound` exception).

Combining `RoundRobin` with `RemoveCurrent` has the same effect as combining `FirstBound` with `RemoveCurrent` and therefore has not been included.

i

Note that the name in the first column of the table is the name that should be passed to `setPolicy` in order to select one of the default policies. These default names are defined in `NamingExtensions.idl` as `const strings`.

4.2.4 LoadBalancerPlugin Interface

The `LoadBalancerPlugin` Interface is illustrated in *Figure 3*.

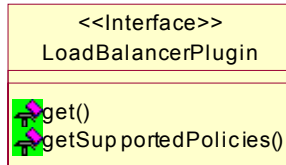


Figure 3 `LoadBalancerPlugin` Interface

Plugins must implement the `LoadBalancerPlugin` interface in the `com.prismt.cos.CosNaming` package. The `LoadBalancer` instantiates plugins listed in the property `Load Balancing Plugin` in the Administration Manager. This property is a comma-separated list of fully qualified class names. Each class must have a public, no argument constructor so that it can be instantiated by the `LoadBalancer`.

Table 10 `LoadBalancerPlugin` Operations

Operation	Description
get	Returns the appropriate object to the <code>LoadBalancer</code> implementation. There are two parameters: first, an array of CORBA objects denoting the available objects in the <code>LoadBalancer</code> , and secondly, a <code>String</code> policy. The policy parameter allows one policy to be chosen when the plugin supports multiple policies. The plugin throws a <code>PluginFailure</code> exception when an error occurs.
getSupportedPolicies	Returns an array of <code>Strings</code> containing the names of the policies that the plugin will support. These names directly correspond to the name that is used by the client when choosing a policy for use.

4.2.5 JNDIObject Interface

The Naming Service can display non-CORBA objects it finds in the JNDI hierarchy. In this situation, a CORBA `JNDIObject` will be created in order to display the object. The `JNDIObject` contains two read-only attributes:

- readonly attribute `string stringifiedObject;`

- `readonly attribute string className;`

For example, the attributes of a `String` stored in JNDI would contain the stringified value of the object and the classname `java.lang.String`.

The OpenFusion JNDI implementation is described in the *JNDI Guide*. Full details of the specification and descriptions of the standard features of the JNDI API and SPI are available from Sun Microsystems.

5 *Supplemental Information*

This section includes additional information which is necessary or useful for developing applications which use the Naming Service.

Administration properties and instrumentation are described first, then how to access them. There is a brief description of the relationship between the Naming Service and JNDI, followed by notes about using LDAP with the Naming Service. Purging and memory management features are described next, then XML import and export; finally there are lists of the exceptions that may be thrown.

5.1 Administration Properties and Instrumentation

Behaviour and performance of the Naming and Load Balancing Services can be controlled both programmatically and from the Administration Manager.

Please refer to *Configuration and Management* for details of controls and parameters for administering the OpenFusion Naming Service and Load Balancing. These properties can all be accessed using (SNMP).

5.2 Java Naming & Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) API is a generic API for accessing naming and directory services. The OpenFusion Naming Service is layered on top of JNDI. This allows it to access the OpenFusion service provider (which supports JDBC and Memory persistence) and also the Sun LDAP provider. Clients may access either transparently or use the OpenFusion JNDI SPI independently of the CORBA service.

Whenever the OpenFusion Naming Service starts, it automatically creates a basic `jndi.properties` file, which contains only the minimum information necessary to run the service. These settings can be overridden and additional properties specified by means of a Java hashtable.

The OpenFusion JNDI implementation is described in the *JNDI Guide*.

Please refer to the relevant Sun Microsystems documentation for details of JNDI standard functionality.

5.3 Lightweight Directory Access Protocol (LDAP)

The OpenFusion Naming Service is implemented in Java for platform independence. It is layered on top of JNDI, and can therefore utilise Sun Microsystems' JNDI LDAP service provider. This makes it useful for those organisations which use LDAP as their enterprise directory service; it can use standard LDAP authentication mechanisms.



It is assumed that the LDAP Server schemas are up to date. For details of LDAP configuration and functionality, please refer to the relevant Sun Microsystems documentation at this location:

<http://java.sun.com/products/jndi/tutorial/basics/prepare/content.html>.

5.4 Purging Options



It is important to be aware of the differences between *purging* and *memory management*. Memory management is related to caching, and is performed without reference to the status of an object. The purging mechanism is part of the OpenFusion Naming Service and its handling of objects depends explicitly on their status.

Purging is the deletion of invalid object references and *purgeable* objects from a service. Object references are regarded as invalid when they are not active and not persistent.

These features can be enabled and controlled with properties specified in the Administration Manager.

Purge on Load

When this option is selected, invalid object references are removed when contexts are first accessed after a server has been restarted.

Purge on List

When this option is selected, invalid object references are removed from a naming context when the list operation is performed on the context.

If either Purge option is enabled, a List operation which encounters an invalid context will automatically unbind the context and then re-try. A warning message is printed in the log file when a binding to an invalid context is removed in this way.

Purge Class Plugin

If used, this property must contain the name of a Java class, which can be publicly instantiated, that implements the `com.prismt.openfusion.plugin.Purgable` interface. This interface has one operation:

```
public boolean isPurgable (org.omg.CORBA.Object obj)
```

This class is used to determine whether or not to purge objects from the Naming Service. Typically a client will implement this operation to determine whether its object is persistent or transient and hence may be purged. This service will also check the active/inactive state.

If no class is specified for this property, the `ORBAdapter.isValid` method is used. This will successfully determine the state of objects created using the OpenFusion framework, but it will not work reliably for foreign objects (objects created in non-OpenFusion environments or on other ORBs).

5.5 Memory Management

Memory Management is the removal of objects from memory. The objects can be naming contexts as well as client and server objects. They are re-loaded on demand.

Memory Management is a caching option that can be enabled in the OpenFusion Naming Service. When enabled, cache purging can be performed either at regular intervals, or when the number of bound objects reaches a specified limit.

Object Purging

Object cache properties cannot be specified unless this option is enabled. When it is enabled, then the properties `Object Cache Minimum Size`, `Object Cache Maximum Size`, and `Object Cache Purging Interval` can be specified.

Object Cache Minimum Size

This is an integer value which specifies the minimum number of objects to keep in the cache. The default value is 0, which means that the cache is always completely flushed.

Object Cache Maximum Size

This is an integer value which specifies the maximum number of objects to hold in the cache; if not 0, it must be greater than the value specified for `Object Cache Minimum Size`. The default value is 0, which means that no object caching is performed.

When the cache is full, objects are removed using a least-recently-used algorithm until the value amount specified in `Object Cache Minimum Size` is reached.

Object Cache Purging Interval

This is an integer value which specifies the time interval in seconds between cache flushing operations. The default value is 0, which means that periodic flushing does not occur; the cache is only flushed when full. For any other value, cache flushing occurs at the specified intervals whether or not the maximum cache size has been reached.

When the cache is purged, objects are removed using a least-recently-used algorithm until the value amount specified in `Object Cache Minimum Size` is reached.



Note

- Care must be taken when specifying memory management properties to avoid values which could result in *thrashing* (objects being rapidly loaded, removed from memory, and reloaded).
- When the Naming Service is being used with purging enabled, clients must always perform operations such as `resolve` from the *root* context, to avoid problems arising from attempts to resolve naming contexts which have been removed from memory.

5.6 XML Export and Import

The OpenFusion Naming Service can both export and import XML files containing a representation of a naming hierarchy. This is performed at the command line; a specific naming hierarchy of a single Naming Service instance is handled with a single command.

To export a naming hierarchy to an XML file, use this command:

```
run com.prismt.cos.CosNaming.xml.ExportXML <options>
```

To import a naming hierarchy from an XML file, use this command:

```
run com.prismt.cos.CosNaming.xml.ImportXML <options>
```

The options for both commands are described in the tables.

The options can occur in any order.

The `-n` parameter specifying the XML file to use must be present.

The `-c` parameter specifies the name of the naming context that will be the root of the exported or imported naming hierarchy. This must be a valid INS name. If this parameter is not specified then the root of the naming hierarchy is used.

The Naming Service to use is determined in one of four ways. The resolve name of the service can be given, or its IOR can be given directly (with the `-i` option) or indirectly (with the `-f` option). If none of these is given, then the resolve name "NameService" is used.

Option	Description
<code>-n namingHierarchyFile</code>	name of XML file to export naming hierarchy into
<code>resolveName</code>	the resolve name of the Naming Service

Option	Description
-c targetNamingContext	the name of the naming context that is to be the root of the exported naming hierarchy
-f namingIORFile	the name of a file containing the IOR of the Naming Service
-i namingIOR	the IOR of the Naming Service

Option	Description
-n namingHierarchyFile	name of XML file to import naming hierarchy from
resolveName	the resolve name of the Naming Service
-c targetNamingContext	the name of the naming context that is to be the root of the imported naming hierarchy
-f namingIORFile	the name of a file containing the IOR of the Naming Service
-i namingIOR	the IOR of the Naming Service

Naming hierarchies can also be exported and imported using the OpenFusion Naming Service Manager as described in Chapter 11, *Naming Service Manager*, on page 82.

5.6.1 Exporting and Importing Cyclics

This section shows how the OpenFusion Naming Service handles *cyclics* (bindings which refer to a parent or grandparent context) when they occur in naming hierarchies included in XML exports and imports.

Figure 4 illustrates the principles of exporting and importing hierarchies with a straightforward example within a single Naming Service instance. The shaded hierarchy is exported (the context labelled B is the hierarchy root nominated with the -c option on the export command), and then re-imported and attached to B (with the -c option on the import command). Naming contexts are transient CORBA objects, so when the hierarchy is imported *new* contexts B1, C1 and D1 are created. Note that the imported hierarchy cannot be attached to A because the new context B1 will of course have the same name as the existing context B (each reference within a context must be to a unique name; we assume that B doesn't already contain a reference to another context or object with the same name as itself). The cyclic reference is created as intended and the integrity of the naming graph is maintained. Note that the new cyclic reference is to B1 and *not* to B.

