OpenFusion®
CORBA Services
Version 4.1
System Guide

PRISMTECH

OpenFusion®
CORBA Services

System Guide

Part Number: OFCOR-SYSG-41

Doc Issue 19, 13 July 2004

Notices

Copyright Notice

Preface

About the System Guide

The *System Guide* is included with the OpenFusion CORBA Services' *Documentation Set*. The *System Guide* provides:

- general information necessary to develop, use, configure and manage the OpenFusion Services and its related framework
- information about the *OpenFusion Graphical Tools*
- information about common service configuration, properties, and instrumentation

Configuration and property information specific to an individual service or interface is provided in that service's or interface's service guide.

The *System Guide* is intended to be used with the individual service and interface guides, and with other OpenFusion documents included with the product distribution: A complete list of documents, comprising the OpenFusion CORBA Services *Documentation Set*, is included in the *Product Guide*.

Intended Audience

The *System Guide* is intended to be used by users, developers, and administrators who wish to integrate or manage the OpenFusion CORBA Services into or with their applications and products. Readers who use this guide should have a good understanding of the relevant programming languages (e.g. Java, IDL) and the relevant underlying technologies (e.g. J2EE, CORBA).

Conventions

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

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

Item contains helpful hint or special information.

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

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

Hypertext links to WWW and other internet services are shown as *blue italic underlined.*