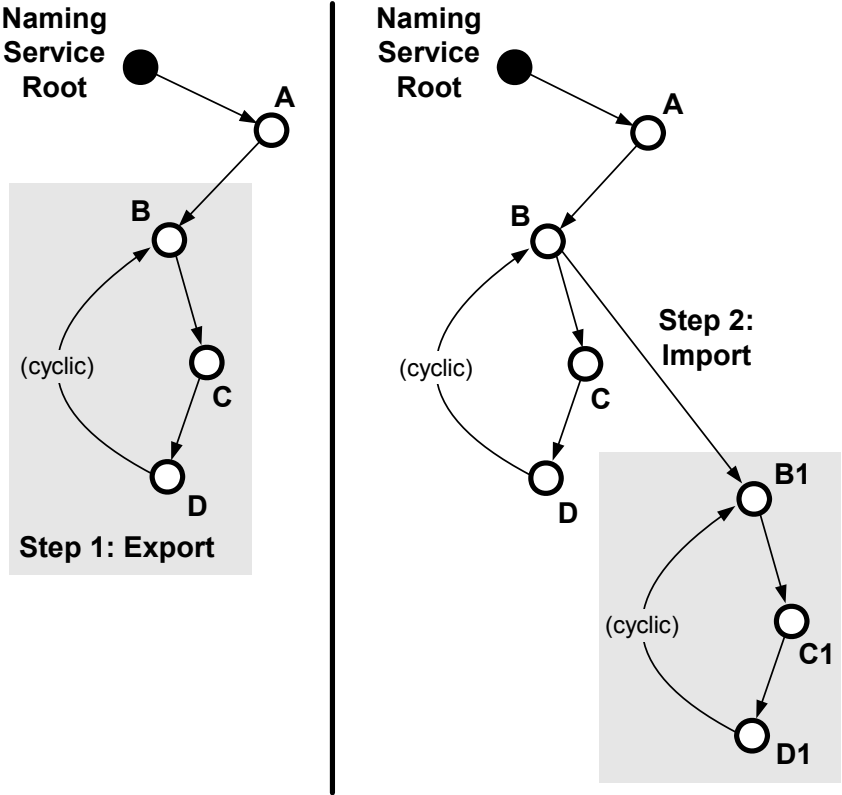


Figure 4 Naming Hierarchy Export and re-Import

5.7 Exceptions

The exceptions raised by the Naming Service are listed in *Table 11*.

Table 11 Naming Service Exceptions

Name	Purpose
AlreadyBound	Indicates an object is already bound to the specified name. Only one object can be bound to a particular name in a context.
CannotProceed	Indicates that the implementation has given up for some reason. The client, however, may be able to continue the operation at the returned naming context. One possible reason for this exception is that a Name Server holding one or more of the name bindings within a compound name is currently unavailable.
InvalidName	Indicates that the name is invalid. This implementation disallows zero length names only.
NotEmpty	Indicates that a naming context has bindings.
NotFound	Indicates that the name does not identify a binding or that the binding is not of the type required for the requested operations.

The exceptions raised by the Load Balancer are listed in *Table 12*.

Table 12 Load Balancer Exceptions

Name	Purpose
NoneBound	No objects are bound into the <code>LoadBalancer</code> .
InvalidPolicy	The specified policy is invalid.
InvalidPlugin	The specified class name is invalid.
ObjectNotFound	The object does not exist in the <code>LoadBalancer</code> .
PluginFailure	The plugin has failed for some reason. This exception is returned when a custom plugin has itself determined that it has failed.

A close-up, low-angle photograph of a computer keyboard, showing several keys in detail. The image is overlaid with a white grid pattern. The text is centered in the upper half of the image.

JAVA NAMING AND DIRECTORY INTERFACE

6 Description

The Java Naming and Directory Interface (JNDI) is an Application Programming Interface (API) and Service Provider Interface (SPI), defined by Sun Microsystems, that provides naming and directory functionality to Java applications whilst remaining independent of any specific directory implementation.

This guide describes the OpenFusion implementation of the JNDI specification rather than the standard functionality defined by Sun Microsystems.

Full details of the specification and descriptions of the standard features of the JNDI API and SPI are available from Sun Microsystems. Although this guide contains brief descriptions of the basic features of JNDI and its underlying concepts, it assumes that readers are familiar with Sun's standard documents and have copies available for reference.

This guide demonstrates how to use the OpenFusion SPI independently of the OpenFusion CORBA Naming Service, but accessing data written by the Naming Service. The OpenFusion SPI supports persistence in memory and JDBC databases.

6.1 Overview

6.1.1 Sun's JNDI Standard Features

The basic features of the JNDI specification include the ability to:

- give meaningful names to objects (*name bindings*)
- find names which have been bound to objects (*resolve*)
- group names in logical hierarchies (*naming contexts*)
- group distributed naming hierarchies (*federation*)
- access data through different directory services using a standard interface

6.1.2 OpenFusion Enhancements

Advantages of OpenFusion JNDI over the basic Sun specification include:

- improved, more robust multi-user access
- speed improvements, including write caching

6.2 Concepts and Architecture

6.2.1 Standard JNDI

The purpose of JNDI is to provide the ability to associate meaningful names with objects to make it easy to access those objects. A name *binding* is an association of a name with an object reference as a name-value pair.

Name bindings are grouped in hierarchies called *naming contexts*. A naming context is an object containing zero or more name bindings. Each name binding within a naming context refers to either another naming context (a *subcontext*) or an object. An hierarchy of contexts, subcontexts and objects is known as a *graph*. A context allows a client to perform various operations upon the objects bound within it.

A *naming system* is a set of many contexts of the same type. JNDI enables different naming systems to be connected together (*federation*).

The process of finding a name and retrieving the associated object reference is called *resolving* the name.

The JNDI architecture is illustrated in *Figure 5*, which shows the relationships between JNDI, Java applications, and object directory services.

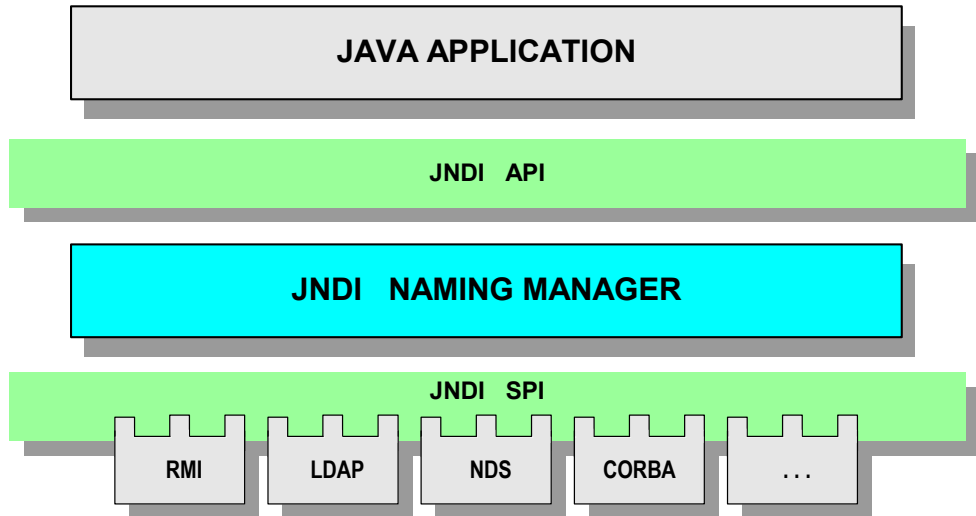


Figure 5 JNDI Architecture

JNDI-compliant applications can use generic calls on different directory services, such as Lightweight Directory Access Protocol (LDAP) servers, which plug in to the SPI. A Java client uses the API specifying the appropriate service provider in order to interact with the directory service.

OpenFusion CORBA Naming Service clients can access either the OpenFusion service provider or the Sun LDAP provider transparently. It is also possible to use the OpenFusion JNDI SPI independently of the CORBA service.

6.2.2 The Initial Context

In the JNDI, all naming and directory operations are performed relative to a context. Unlike the CosNaming Service, there is no absolute root. Therefore, the JNDI defines an initial context, `InitialContext`, which provides a starting point for naming and directory operations. This is retrieved through the `NamingManager` interface as shown below:

```
initialctx = NamingManager.getInitialContext (env);
```

A service provider must be specified in order to use the JNDI. This is part of the standard JNDI configuration. However, further configuration information may need to be supplied depending upon the service provider.

JNDI properties can be configured by application resource files, environment parameters passed via a hashtable (as above), system properties or applet parameters in JNDI, with those specified in the hashtable taking priority. See also *Supplemental Information* on page 58.

6.2.3 Naming Systems

A naming system maps names to objects within a directory service. The underlying directory service determines the syntax the JNDI client must use in the name, as a naming context represents a node within that directory service. For example, the `OpenFusionSPI` follows a left-to-right naming convention while the LDAP SPI uses a right-to-left notation.

The API methods that accept a name have two overloads: one that accepts a `Name` argument and one that accepts a string name. `Name` is an interface that represents a generic name; that is, an ordered sequence of zero or more components.

6.2.4 References and Addresses

Different SPIs may restrict what they can store directly, whereas the JNDI API does not carry any restrictions on what sort of objects may be stored. For instance, the CosNaming SPI only accepts `org.omg.CORBA.Object` (or its subclasses). JNDI defines a *Reference* for use when the serialized form of an object cannot be directly stored in the directory. A reference to an object contains one or more addresses, or communication end points, and information on how to construct a copy of this object. The JNDI will attempt to turn references looked up from the directory into the Java objects they represent. JNDI clients therefore present the illusion of directly storing Java objects in the directory.

7 OpenFusion SPI Implementation

The OpenFusion SPI implements the `javax.naming.Context` and `javax.naming.Reference` interfaces as described in the specification published by Sun Microsystems, except for one operation: the `javax.naming.Context` interface does not implement the operation `getNameInNameSpace`. This is because the OpenFusion SPI supports cyclic references in the name hierarchy, and a distinct fully qualified name does not make sense in this case. (A cyclic reference is one where a context contains a binding which refers back to a parent or grandparent context, which may be in a different naming system.) This feature was included in order to support the OpenFusion CORBA Naming Service.

7.1 Names

The naming scheme of the provider is very similar to that of the CosNaming interoperable Naming Service (INS) specification and the Sun CosNaming SPI.

The naming scheme is left-to-right, slash-separated, case sensitive and hierarchical. String names accepted by the SPI should be JNDI composite names in which each component is the stringified JNDI escaped form of a `CosNaming::NameComponent`. The stringified form of a `CosNaming::Name` is defined in the INS specification. Quoting problems may arise when the JNDI syntax defines meta-characters and the underlying provider has its own syntax. These can lead to many levels of escaping.

Two options are available:

- a `Name` may be returned by `nameParser.parse()`, where `nameParser` is a value obtained from the service provider
- the class `com.prismt.cos.CosNaming.OFNamingConverter` may be used

The name parser will return a compound name as the example below shows.

```
String strname = "A\\.\\"/B";
NameParser parser = rootctx.getNameParser ("");
Name jndiname = parser.parse (strname);

subctx = (Context)rootctx.lookup (jndiname);
```

The class `com.prismt.cos.CosNaming.OFNamingConverter` implements the interface shown in *Figure 6*. Note that this class carries out validity checks on the data passed to it.

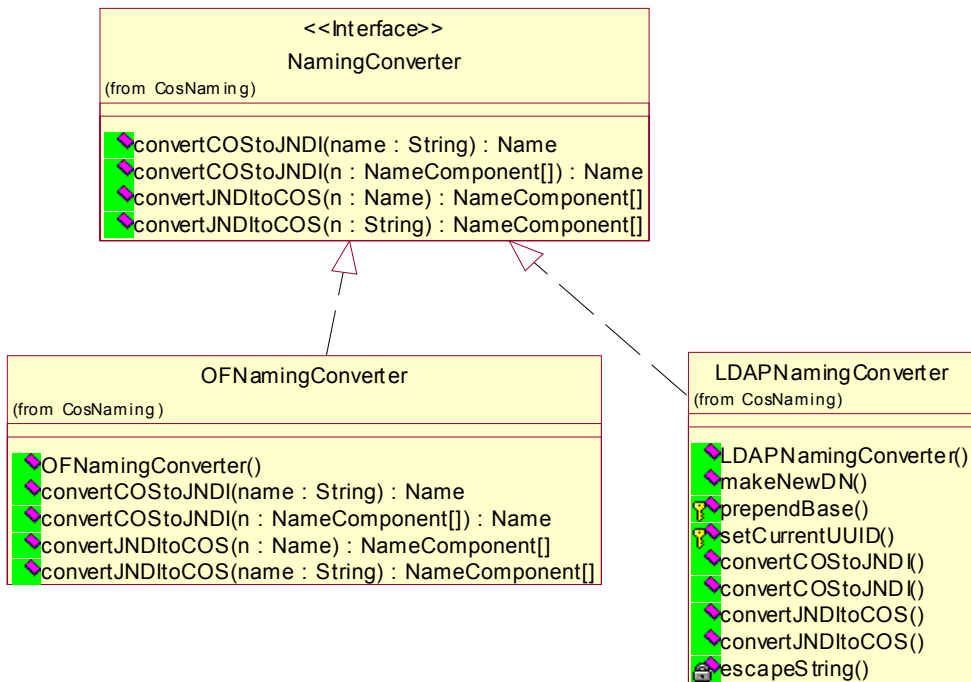


Figure 6 OFNamingConverter Interface

7.2 Java Objects

The OpenFusion SPI supports storage of the following types of Java objects using JDBC to store to disk or memory:

- serializable
- referenceable
- references

Note that any type of Java object may be stored when the provider is configured to use memory-based persistence and `StoreAnyObject` is set to `true`.

7.3 Supplied Factories

7.3.1 Storing CORBA Objects

The OpenFusion CORBA Naming Service stores CORBA objects in the OpenFusion SPI using the following factories, which implement `DirState` and `DirObject`:

```
com.prismt.cos.CosNaming.CORBAStateFactory
com.prismt.cos.CosNaming.CORBAObjectFactory
```

7.3.2 Storing RMI-IIOP Objects

To store RMI-IIOP objects in the OpenFusion JNDI, an additional `StateFactory` is required. This works in conjunction with the `CORBAStateFactory` and `CORBAObjectFactory` factories. The client must set the properties either programmatically or as system properties, as follows.

Programmatically

```
env.put (javax.naming.Context.OBJECT_FACTORIES,
  "com.prismt.cos.CosNaming.CORBAObjectFactory");
env.put (javax.naming.Context.STATE_FACTORIES, "com.prismt.j2ee.jndi.RMIStateFactor
y");
```

As System Properties:

```
-Djava.naming.factory.state=com.prismt.j2ee.jndi.RMIStateFactory
-Djava.naming.factory.object=com.prismt.cos.CosNaming.CORBAObjectFactory
```

7.4 Federation

The OpenFusion SPI supports federation. The JNDI specification defines the method of 'hooking' together naming systems so that the aggregate system can process composite names (names that span the naming systems).

The federation method uses:

- Weak separation. The context does not necessarily treat the separator as a naming system boundary. When processing a composite name, it consumes as many leading components as appropriate for the underlying naming system.
- Next Naming System pointers (junctions). The OpenFusion SPI supports dynamic implicit NNS pointers.

Note that the naming system is *non-terminal*: components from the naming system can appear anywhere in the composite name. Also the OpenFusion SPI cannot determine the naming system boundary syntactically but it can determine it dynamically.

8

Using Specific Features

This section provides some example Java code which demonstrates the use of the `OpenFusionSPI`.

Further source code examples are supplied elsewhere as part of the product distribution.

Detailed instructions for using JNDI can be found in the JNDI Tutorial published by Sun Microsystems.

It is possible to use the SPI to access data written by the Naming Service (assuming that the Naming Service has been configured to use the `OpenFusionSPI`). The following describes how to configure JNDI for access to Naming Service data written under JDBC.



It is recommend that `close` is always called in order to clean up and free resources used by the `OpenFusionSPI`.

8.1 JDBC-based Persistence

The `OpenFusion` Naming Service uses the JNDI root UUID (Universally Unique Identifier) and a SID (Service ID) value to establish access to the data in the database. These values must be set in order to access the data. It is possible for a standalone JNDI client to just set the root UUID. In this case, the SID value is set internally to be the same as the UUID.

When logging is enabled for the Naming Service and the level is set to `INFO`, these values are output to the log file, and can be retrieved from there if required. Typical log file entries are shown below:

```
INFO - Process ID: 0ba57cb0-4dae-11d4-ada7-ce8c9fa68378
INFO - Server ID: 0ba57cb0-4dae-11d4-ada7-ce8c9fa68378
INFO - Common: database = com.prismt.jdbc.Database@16304c8
INFO - initSID: database = com.prismt.jdbc.Database@16304c8
INFO - NamingService UUID is 0d68b080-4dae-11d4-ada7-ce8c9fa68378
```

The second and fifth lines are the ones containing the necessary information. These values can then be passed into the JNDI environment using a hashtable:

```
// Set the UUID.
env.put ("com.prismt.j2ee.jndi.OpenFusionSPI.UUID",
        args[0]);
// Set the SID.
```

```
env.put ("com.prismt.j2ee.jndi.OpenFusionSPI.SID",
        args[1]);
```

8.2 Accessing Data

The hierarchy can be browsed and modified once the root context is located.

```
initialctx = NamingManager.getInitialContext (env);
```

A JNDI client may bind a non-CORBA object into the Naming Service hierarchy.

```
Enumeration e = initialctx.list ("");

while (e.hasMoreElements ())
{
    NameClassPair np = (NameClassPair)e.nextElement ();
    System.out.println
    (
        "\tFound name " + np.getName () +
        " of class " + np.getClassName ()
    );
    System.out.println
    ("\tResolve: " + initialctx.lookup (np.getName ()));
}
```

The Naming Service detects that this is not a CORBA object when a CORBA client attempts to look it up. The service will display the JNDI object when it has been configured to view them. Otherwise, the service will log the following warning messages:

```
WARN - unable to process non-CORBA object. The object is a java.lang.String
WARN - ignoring element with the name of StringObject because it is a
non-CORBA object
```


9 Supplemental Information

i Refer to the Sun Microsystems documentation for JNDI specification details.

9.1 Configuration Properties

JNDI properties can be configured by means of a Java hashtable. Properties specified by this means are merged with any properties specified in the `jndi.properties` file, with those specified in the hashtable taking priority.

The location of the `jndi.properties` file is specified by

```
com.prismt.j2ee.jndi.OpenFusionSPI.JNDIPropertiesFile
```

If this is not specified, then the `jndi.properties` file is not written.

Extra configuration information on top of the standard JNDI environment may be needed, depending upon the naming/directory service and the SPI. Default values are used when the environment has not been configured; for example, `JDBC.URL` is set storing to the user's home directory.



When the OpenFusion CORBA Naming Service starts, it can automatically create a `jndi.properties` file in the location specified. This file contains only the JNDI settings relevant to the current Naming Service configuration (normally specified using the OpenFusion Administration Manager) to ensure that JNDI clients are configured to access the OpenFusion CORBA Naming Service hierarchy. Any existing `jndi.properties` file in the specified location is overwritten.

9.1.1 Standard Properties

The OpenFusion service provider supports these standard JNDI properties:

INITIAL_CONTEXT_FACTORY

This is the fully qualified class name of the factory class that creates the initial context for the provider, for example:

```
env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "com.prismt.cos.CosNaming.jndi.OpenFusionCtxFactory")
```

OBJECT_FACTORIES

This is a colon-separated list of fully qualified class names of object factory classes. The factories are responsible for creating objects from the information returned by the provider.

STATE_FACTORIES

This is a colon-separated list of fully qualified class names of state factory classes. The factories are responsible for creating and transforming an object into an acceptable form for storage.

9.1.2 Provider-specific Properties

The following properties are specific to the OpenFusion SPI.



The following prefix must be included with all of these properties:

```
com.prismt.j2ee.jndi.OpenFusionSPI
```

For example, the fully-qualified `JDBC.User` option is:

```
com.prismt.j2ee.jndi.OpenFusionSPI.JDBC.User="myUserName"
```

9.1.2.1 General

JndiPropetiesFile

The location of the `jndi.properties` file. If this is left blank, the `jndi.properties` file will not be created. The default is blank.

The `jndi.properties` file is useful for JNDI client applications that need to connect to the Naming Service hierarchy.

The OpenFusion JMS Manager requires a valid `jndi.properties` file. See the *Java Message Service Guide* for details.



When more than one Naming Service is used, each one *must* be configured to use a different `jndi.properties` file.

JndiOFPropetiesFile

The location that the `of.jndi.properties` file will be written to. If this is left blank, the file will not be created. The default is blank.

The `of.jndi.properties` file can be used by JBoss (and other application servers) to access the OpenFusion JNDI properties. As an alternative to using this file, properties could be hard coded or passed to an application as command-line parameters.

9.1.2.2 Persistence

JDBC.User

This is the name of the database user with create rights on the database.

JDBC.Password

This is the password of the database user named in `JDBC.User`.

JDBC.URL

This is the URL for the JDBC Connection to the database.

For memory-based persistence the URL string to should be:

```
JDBC:hsqldb: .
```

noting hsqldb is used for memory-based persistence and that a “.” must follow the last colon of the URL string.

JDBC.Type

The choices for the JDBC Type are `Oracle`, `Sybase`, `SQL Server` and `Informix`.

JDBC.Driver

This is the name of the JDBC Driver used to connect to the database. This is given in the form:

```
jdbc:oracle:thin:@ultra2:1526:EXPL
```

where `ultra2:1526` is the server name and port number (for example).

JDBC.AutoCreate

If the tables for the chosen database (selected using `JDBC.Type`) do not exist, then they are automatically created. The default value for hsqldb is `True` and for all other databases the default value is `False`, where `True` sets automatic table creation on.

9.1.2.2.1 Caching

The following properties are relevant to caching. Note that if write caching is required then read caching must also be enabled. To disable write caching, both `TimedWrite` and `BatchedWrite` must be set to 0. To disable read caching, both `ReadCache.Min` and `ReadCache.Max` must be set to 0.

TimedWrite

This is an integer value which specifies the time interval in seconds between cached writes. The default value is 0, which means that writes are not cached.

BatchedWrite

This is an integer value which specifies the time interval in milliseconds between batched writes. The default value is 0, which means that writes are not batched.

ReadCache.Min

This is an integer value which specifies the minimum number of objects to keep in the read cache. The default value is 0, which means that the cache is always completely flushed.

ReadCache.Max

This is an integer value which specifies the maximum number of objects to hold in the read cache; if not 0, it must be greater than the value specified for *ReadCache.Min*. The default value is 0, which means that no read caching is performed.

ReadCache.Int

This is an integer value which specifies the time interval in seconds between read cache flushing operations. The default value is 0, which means that periodic flushing does not occur; the cache is only flushed when full. For any other value, cache flushing occurs at the specified intervals whether or not the maximum cache size has been reached.

9.1.2.2.2 UUID and SID

UUID

This is the context identifier. It must be specified in order for data to remain persistent across sessions when persistence is set to `File` and `JDBC`. By default a new UUID is generated for each instance.

SID

This is the Service ID. It is a UUID used internally by the OpenFusion Naming Service. It is required when access to the Naming hierarchy is desired.

```
// Create a hashtable for the environment
Hashtable env = new Hashtable ();

// Use file based persistence
env.put
    ("com.prismt.j2ee.jndi.OpenFusionSPI.JDBC.URL",
    "jdbc:hsqldb:/tmp");
// Use a read cache.
env.put ("com.prismt.j2ee.jndi.OpenFusionSPI.ReadCache", "100");
// Set the UUID.
env.put
    (
        "com.prismt.j2ee.jndi.OpenFusionSPI.UUID",
        "8e82d2c0-1d04-11d4-844f-a0b231700aae"
    );

// Set initial context
env.put
    (
        javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "com.prismt.j2ee.jndi.OpenFusionCtxFactory"
    );

try
{
    // Get the root context
```

```
rootctx = NamingManager.getInitialContext (env);  
}
```



The provider generates a new UUID when the `UUID` option is not specified. However, this represents the starting point for the hierarchy, much like an LDAP server URL. Note that the data *cannot* be retrieved when this is not specified in future sessions.

9.2 Exceptions

JNDI has an hierarchy of exceptions that may be thrown. Clients may catch `NamingException` or any of its derived classes.

Full details of the standard exceptions are available in the JNDI API documentation available from Sun Microsystems.

A close-up, low-angle photograph of a computer keyboard, showing several keys in detail. The keys are white and set against a dark background. A white grid pattern is overlaid on the entire image, creating a sense of structure and connectivity. The lighting is soft, highlighting the texture of the keys.

CONFIGURATION AND MANAGEMENT

10 Naming Service Configuration

The configuration of Singleton properties specific to the Naming Service is described in this section. These properties appear in the Administration Manager, a graphical user interface (GUI) based administration tool included with the OpenFusion Graphical Tools.

The Administration Manager can be used to set the Singleton properties. These properties can also be set programmatically, generally as described in the service description sections.

Details for configuring Persistence, Logging, CORBA, Java and System properties for the Naming Service are described in the System Guide.

10.1 Common Properties

Instances of some common properties are used by a number of different OpenFusion CORBA Services' interfaces and services. Settings for these property instances appear in the Administration Manager's Object Hierarchy for the service's Singleton node. This small group of properties are included in this section in order to facilitate configuration of the service while using the Administration Manager. These properties include:

- IOR Name Service Entry
- IOR URL
- IOR File Name
- Resolve Name
- IOR Name Service

10.2 NameSingleton Configuration

10.2.1 CORBA PropertiesOR Name Service Entry

The Naming Service entry for the Singleton.

Property Name	Object.Name
Property Type	FIXED

Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

IOR URL

The **IOR URL** property specifies the location of an Interoperable Object Reference (IOR) for the Service, using the Universal Resource Locator (URL) format. This information is used when a client attempts to resolve a reference to the Service. Some examples are:

```
file:/usr/users/openfusion/servers/NameService.ior
http://www.prismtech.com/of/servers/NameService.ior
corbaloc::server.prismtechnologies.com/NameService
```

The Naming Service supports URLs in *Corbaloc*, *Corbaname*, *file*, *FTP* and *HTTP* URL formats, although some ORBs do not support all of these mechanisms. Consult your ORB documentation for specific details.

Property Name	IOR.URL
Property Type	FIXED
Data Type	URL
Accessibility	READ/WRITE
Mandatory	NO

IOR File Name

The **IOR File Name** option specifies the name and location of the IOR file for the Singleton. If this property is not set, the IOR file name will be:

```
<INSTALL>/domains/<domain>/<node>/<service>/<singleton>/<singleton>.ior
```

where **<INSTALL>** is the OpenFusion installation path. See the *System Guide* for details of the `domains` directory structure.

Property Name	IOR.File
Property Type	FIXED
Data Type	FILE
Accessibility	READ/WRITE
Mandatory	NO

Resolve Name

The ORB Service resolution name used to resolve calls to the Singleton

Property Name	ResolveName
Property Type	FIXED
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

IOR Name Service

The name of the Naming Service which will be used to resolve the Singleton object.

Property Name	IOR.Server
Property Type	FIXED
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

10.2.2 Lightweight Directory Access Protocol (LDAP)

The Naming Service uses Sun Microsystems' JNDI (Java Naming and Directory Interface) LDAP provider. This allows the Naming Service to be stored in a standard LDAP server. Caching is not supported under the LDAP persistence option.

LDAP User

The administrator of the LDAP server may want each user to have their own login name and password. This property specifies the user name. The user name should be in the fully qualified LDAP format, for example:

```
uid=RNCross,ou=People,o=prismtechnologies.com
```

Property Name	DB.LDAP.User
Property Type	STATIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

LDAP Password

The administrator of the LDAP server may want each user to have their own login name and password. This property specifies the password.

Property Name	DB.LDAP.Password
Property Type	STATIC
Data Type	PASSWORD
Accessibility	READ/WRITE
Mandatory	YES

LDAP URL

The URL specifies the location within the LDAP server where the Naming Service should store its persistent data. The data will not appear in the traditional hierarchical format due to limitations of the LDAP storage mechanism.

An example URL is:

```
ldap://excalibur.prismsystems.com:2809/ou=OpenFusion
Naming Service,o=prismtechnologies.com
```

Property Name	DB.LDAP.URL
Property Type	STATIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

LDAP Trace

Output hexadecimal dump of the incoming and outgoing LDAP ASN.1 BER packets from the LDAP server.

Property Name	DB.LDAP.Trace
Property Type	FIXED
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

LDAP Security

LDAP Authentication Mechanism. The security method may be:

- **None:** anonymous bind.
- **Simple:** clear-text password.

- **SASL**: Simple Authentication and Security Layer, defined in RFC2222.

The administrator must enable all privileges upon the target location in the LDAP hierarchy when anonymous bind is selected. The LDAP v3 protocol uses the SASL to support pluggable authentication. This means that the LDAP client and server can be configured to negotiate and use possibly non-standard and/or customized mechanisms for authentication, depending on the level of protection desired by the client and the server. The LDAP v2 protocol does not support the SASL.

Property Name	DB.LDAP.Security
Property Type	STATIC
Data Type	ENUM
Accessibility	READ/WRITE
Mandatory	YES

LDAP SASL Mechanism Names

A list of mechanisms should be entered in the configuration tool when the SASL option is chosen, for example:

DIGEST-MD5 CRAM-MD5

This specifies that DIGEST-MD5 authentication is to be used, or that CRAM-MD5 authentication is to be used when the SASL mechanism is unavailable. An `AuthenticationNotSupportedException` will be thrown when neither is available.

Property Name	DB.LDAP.SASL
Property Type	STATIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

10.2.3 Persistence Options

The Naming Service provides two extra persistence options and a read cache for the caching of naming contexts.

The different kinds of caching available to the Naming Service are:

- No Cache, i.e. Read Through / Write Through. This is automatically used for failover.
- Read Cache / Write Through.
- Read Cache / Timed Writes with the value in seconds.

- Read Cache / Batched Writes.
- Read Cache / Batched and Timed Writes.

Read Cache Flush Interval

The interval, in seconds, between read cache flush operations. A least-recently-used algorithm is employed to reduce the size of the cache to the level of the **Read Cache Minimum Size**.

The default value is 0 (zero), which indicates no timed cache flush will be performed.

Property Name	DB.ReadCache.Int
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Read Cache Maximum Size

The maximum number of objects that the read cache will be allowed to hold. A value of zero means that there is no read cache. When the cache reaches the read limit size, a least-recently-used algorithm is employed to reduce the size of the cache to the level of the **Read Cache Minimum Size**.

The default value is 500.

The Read Cache Maximum Size must be set greater than zero if a write cache is required, as it is not possible to have a write cache without a read cache.

Property Name	DB.ReadCache.Max
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Read Cache Minimum Size

The minimum number of objects which will be left in the cache when it is cleared. The default value is 0 (zero).

Property Name	DB.ReadCache.Min
Property Type	DYNAMIC

Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Write Cache Write Interval

The write interval option refers to the delay (in seconds) between saving object state changes within a server, and writing this information to persistent store. This option is a performance optimization feature as it can be used to prevent the service making a lot of small updates to the persistent store.

A value of `zero` indicates no delay. Changes are written immediately to the persistent store if both the **Write Cache Write Interval** and **Write Cache Batch Size** are set to `zero`.

The default value for this property is `zero`. Increasing the write interval value may improve performance when the data held by a service is changing rapidly.

Property Name	DB.WriteInterval
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Write Cache Batch Size

The *Write Batch Size* option specifies the maximum number of updates that will be buffered before the data is written to persistent storage. Just as for the write interval option, the write batch size option is also a performance optimization feature.

A value of `zero` indicates that the updates are not buffered but are written immediately to the datastore. Increasing this property value may improve performance when the data held by a service is changing rapidly.

The `Read Cache Maximum Size` must be set greater than `zero` if a write cache is required, as it is not possible to have a write cache without a read cache.

The effect of setting both the *Write Interval* and *Write Batch Size* to values greater than `zero` is that of batched timed writes.

Property Name	DB.WriteBatch
Property Type	DYNAMIC

Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Naming Data Storage Type

This property sets the persistent storage type. The type can be:

- Default
- LDAP

If Default is selected, the data store will default to the location of the service data (using JDBC). See the *System Guide* for details.

Property Name	DB.NameDataPersistence
Property Type	STATIC
Data Type	ENUM
Accessibility	READ/WRITE
Mandatory	YES

10.2.4 Instrumentation Properties

The interfaces for setting the instrumentation properties are given below. For information on managing instrumentation, in the *System Guide*.

Count of resolve operations

The number of resolve operations since the Service started or was last reset.

Property Name	ResolveCount
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

Count of rebind context in service

The number of rebind contexts in service since the Service started or was last reset.

Property Name	ReBindContextCount
Property Type	DYNAMIC

Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

Count of context bind operations

The number of context binds in service since the Service started or was last reset.

Property Name	BindContextCount
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

Count of unbind operations

The number of unbinds in service since the Service started or was last reset.

Property Name	UnBindCount
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

Count of rebind operations

The number of rebinds in service since the Service started or was last reset.

Property Name	ReBindCount
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

Count of bind operations

The number of binds in service since the Service started or was last reset.

Property Name	BindCount
Property Type	DYNAMIC

Data Type	COUNTER
Accessibility	READ/WRITE
Mandatory	NO

10.2.5 General Properties

JNDI ContextFactory Cache Flush Interval

The internal ContextFactory cache can be purged to prevent the possibility of memory leaks. This property specifies the interval, in seconds, between ContextFactory cache flush operations. A value of `zero` indicates that no timed cache flush will take place.

JNDI ContextFactory Cache Flush Interval is used in conjunction with the **JNDI ContextFactory Cache Maximum Size** and **JNDI ContextFactory Cache Minimum Size** properties to determine the purging behaviour.

Property Name	<code>jndiCtxCacheInt</code>
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

JNDI ContextFactory Cache Maximum Size

The maximum number of contexts allowed in the ContextFactory cache. When the cache exceeds this size, contexts are purged according to a least-recently-used algorithm

Property Name	<code>jndiCtxCacheMax</code>
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

JNDI ContextFactory Cache Minimum Size

The size that the ContextFactory cache will be reduced to following a cache flush. For example, if this property is set to 10 then all but 10 contexts will be purged during a flush operation.

Property Name	jndiCtxCacheMin
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

JNDI Properties File

The location of the `jndi.properties` file. If this is left blank, the `jndi.properties` file will not be created.

The `jndi.properties` file is useful for JNDI client applications that need to connect to the Naming Service hierarchy.

The OpenFusion JMS Manager requires a valid `jndi.properties` file. See the *Java Message Service Guide* for details.



When more than one Naming Service is used, each one *must* be configured to use a different `jndi.properties` file.

Property Name	jndiPropertiesFile
Property Type	STATIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

JNDI OF Properties File

The location that the `of.jndi.properties` file will be written to. If this is left blank, the file will not be created.

The `of.jndi.properties` file can be used by JBoss (and other application servers) to access the OpenFusion JNDI properties. As an alternative to using this file, properties could be hard coded or passed to an application as command-line parameters.

Property Name	jndiOFPropertiesFile
Property Type	STATIC

Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

JNDI Root ID

This option allows the root ID used by the JNDI hierarchy to be manually configured. This is useful when used in conjunction with the Server Persistent ID (SID) property (see the *System Guide*) as these are then known values that may be passed to JNDI client programs. These clients can then access the Naming Service persistent data.

Property Name	JNDIID
Property Type	STATIC
Data Type	UUID
Accessibility	READ/WRITE
Mandatory	NO

Enable Load Balancing

This allows load balancing to be performed by the Naming Service.

Property Name	LoadBalancing
Property Type	DYNAMIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

View Non-Corba Objects

This allows the Naming Service to browse a JNDI hierarchy even when non-CORBA objects (e.g. `java.lang.String`) have been stored. The Naming Service will log and ignore any non-CORBA objects it encounters when this option is disabled.

Property Name	ViewNonCorba
Property Type	DYNAMIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

Purge on List

Invalid object references (that is, those object references which are not active and not persistent) are removed from a naming context when the list operation is performed on the context and **Purge on List** is selected.

Property Name	Clean.List
Property Type	DYNAMIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

Purge on Load

Invalid object references (that is, those object references which are not active and not persistent) are removed when contexts are first accessed after a server has been restarted and **Purge on Load** is selected.

Property Name	Clean.Load
Property Type	DYNAMIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

Purge Class Plugin

This should be a publicly instantiable Java class that implements the `com.prismt.openfusion.plugin.Purgable` interface. This interface has one operation:

```
public boolean isPurgable (org.omg.CORBA.Object obj)
```

This class is used to determine whether or not to purge objects from the Naming Service. Typically, a client will code this operation to determine whether their object is persistent or transient and hence may be purged. This service will also check the active/inactive state. The `ObjectAdapter.isTransient` method is the default used when a class is not specified. This will successfully determine the persistent state for objects created using the OpenFusion framework, but it will not work for foreign objects.

Purging is the deletion of invalid object references and *purgable* objects from a service. Object references are regarded as invalid when they are not active and not persistent. The OpenFusion Naming Service can most easily determine whether an object is purgable if the `com.prismt.openfusion.plugin.Purgable` interface is implemented. See the *OpenFusion Naming Service Guide* for further details.

Property Name	<code>Clean.PurgeClass</code>
Property Type	DYNAMIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

System Master

This property should be set to `true` (checked) if this is the master naming service for a system. There can be only one master naming service.

Property Name	<code>Resolver</code>
Property Type	STATIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

10.3 LoadBalancingFactorySingleton Configuration

IOR Name Service Entry

The Naming Service entry for the Singleton.

Property Name	<code>Object.Name</code>
Property Type	FIXED
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

IOR URL

The **IOR URL** property specifies the location of an Interoperable Object Reference (IOR) for the Service, using the Universal Resource Locator (URL) format. This information is used when a client attempts to resolve a reference to the Service. Currently only *http* and *file* URLs are supported, for example:

```
file:/usr/users/openfusion/servers/NameService.iior
```

<http://www.prismtech.com/openfusion/servers/NameService.ior>

Property Name	IOR.URL
Property Type	FIXED
Data Type	URL
Accessibility	READ/WRITE
Mandatory	NO

IOR File Name

The **IOR File Name** option specifies the name and location of the IOR file for the Singleton. If this property is not set, the IOR file name will be:

```
<INSTALL>/domains/<domain>/<node>/<service>/<singleton>/<singleton>.ior
```

where <INSTALL> is the OpenFusion installation path. See the *System Guide* for details of the domains directory structure.

Property Name	IOR.File
Property Type	FIXED
Data Type	FILE
Accessibility	READ/WRITE
Mandatory	NO

Resolve Name

The ORB Service resolution name used to resolve calls to the Singleton

Property Name	ResolveName
Property Type	FIXED
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

IOR Name Service

The name of the Naming Service which will be used to resolve the Singleton object.

Property Name	IOR.Server
Property Type	FIXED

Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

Load Balancing Plugin

Plugin class used to implement load balancing. This should be a publicly instantiable Java class.

Property Name	Classnames
Property Type	FIXED
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

Load Balancing Timeout

This property is used when an **Active** policy is selected. It controls the length of time that the ORB will attempt to communicate with an object before regarding it as inactive. The default value is zero.

Property Name	Timeout
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

11 Naming Service Manager

Use the Naming Service Manager to:

- Browse the Naming Service hierarchy.
- Add or delete **naming contexts**.
- Bind CORBA objects to the Naming Service.
- Launch other managers and browsers.
- Export and import the hierarchy as XML files (this function can also be performed from the command line).

i Various Naming Service management operations can also be performed using a command line tool, `NamingServiceMgrTool`. This tool and how to use it is described in the *System Guide*.

11.1 Running the Naming Service Manager

The Naming Service Manager can only be started if the Naming Service has been started. To start the Naming Service Manager, right-click on a running **NameSingleton** in the Administration Manager's **Object Hierarchy** and select **Naming Service Manager** from the pop-up menu. See the *System Guide* for details.

Alternatively, start the Naming Service Manager from the command line with the following command:

```
% run
  com.prismt.cos.treebrowser.naming.NamingServiceBrowser
  -name NameService
```

11.2 Using the Naming Service Manager

The Naming Service Manager shows the naming hierarchy as a graphical tree view. *Figure 7* illustrates how OpenFusion CORBA objects might be registered in the Naming Service Manager.

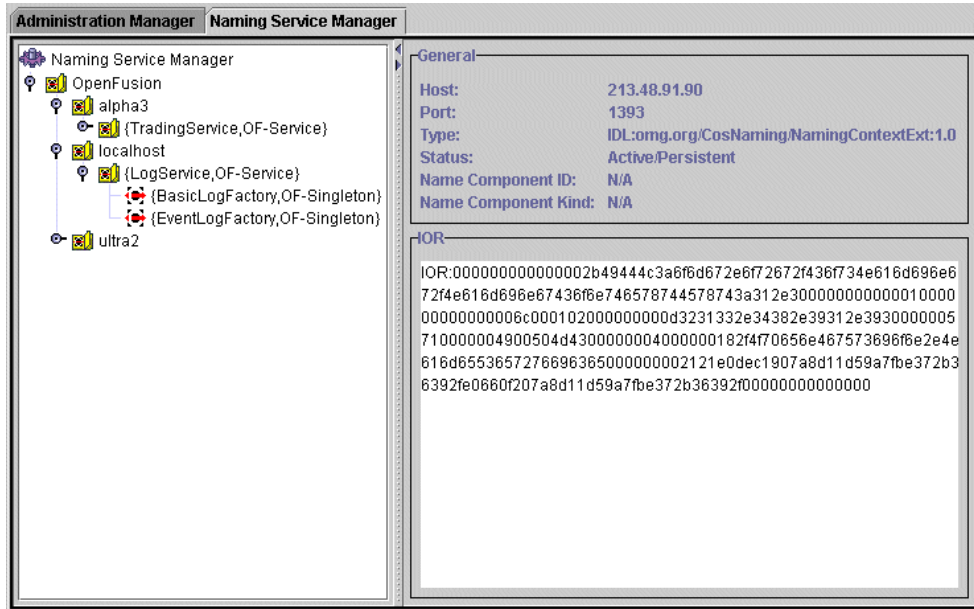


Figure 7 Naming Service Manager

Each object in the Naming Service is represented by an icon in the tree view. The object is labelled with its Id and/or Kind (if specified), in one of the following formats:

- {id,kind}
- id
- {,kind}






When an object is selected (highlighted) in the tree view, its details are shown in the properties pane to the right of the tree view.

An object's IOR can be selected in the properties pane and copied to the clipboard.

11.2.1 Object Icons

Different objects in the Naming Service Manager are identified by different icons in the tree view. These icons are shown in *Table 13*.

Table 13 Naming Service Object Icons



Icon	Node
	<p>Root Context</p> <p>The Naming Service root node represents the current instance of the Naming Service.</p>
	<p>Naming Context</p> <p>Represents an OpenFusion CORBA naming context.</p>
	<p>CORBA Object</p> <p>Represents a CORBA object binding. This must always be a <i>leaf</i> node in the hierarchy.</p>
	<p>Non-CORBA Object</p> <p>Represents a non-CORBA object binding. This must always be a <i>leaf</i> node in the hierarchy.</p>
	<p>Invalid Naming Context</p> <p>Represents a naming context which has been invalidated as a result of a linked object being destroyed.</p>

11.2.2 Tool Bar Buttons

The Naming Service Manager adds new buttons to the tool bar. These buttons are only available when the Naming Service Manager is active. The new tool bar buttons are shown in *Table 14*.

The buttons are disabled if a leaf node or an invalid naming context is selected in the naming hierarchy.

Table 14 Naming Service Manager Tool Bar

Button	Function
	Load Naming XML Load Naming Service information from an XML file.
	Save Naming XML Save Naming Service information in an XML file.

11.2.3 Adding a Naming Context

A new naming context must be added as a child of an existing naming context (or the root context) in the naming hierarchy. A naming context cannot be added as a child of a bound CORBA or non-CORBA object.

To add a new naming context to the naming hierarchy:

- Step 1:** Right-click on the parent naming context.
- Step 2:** Select **Add New Context** from the pop-up menu.
- Step 3:** Enter the **Id** and **Kind** of the new naming context in the **New Context** dialog box. Both of these fields are optional.
- Step 4:** Click the **OK** button.

The new naming context is added to the naming hierarchy as a child of the selected parent naming context.

11.2.4 Binding OpenFusion Services

When a Service is started in the Administration Manager, each of its Singletons will attempt to bind to the Naming Service if it is configured to do so (see below), and if the Naming Service is running when the Service starts.

This occurs each time the Service is started. (*Persistent* objects remain registered when the Service is stopped, with a **Status** of *Inactive/Persistent*.) If the bind is successful, entries for the Singletons are added to the Naming Service Manager hierarchy. To see any newly-started objects, right-click on the root node of the Naming Service Manager and select **Refresh Node** from the pop-up menu.

OpenFusion Singletons that register themselves in this way are bound directly under the Naming Service root context.

If a Singleton is to register itself with a running Naming Service when it is started, it must be configured to do so, as follows:

- The **IOR Name Service Entry** property of the Singleton must contain a valid INS name to identify the Singleton.
- The **IOR Name Service** property must contain the name of the Naming Service it is to bind to. (This should be `NameService` for the OpenFusion Naming Service.)
- The **Use Xbootclasspath** property of the Service containing the Singleton must be set to `true`.

11.2.5 Binding Objects

An object must be added as a child of an existing naming context (or the root context) in the naming hierarchy. An object cannot be added as a child of a bound CORBA or non-CORBA object (bound objects are always *leaf* nodes in the hierarchy).

An object's IOR is used to bind the object into the naming hierarchy. The IOR of an existing object must be copied to the clipboard before the object can be bound into the naming context. See the *System Guide* for details of querying objects in the Object Browser and copying the object's IOR to the clipboard.

Once the required object's IOR is copied, follow these steps to bind the object:

- Step 1:** Right-click on the naming context that the object will be bound to.
- Step 2:** Select **Paste New Binding** from the pop-up menu.
- Step 3:** Enter an **Id** and **Kind** for the bound object. Both of these fields are optional.
- Step 4:** Click the **OK** button.

The new object is added to the naming hierarchy as a child of the selected parent naming context.

It is possible to bind a naming context object as either an object or a naming context. If it is bound as an object, it becomes a *leaf* node and will not be used for name resolution.

11.2.6 Deleting a Naming Context or Object Binding

To delete a naming context or object binding from the naming hierarchy, right-click on the node and select **Delete** from the pop-up menu. Click the **DELETE** button in the **Warning** dialog box.

Deleting an object binding will not remove the underlying object; only its resolution through the Naming Service will be affected.

When a naming context is deleted, all of its children (naming contexts and object bindings) are also removed from the hierarchy.

11.2.7 Exporting XML

Any portion of the naming hierarchy can be exported to an XML file.

Step 1: Select the naming context that is to be exported. Every node under the selected naming context is exported. To export the entire hierarchy, select the root node.

Step 2: Right-click on the selected naming context and select **Export xml** from the pop-up menu. Alternatively, click the **Save Naming XML** tool bar button.

Step 3: Select a file location and enter a file name in the **Save Naming XML** dialog box.

The XML Export will fail if the tree being exported contains any invalid naming contexts.

The XML export can also be performed from the command line. The command line method is preferred when dealing with very large naming hierarchies (where the export operation may take a considerable time).

11.2.8 Importing XML

An XML file containing a previously-exported section of the naming hierarchy can be re-imported into the naming hierarchy.

The imported branch of the naming hierarchy must be added to an existing naming context node.

As an exported branch of the naming hierarchy can be imported into a completely different location, this is a convenient way to move or replicate large sections of the naming hierarchy

Step 1: Select the naming context that the imported branch will be added under.

Step 2: Right-click on the selected naming context and select **Import xml** from the pop-up menu. Alternatively, click the **Load Naming XML** tool bar button.

Step 3: Use the **Load Naming XML** dialog box to select a previously exported XML file to import.

The contents of the imported file are added to the naming hierarchy in the selected location.

The XML import can also be performed from the command line. The command line method is preferred when dealing with very large naming hierarchies (where the import operation may take a considerable time).

11.2.9 Launching Managers and Browsers

Other OpenFusion graphical tools can be launched from the Naming Service Manager.

The right-click menu options of each object in the naming hierarchy include options for launching browsers and managers specific to that object.

For example, a bound **NotificationSingleton** object has a menu option to launch the Notification Service Manager.

11.2.9.1 CORBA Object Browser

All nodes include an option to launch the CORBA Object Browser. See the *System Guide* for details. The CORBA Object Browser can be used to view naming contexts as well as CORBA objects.

11.2.9.2 Naming Service Manager

Naming contexts include an option to launch a new instance of the Naming Service Manager. The new instance is rooted at the selected naming context.

This is *not* a new instance of the Naming Service. The new manager is simply a new view of the selected portion of the Naming Service.

12 *The Purgable Interface*

The purgable interface is an OpenFusion plugin intended to be used to assist in the determination of whether an object is inactive and can be safely removed.

12.1 Purge Class Plugin

This is a property of the NameSingleton which can be specified through the Administration Manager (the *System Guide*). If used, this property must contain the name of a publicly instantiable Java class that implements the `com.prismt.openfusion.plugin.Purgable` interface.

12.2 Using the Purgable Interface

This interface has one operation:

```
public boolean isPurgable (org.omg.CORBA.Object obj)
```

The class specified as the **Purge Class Plugin** is used to determine whether or not to purge objects from the Naming Service. Typically a client will implement this operation to determine whether its object is persistent or transient and hence may be purged. This service will also check the active/inactive state.

If no class is specified for this property, the `ORBAdapter.isValid` method is used (an object is valid if it is active or not transient). This will successfully determine persistent state for objects created using the OpenFusion framework, but it will not work reliably for foreign objects (objects created in non-OpenFusion environments or on other ORBs).

The following pseudocode illustrates how the interface is used.

```
MyPurgable implements Purgable
{
  if 'mine'
    test & return
  else
    return ! ORBAdapter.isValid(obj)
}
```

The `if 'mine'` is a test which first establishes whether the client object is one that OpenFusion cannot determine the status of (for example, an object from a C++ orb). If it is, then its status is determined by the `test & return` clause; otherwise the

default OpenFusion check of the object's status is performed. If the `isValid()` check is not included, then no checking is performed on OpenFusion created objects and the Naming Service will only purge the client objects.

A close-up, low-angle photograph of a computer keyboard, showing several keys in detail. The keys are white and set against a dark background. A white grid pattern is overlaid on the entire image, creating a sense of depth and structure. The lighting is soft, highlighting the texture of the keys and the grid lines.

APPENDICES

A Command Line Management Tool

The Naming Service Command Line Management Tool, *nsMgrTool*, provides management capabilities which are not available in the GUI-based *Administration Manager*, namely the ability to manage Naming Service instances which reside in diverse OpenFusion installations and domains (refer to the *Administration Manager* section of the *System Guide* for a description of domains and their related directory hierarchy).

An example of a *Domains* hierarchy (as it would appear in the Administration Manager) and its associated directory structure is shown in *Figure 8*:

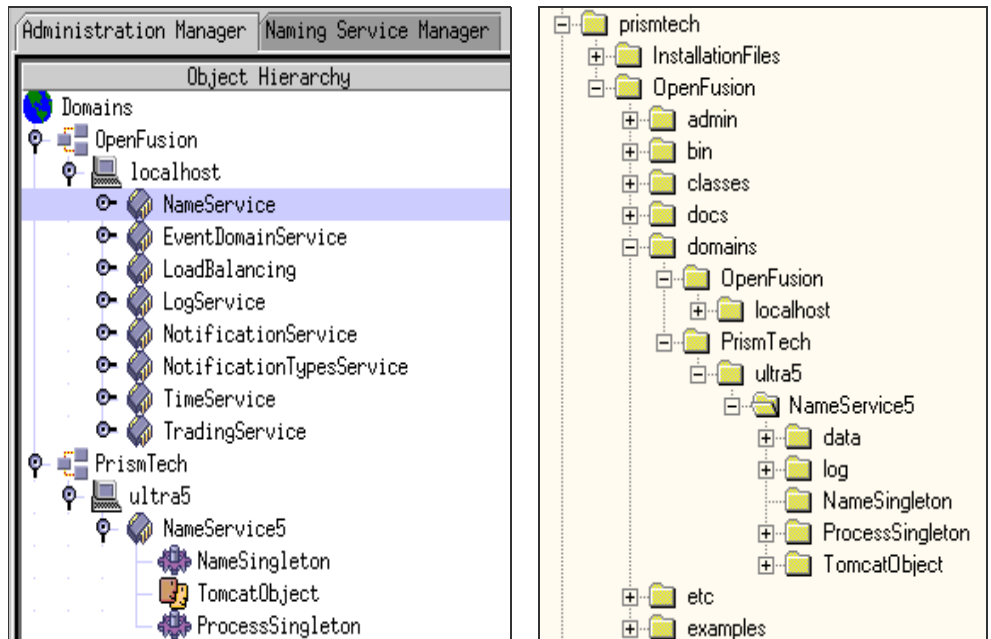


Figure 8: Example Domains Hierarchy and Directories

Features

Operations which *nsMgrTool* can perform include:

- viewing and managing running Naming Service instances located in local or remote installations
- creating and listing the contents of naming contexts for those instances
- binding and unbinding objects to contexts and name objects
- resolving named objects
- destroying contexts and name objects

The Command Line Management Tool, as its name indicates, can be run directly from the command line or as part of a shell or batch script.

Configuration

The command line tool must be able to locate and access either the *NameSingleton.ior* file or a file containing the corbaloc URL for the Naming Service instance it is required to manage. Alternatively, the tool may be provided with the Name Service instance IOR or corbaloc URL. The ***NS_LOCATION*** environment variable is used to pass one of those values to the command line tool. File locations can use either a file or HTTP address (described below).

i The *NS_LOCATION* environment variable must be set ***before*** running the command line tool.

If the *http* protocol is used, then a *Tomcat* object must be added to the specific Naming Service instance to be managed.

A Tomcat object can be added to a Naming Service instance by using the *Administration Manager*'s pop-up menu command **Add | Java Object | TomcatObject** for the instance - see *Tomcat Web Server Integration* in the *System Guide* for details.

Using the *file* Protocol

When using the *file* protocol, set *NS_LOCATION* to:

```
file:///<path>
```

where *<path>* is the complete pathname to the *NameSingleton.ior* file or a file containing the corbaloc URL of the Name Service instance.

WIN When using Windows to access an OpenFusion installation located on a remote host, the remote host's file space must be available as a mapped drive in Windows Explorer (see *Example 1*, Windows version, below).

Example 1 Setting `NS_LOCATION` using the file protocol

Referring to the example installation shown in *Figure 8*, `NS_LOCATION` is set to manage a Naming Service instance called *NameService5*, located on a host called *ultra5*, in the user-defined *PrismTech* domain (note that a mapped drive is used for the Windows version):

UNIX

```
% export NS_LOCATION=file:///var/usr/local/prismtech/
OpenFusion/domains/PrismTech/ultra5/NameService5/
NameSingleton/NameSingleton.ior
```

WIN

```
> set NS_LOCATION=file:///m:/prismtech/OpenFusion/
domains/PrismTech/ultra5/NameService5/NameSingleton/
NameSingleton.ior
```

Using the *http* Protocol

When using the *http* protocol, set `NS_LOCATION` to:

```
http://<host>:<port>/NameService/domains/<domain>/<node>/
<service_name>/NameSingleton/NameSingleton.ior
```

where

`<host>` is the host or machine name of the OpenFusion installation where the Naming Service instance resides

`<port>` is the port address of the Naming Service instance's Tomcat object; the default port address used by Tomcat is *8080*,

`<domain>` is the domain defined under the installation's *Domains* item,

`<node>` is the node name (i.e. the host machine) under the domain,

`<service_name>` is the service name for the Naming Service instance

Example 2 Setting `NS_LOCATION` using the *http* protocol

Referring to the example installation shown in *Figure 8*, `NS_LOCATION` is set to manage a Naming Service instance called *NameService5*, located on a remote host called *ultra5*, in the user-defined *PrismTech* domain:

UNIX

```
% export NS_LOCATION=http://ultra5:8080/NameService/
domains/PrismTech/ultra5/NameService5/NameSingleton/
NameSingleton.ior
```

WIN

```
> SET NS_LOCATION=http://ultra5:8080/NameService/
domains/PrismTech/ultra5/NameService5/NameSingleton/
NameSingleton.ior
```

Using IOR and corbaloc URL

NS_LOCATION can be also set directly to the IOR or the corbaloc URL of the Name Service instance, for example:

UNIX

```
% export NS_LOCATION=corbaloc:iiop:160.45.110.41:38693/
OpenFusion.NameService.NameSingleton/NameSingleton.ior
```

WIN

```
% set NS_LOCATION=corbaloc:iiop:160.45.110.41:38693/
OpenFusion.NameService.NameSingleton/NameSingleton.ior
```

or

UNIX

```
% export
NS_LOCATION=IOR:000000000000002B49444C3A6F6D672E6F72672F436F...
```

WIN

```
% set
NS_LOCATION=IOR:000000000000002B49444C3A6F6D672E6F72672F436F...
```

i

Note

Setting the NS_LOCATION environment variable to the corbaloc URL may require escaping of certain characters.

Running

After setting the NS_LOCATION environment variable, described above, *nsMgrTool* is run using the commands listed in Table 15, *Command Line Management Tool Commands*, as follows:

```
% nsMgrTool <commands>
```

nsMgrTool is located in the *bin* sub-directory where OpenFusion is installed.



Note

- The required OpenFusion Naming Service instance must be running in order for the tool to work.
- The commands must be entered in the order they are listed in *Table 15*.

Table 15 Command Line Management Tool Commands

Command	Description
<code>-h, -?, -help</code>	Displays the list of commands (described below)
<code>-l <path></code>	<p>This lists the contents specified by the path.</p> <p>If the path resolves to a context, its contents are displayed.</p> <p>If the path resolves to an object, then the object is displayed.</p> <p>If no path is specified, then the contents of the root naming context are displayed.</p> <p>The path argument should be in the form of a string.</p> <p>Example: <code>-l Videos/Films</code></p>
<code>-create [path]</code>	<p>This creates a new naming context. If an element of the path does not exist then it is created automatically, e.g. if the path entered was <i>Videos/Sport/Football</i> and only the <i>Videos</i> context existed, then a context would be created for <i>Sport</i> (under <i>Videos</i>) and for <i>Football</i> (under <i>Videos/Sport</i>). If a path is not supplied an unbound Naming Context is created and the object reference string is displayed to the user.</p> <p>Example: <code>-create Videos/Sport</code></p>
<code>-bind <-c -o> <-p path></code> <code><IOR> </code> <code>-bind <-c -o> <-p path></code> <code><-f filename></code>	<p>This option binds a given IOR as a context or object to the specified path. The IOR can either be provided directly or can be read from a file. If the IOR does not resolve to a context, then it is bound as an object.</p> <p>Example: <code>-bind -c -p Videos/Films/ET -f /filedir/filename.ior</code></p>
Items shown in <> are required; items shown in [] are optional; + indicates one or more items	

Table 15 Command Line Management Tool Commands

Command	Description
<code>-resolve [path]</code>	Returns the object reference string for the specified path. If a path is not specified, then the object reference string for the root context is returned. The object reference string is in the format: <i>IOR:0000000000000002B49444C3A6F6D672E67...</i> Example: <code>-resolve Videos/Films</code>
<code>-destroy [-r] <path></code>	This unbinds and destroys the context or object specified by the path. If the path refers to a context the context is only destroyed if it is empty. If it is not empty and the <code>-r</code> argument has not been set, then it is not destroyed and a message is displayed. If the <code>-r</code> argument has been set, then the context and its contents are unbound and destroyed recursively. Example: <code>-destroy -r Videos/Films</code>
<code>-unbind <path></code>	This option unbinds the context or object for the path specified. The unbind will fail if the path relates to a context and the context is not empty. Example: <code>-unbind Videos/Films/ET</code>
Items shown in <code><></code> are required; items shown in <code>[]</code> are optional; + indicates one or more items	

Example 3 Managing a Naming Context

The following example shows a naming object called *test*, located in a Naming Service instance called *NameService5*, being created, bound, resolved, and destroyed. This example assumes that the `NS_LOCATION` environment variable was set as shown in *Example 1* or *Example 2*, above. The example shows the UNIX command line; the tool works identically in Windows, except that Windows users should substitute the forward-slashes with back-slashes *for file paths* only - context paths should *always* use forward slashes.

```
% nsMgrTool -create test
% nsMgrTool -bind -o test/myObject -f /var/user/
application/client.ior
% nsMgrTool -resolve test/myObject > test.ior
% nsMgrTool -destroy test/myObject
```

The `-create` command creates a naming context called `test`. An object is then bound (and automatically created) in this context, using the IOR defined in a file called `client.ior`.

The `-resolve` command, in this example, is used to retrieve the IOR bound to `myObject` and save it to a file call `test.ior`. The `myObject` object is then destroyed.

A close-up, low-angle photograph of a computer keyboard. The keys are white and slightly blurred, creating a sense of depth. A white grid overlay is superimposed on the image, consisting of thin, intersecting lines that form a pattern of squares and rectangles. The overall color palette is a soft, muted blue-grey. The word "INDEX" is printed in a dark blue, sans-serif font in the upper right quadrant of the image.

INDEX

Index

A

Access		Object Reference	50
Naming Service Data	48, 61	Alias	12
Adding		Applet parameters	50
Naming Context	85	Applications	
Address		Java	49

B

BindContextCount (property)	74	Object	57
BindCount (property)	74	Binding a CORBA Object	86
Binding		Binding OpenFusion Services	85
Name	48, 49	BindingIterator Interface	32

C

Cache		Initial	50, 58
Disabling	60	Naming	48, 49
Enabling	60	Root	57
Flushing	61	Sub-	49
Properties	60	Convention	
Caching	9	Name	50
Classnames (property)	81	CORBA	
Clean.List (property)	78	Client	57
Clean.Load (property)	78	Naming Service	48, 52
Clean.PurgeClass (property)	79	Object	50
Client		CORBA Object	
CORBA	57	Binding	86
CORBA Naming Service	50	Corbaloc	7
Java	49	Corbaname	8
JNDI	50, 56, 57	file	8
Component		http	8
Name	50	CosNaming	
Composite name	52	SPI	50, 52
Compound name	52	Count of binds in service (property)	74
Configuration		Count of context binds in service (property)	74
JNDI	50	Count of rebind context in service (property)	73
Naming Service	58	Count of rebinds in service (property)	74
Service provider dependence	50	Count of resolve operations (property)	73
Context		Count of unbinds in service (property)	74
Identifier	61	Create rights, JDBC database	59

Cyclic reference. 52 Cyclics, exporting and importing 42

D

Data
 Accessing Naming Service 48
Database types. 60
DB.LDAP.Password (property) 69
DB.LDAP.SASL (property) 70
DB.LDAP.Security (property) 70
DB.LDAP.Trace (property). 69
DB.LDAP.URL (property) 69
DB.LDAP.User (property) 68
DB.NameDataPersistence (property) 73
DB.ReadCache.Int (property)
 Naming Service 71
DB.ReadCache.Max (property)
 Naming Service 71
DB.ReadCache.Min (property)
 Naming Service 71
DB.WriteBatch (property)
 Naming Service 72
DB.WriteInterval (property)
 Naming Service 72
Delegate 12
Delete
 Naming Context 86
 Object Binding 86
Directory service (objects within). 50
Directory services 49
Driver, JDBC 60

E

Enable Load Balancing 77
Environment
 JNDI. 56, 58
 Parameters 50
Escaping in strings 52
Examples
 Naming Service 25, 28
Exceptions. 44, 62
Export
 XML 86
Exporting and Importing Cyclics 42

F

Factories
 Supplied 54
Factory Classes
 Object. 58
 State 59
Fail-over 14
Federation 5, 48, 49, 54
Flushing, cache 61

G

Generated UUID 61, 62 Graph (hierarchy) 49

H

Hashtable. 50, 56, 58 Hierarchy of naming contexts 48, 49

I

Identifier		Interoperable Naming Service (INS)	6
Context	61	IOR	7
UUID	61	IOR File Name (property)	67, 80
Importing XML	87	IOR Name Service (property)	68, 80
Initial context	50, 58	IOR Name Service Entry (property)	79
INITIAL_CONTEXT_FACTORY	58	IOR URL (property)	67, 79
INS (Interoperable Naming Service)	52	IOR.File (property)	67, 80
Instrumentation	13	IOR.URL (property)	67, 80
Naming Service Properties	73		

J

Java		Properties File	76
Applications	49	Root ID	77
Client	49	Root ID option	77
Objects	50, 53	Specification	52
Java Naming & Directory Interface (JNDI)	8, 38	Standard properties	58
javax.naming.Context (interface)	52	Tutorial (Sun Microsystems)	56
JDBC		JNDI ContextFactory Cache Flush Interval (properties)	75
Create rights	59	JNDI ContextFactory Cache Maximum Size (property)	75
Database type	60	JNDI ContextFactory Cache Minimum Size (property)	76
Database URL	60	jndi.properties file, location of	58
Database user	59	jndiCtxCacheInt (property)	75
Driver	60	jndiCtxCacheMax (property)	75
JMX		jndiCtxCacheMin (property)	76
Instrumentation Properties	73	JNDIID (property)	77
JNDI		JNDIObject Interface	35
Client	56	jndiOFPropertiesFile (property)	76
Configuration	50	jndiPropertiesFile (property)	76
Environment	56, 58		
Object	57		
OF Properties File	76		
Properties	50, 58		

L

LDAP	38, 39, 68	LDAP URL (property)	69
LDAP (Lightweight Directory Access Protocol)	49	LDAP User (property)	68
Server URL	62	Lightweight Directory Access Protocol	68
LDAP Password (property)	69	Load	
LDAP SASL Mechanism Names (property)	70	Naming XML	85
LDAP Security (property)	69	Load Balancing	
LDAP Trace (property)	69	Concepts	11

Implementation	12	LoadBalancing (property)	77
Policies	13, 34	LoadBalancingFactory	12
Load Balancing Plugin (property)	81	LoadBalancingFactory Interface	32
Load Balancing Timeout (property)	81	LoadBalancingFactorySingleton Configuration	79
LoadBalancer Interface	32	Log file	56
LoadBalancerPlugin Interface	34, 35		

M

Memory Management	40	Messages, warning	57
Concepts	10	Meta-characters	52
Memory-based persistence	53		

N

Name		CORBA	48, 52
Binding	48, 49	Corbaloc	7
Component	50	Corbaname	8
Composite	52	file	8
Compound	52	http	8
Conventions	50	Data, accessing	56
Resolving	48, 49	example	
String	50	BindingIterator	20
Stringified	52	client	28
Syntax	50, 52	LoadBalancer, customizing	25
Validity checks	53	LoadBalancer, manipulating objects	24
Name (interface)	50	LoadBalancer, using	24
Name Components	5	LoadBalancingFactory, using	23
NameService option		naming context contents, accessing	19
Purge Class Name	39	naming context, extension	21
Purge on Load	39	Interoperable	52
NameSingleton Configuration	66	IOR	7
Naming Context	3	JNDIObject Interface	35
Adding	85	LoadBalancer Interface	32
Deleting	86	LoadBalancerPlugin Interface	34, 35
Naming context	48, 49	LoadBalancingFactory Interface	32
Naming context hierarchy	48	Manager	82
Naming Data Storage Type	73	Naming context	3
Naming scheme	52	NamingContext Interface	30
Naming Service	57, 58, 61	NamingContextExt Interface	31
Access to	61	URL	6
BindingIterator Interface	32	Naming System	49, 50
Configuration	58, 66	Federation	49
Contexts	3	NamingManager (interface)	50

O

Object		Object Binding	86
CORBA	50	Deleting	86
Factories	58	Object Cache Maximum Size option	40
Java	50, 53	Object Cache Minimum Size option	40
JNDI	57	Object Cache Purging Interval option	40
Non-CORBA	57	Object Purging option	40
Reference	53	Object.Name (property)	66, 79
Referenceable	53	OBJECT_FACTORIES	58
Serializable	50, 53	Obtaining the Root Context	17
Stored in directory	50	OMG Standard API Definitions	30

P

Parameters		Properties	
Applet	50	JNDI	50
Environment	50	OpenFusion SPI	59
Persistence		System	50
Across sessions	61	Purgable interface	90
JDBC	56	Purge Class Name option	39
Memory	53	Purge Class Plugin	78, 90
Multiple forms of	3, 9	Purge on List (property)	78
Persistence Options		Purge on Load (property)	78
NameSingleton	70	Purge on Load option	39
Policies	11	Purging	
Prefix		Concepts	10
Properties	59		

Q

Quoting in strings	52
------------------------------	----

R

Read Cache Flush Interval (property)	71	Replication	14
Read Cache Maximum Size (property)	71	Resolve a name	48, 49
Read Cache Minimum Size (property)	71	Resolve Name (property)	68, 80
ReBindContextCount (property)	73	ResolveCount (property)	73
ReBindCount (property)	74	ResolveName (property)	68, 80
Reference		Resolver (property)	79
Cyclic	52	Restrictions	
Stored	50	SPI	50
Referenceable object	53	Root context	57
References to objects	53	Root Context, obtaining	17

Root UUID 56

S

Save
 Naming XML 85
 Serializable object 50, 53
 Service provider 49
 LDAP 50
 OpenFusion 50
 Services, directory 49
 SID (Service ID) 56, 61
 Singletons
 LoadBalancingFactorySingleton 79
 NameSingleton 66
 SNMP 13
 Specification
 JNDI 52
 SPI
 CosNaming 50, 52
 Restrictions 50
 Starting
 Naming Service Manager 82
 STATE_FACTORIES 59
 String
 Escaping within 52
 Name 50
 Quoting in 52
 Stringified name 52
 Stringified Names 6
 Subcontext 49
 Supplied factories 54
 Syntax
 Name 50, 52
 System Master (property) 79
 System, Naming 49

T

Timeout (property)
 Load Balancing Singleton 81
 Tool Bar
 Naming Service Manager 85

U

UnBindCount (property) 74
 URL 6
 JDBC database 60
 UUID (Universally Unique Identifier) 56, 61
 Generated 61, 62

V

Validity checks on names 53
 View Non-Corba Objects (property) 77
 ViewNonCorba (property) 77
 VisiBroker 14

W

Warning messages 57
 Write Cache Batch Size (property)
 Naming Service 72
 Write Cache Write Interval (property) 72

X

Xbootclasspath 86
 XML Export and Import 41