On-Line (PDF) versions of this document: Items shown as cross references to other parts of the document, e.g., *Contacts* on page vi, behave as hypertext links: readers can jump to that section of the document by clicking on the cross reference.

```
% 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, full width boxes (to allow for standard 80 column wide text), as shown below:

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

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

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

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: Indicates that this item is a step or stage of completing a task by a user.

Contacts

Information and technical support can be obtained at the following contact points.

Corporate Headquarters	European Head Office
PrismTech Corporation 6 Lincoln Knoll Lane Suite 100 Burlington, MA 01803 USA	PrismTech Limited PrismTech House 5th Avenue Business Park Gateshead NE11 0NG UK
Tel: +1 781 270 1177 Fax: +1 781 238 1700	Tel: +44 (0)191 497 9900 Fax: +44 (0)191 497 9901

Web: *http://www.prismtechnologies.com*

General Enquiries: *info@prismtechnologies.com*

Support Enquiries: *http://www.prismtechnologies.com/Contacts*

Contents

Table of Contents

Notices iii

Preface v

About the System Guide v

Contacts vi

List of Figures xv

List of Tables xvii

Introduction 1

Who Should use this Guide 3

What the System Guide Includes 3

Common System Operations 5

1 Running Servers 7

2 OpenFusion Graphical Tools 11

2.1 Overview 11

2.2 The Browser Framework 11

Starting the Administration Manager 12

Command Line Switches 12

Domain Configuration Parameters 13

2.3 Administration Manager 13

Object Hierarchy 14

Tool Tips 15

Object Hierarchy Icons 15

Status 17

Starting the Services 18

Extending the Object Hierarchy 19

Locking 22

Locking Nodes 23

Locking Properties 23

Restoring Services and Singletons 24
Properties 25
Type 25
Mandatory 26
Accessibility 27
Conditional Properties 27
Assigning Values to Properties 27
Actions that Can be Performed on Properties 28
Signals 30
User Identity 30
Service Log 30
Memory Profiler 31
Tool Bar Options 32

2.4 **The CORBA Object Browser 34**

2.5 **Distributed Installation Configuration 36**
The Central Configuration Host 36
Using a Shared File System 38
Set up the Central Host 38
Implementation Repository 38
Environment Properties 39
Using a Web Server 39
Set up the Central Host 39
Configure Remote Singletons 39
Set the Central Host Properties 40
Set up the Remote Machine 41
Working with Central Configuration 42

2.6 **Tomcat Web Server Integration 42**
Deployment of Web Archives 42
Security 43
Deploying Java Server Pages 43
Configuration 43
Testing the Tomcat Installation 47

3 **Common Configuration Properties 49**

3.1 **Overview 49**

3.2 **Persistence Properties 49**

3.3 **Logging Properties 55**

3.4 **CORBA Properties 60**

3.5 **Security Properties** 66
3.6 **Java Properties** 66
3.7 **System Properties** 68
3.8 **Common Singleton Properties** 69
3.9 **Administration Manager Properties** 71
CORBA Properties 72
Configure Properties 73
General Properties 74

4 **Instrumentation** 77

4.1 **Overview** 77
Manageable Resources 77
Object Counters 77
4.2 **SNMP Agent** 78
Configuring the SNMP Agent 78
Notifications 82
Trap Hosts File 82
Starting the SNMP Agent 83
Stopping the SNMP Agent 83
OpenFusion MIBs 83
4.3 **CORBA Process Interface** 84
Using the Process Interface 84
Example Program 86

5 **Service Portability** 89

5.1 **Portability Classes** 89
The ORBAdapter Class 90
ORB Initialization 90
ORB Shutdown 90
Object Information 91
Object Stringification 91
Service Resolution 91
The ObjectAdapter Class 92
Initialization 92
Object Creation 92
Object Identity 94
Multiple Object Identity 95

Object Deactivation. . . . 95
Object Destruction 95
Object Reactivation. . . . 96
Object Existence 96
Object References 96
Object Implementations 96
Persistent Object State 97
Running a Server. . . . 98
Restrictions 98
Recommendations. . . . 98
The DynAnyFactory Class 99
Creation Operations 99
Implementing an Interface. . . . 100
Persistent Servers. . . . 101

5.2 **Running User Defined Clients and Servers. . . . 102**
Resolving Services 102
Jar Files 103
Using OpenFusion Run Scripts 103
Command Line Format 104

5.3 **OpenFusion Java IDL Compilation. . . . 104**

5.4 **C++ Support. . . . 106**

6 **Configuring Persistent Storage 107**
Configuring a JDBC Data Source 107
Oracle. . . . 109
Oracle Thin Drivers 109
Oracle OCI Drivers 110
Sybase 110
Informix. . . . 111
SQL Server 112
hsqldb. . . . 112
Create an hsqldb Instance 112
Configure OpenFusion Services to Run with hsqldb Persistence. . . . 113
hsqldb in Client/Server Mode 113
Restoring Data. . . . 116

7 **Command Line Tools 117**

7.1 **IOR Decoder 117**

7.2 **Administration Manager Tool 117**

7.3 Configuration Generator 119

Security Service 121

8 Description 123

8.1 Overview 123

8.2 Concepts and Architecture 123
Securable Objects 123
Authentication 124
ACLs 124
Groups 125
Mapping Principals 126
LoginModule 126

9 Using Specific Features 127

9.1 Securing an Interface or Method 127
Excluding Methods from the Security Manager 128

9.2 Creating ACL Groups 129

9.3 Creating Principal Mappings 130

9.4 Supplying Authorised Credentials 131

10 Security Configuration 133

10.1 Configuring a Secure OpenFusion Service 133
Security Administration Manager Properties 133

10.2 Configuring a Secure Client 136
Security Configuration File Properties 136
Example Security Configuration File 139

11 Security Administration Manager 141

11.1 Starting the Security Administration Manager 141

11.2 Using the Security Administration Manager 142
Object Hierarchy 143
Security Hierarchy Options 144
Excluding Methods from the Object Hierarchy 145
Tool Bar Buttons 145
Principals Panel 145

Operations 146
Implementing Security Configuration Changes 149
Interfaces 149

Appendices 151

A XML Configuration Files 153

Overview 153
The Object Hierarchy 153
Domains and Nodes 155
Services 156
Singletons 157
Java Objects 157
XML Templates 158
GroupName 160
CategoryName 160
Dependencies 160
Conditional 160
XML Schema 160
Command-line Configuration 161

B Log Messages 163

Overview 163
Using a Pattern Layout 163
Conversion Characters 164

C Managing Java Objects 167

Overview 167
Creating the Java Object 167
Describing the Java Object in XML 167
Defining Properties for the Java Object 168
The Object Hierarchy 169

Glossary 171

Index 185

List of Figures

Figure 1 OpenFusion Browser Framework 11
Figure 2 The Object Hierarchy 14
Figure 3 Viewing Tool Tips 15
Figure 4 Adding a Node 20
Figure 5 Adding a Java Object 21
Figure 6 Deleting a Singleton 22
Figure 7 Administration Browser Properties Pane 25
Figure 8 A Signal Button 30
Figure 9 Memory Profiler 32
Figure 10 Starting the CORBA Object Browser 34
Figure 11 CORBA Object Browser 35
Figure 12 Remote Nodes 37
Figure 13 Tomcat Default Web Page 48
Figure 14 ACL UML Model 125
Figure 15 The Security Administration Manager 142
Figure 16 The Security Object Hierarchy 143
Figure 17 The Principals Panel 146
Figure 18: Object Hierarchy and Directory Structure 154
Figure 19: Domains Directory Tree 155
Figure 20: Templates Directory Tree 159

List of Tables

Table 1 Object Hierarchy Icons 16
Table 2 Service Status Icons 17
Table 3 Property Types 26
Table 4 Mandatory Properties 27
Table 5 Tool Bar Buttons 33
Table 6 CORBA Object Browser Tool Bar Buttons 36
Table 7 Services' Access Names 84
Table 8 ObjectAdapter Object Creation Flags 94
Table 9 Jar Files 103
Table 10 ORB Definitions 105
Table 11 IDL Includes 105
Table 12 SQL Scripts 108
Table 13 Security Object Icons 144
Table 14 Security Administration Manager Tool Bar 145
Table 15 DTD Files 160
Table 16 Conversion Characters 164

Introduction

Who Should use this Guide

This guide can be used by both system administrators, responsible for configuring and managing the OpenFusion installation, and software developers.

What the System Guide Includes

The System Guide covers the following topics:

- how to run the OpenFusion Services
- a description of the Administration Manager, which is used to configure the OpenFusion Services and launch the Service Managers
- details of common properties
- how to configure and use remote JMX Instrumentation
- service portability issues (portability classes, user-defined clients and servers, OpenFusion IDL compilation, and C++ Support)
- how to configure a JDBC data source to provide a persistent storage mechanism for OpenFusion Services
- details of various command-line tools provided with OpenFusion
- how to configure and use the OpenFusion Security Service to apply access control to CORBA Services and Java Objects

Appendix A, XML Configuration Files describes how the Service configuration files are stored in the OpenFusion installation. This appendix is only relevant to developers who want to edit the configuration files programmatically; configuration should normally be performed through the Administration Manager, where the configuration files are hidden from the user.

Appendix B, Log Messages describes how to use pattern layouts to configure log messages for any Service.

Appendix C, Managing Java Objects describes how to configure user-defined Java Objects to make them available for management through the Administration Manager.

i The full text of this guide is also available as on-line help, accessible from the Administration Manager.

Common System Operations

1 Running Servers

This section describes the ways in which the OpenFusion CORBA Services can be started, either from the Administration Manager or from command-line scripts.

---

i Explanatory Note

The terms *server* and *service* are often used interchangeably. However, there is a subtle and important distinction:

- a *service* is a set or collection of features, functions, etc. (e.g. the OMG Notification Service or OMG Naming Service)
- a *server* is an entity which provides services and makes their features, etc. available for use

A server must be running before a service can be provided. A single server can provide one or more services.

Starting Servers from the Administration Manager

Step 1: Running the ORB Daemons

Start or check that the appropriate ORB daemon is running, if required: please refer to the *OpenFusion Installation Guide* and your ORB's documentation.

i When servers are run on fixed ports, an ORB daemon may not be necessary.

Step 2: Configure the System

OpenFusion can be configured using the **Administration Manager**. Before running the Administration Manager, ensure that the CLASSPATH environment variable includes the appropriate ORB jar files as described in the *OpenFusion Installation Guide*.

Step 3: Starting the Administration Manager

WIN The Administration Manager can be started by selecting **Start | Programs | OpenFusion | Administration Manager** from the *taskbar* or by running the following batch file:

```
>  <install_dir>\bin\manager
```

where *<install_dir>* is the OpenFusion installation directory.

UNIX Use the following script to start the Administration Manager:

```
%  <install_dir>/bin/manager
```

where <*install_dir*> is the OpenFusion installation directory.

When the OpenFusion Naming Service is used with Orbix 2000 on Unix, the Administration Manager should be started with a `-x` parameter as follows:

```
%  <install_dir>/bin/manager -x
```

This ensures that the JDK CosNaming classes are not in the CLASSPATH and prevents problems with the Naming Services Manager.

Step 4: Starting and Stopping Servers

The Administration Manager can be used to manage servers.

To start a server, right-click on the *service* name and select *Start* from the pop-up menu.

If the *Name Service Entry* field is filled in at the time of system configuration, the **`NameService`** server must be started first.

Starting Servers from the Command Line

Individual servers can also be controlled from the command line using a server script.

No servers can be started from the command line until the XML configuration files have been fully populated using either the Administration Manager or the command line Administration tool. See Section 7.2, *Administration Manager Tool*, on page 117 for details.

The server script takes a command-line parameter followed by one or more server names. For example, the *XYZServer* (where *XYZServer* is the server you want to start) can be started from the command line with:

```
%  <install_dir>/bin/server -start XYZServer
```

where <`install_dir`> is the OpenFusion installation directory.

Alternatively, the server can be run in the foreground, rather than the background, with:

```
% <install_dir>/bin/server -run XYZServer
```

It is also possible to start a server so that it is automatically restarted on any abnormal exit. To do this, use the `-restart` option:

```
% <install_dir>/bin/server -restart XYZServer
```

This is a blocking operation, so to stop the server either the Administration Manager or another shell should be used.

In order to stop a running server, use:

```
% <install_dir>/bin/server -stop XYZServer
```

To check whether a server is already running, use:

```
% <install_dir>/bin/server -status XYZServer
```

Use the `-status` command without specifying a server name to show the status of all supported servers.

To start a server and wait for startup (that is, use a blocking call to the server), use:

```
% <install_dir>/bin/server -exec XYZServer
```

The `-x` option runs the JVM using the `-Xbootclasspath` flag to avoid problems with CosNaming classes supplied as part of the JDK. For example, use:

```
% <install_dir>/bin/server -start -x XYZServer
```

All the servers configured to run on the local node can be controlled via the node script. This node script has `-start`, `-stop`, `-status`, and `-x` options.

To start every server on the local node, use:

```
% <install_dir>/bin/node -start
```

To stop every server on the local node, use:

```
% <install_dir>/bin/node -stop
```

To check whether servers on the local node are running, use:

```
%  <install_dir>/bin/node -status
```

Status and other information about servers residing on nodes other than the default node (*localhost*) can be obtained by altering the `OF_Node_URL` property.

For example, to set `OF_Node_URL` to a valid, existing node called *`mynode`* under *`OpenFusion`*, change the default value from:

`<install_dir>/domains/OpenFusion/localhost`

to:

`<install_dir>/domains/OpenFusion/mynode`

`OF_NODE_URL` is located in:

UNIX **`bin/.javaenv`** or

WIN **`bin\server.bat`**

2 OpenFusion Graphical Tools

2.1 Overview

The OpenFusion Graphical Tools can be used to configure, test, and manage the OpenFusion CORBA Services (OpenFusion).

2.2 The Browser Framework

All *service managers* and *browsers* use the same *browser framework*, which provides common menus and tool bars. Common functions are standardised between individual browsers so each new browser presents a gentle learning curve.

New browsers can be opened to manage individual service elements. Each new browser opens in the browser framework as a new panel identified by a named tab. Switch between different browsers by clicking on the browsers' name tabs.

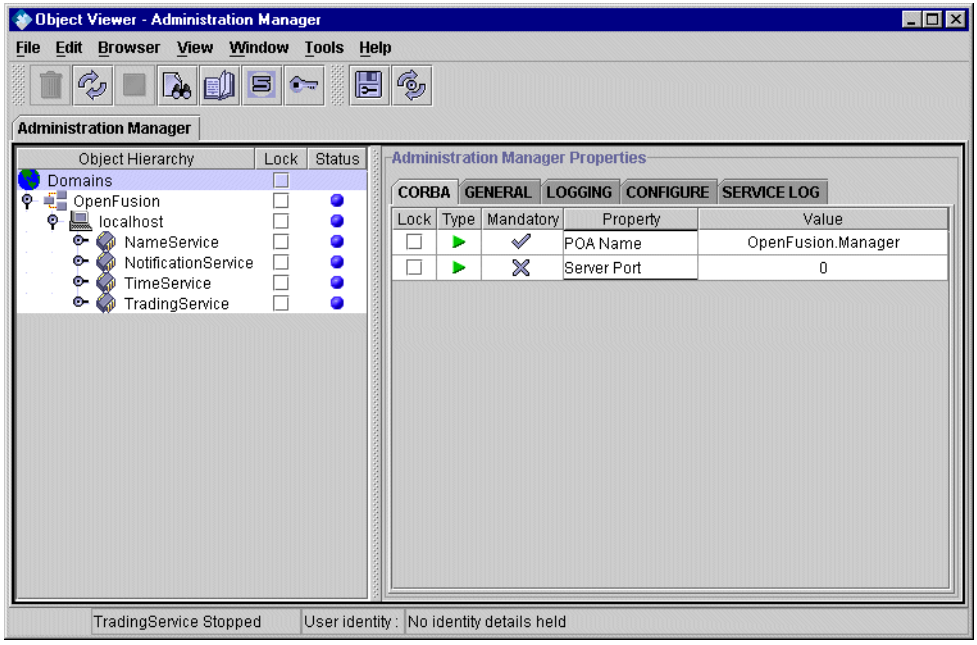

Figure 1 OpenFusion Browser Framework

The hub of the browser framework is the Administration Manager. This is where OpenFusion can be configured and controlled.

Starting the Administration Manager

WIN The Administration Manager can be started by selecting **Start | Programs | OpenFusion | Administration Manager** from the *taskbar* or by running the following batch file:

```
>  <install_dir>\bin\manager
```

where <*install_dir*> is the OpenFusion installation directory.

UNIX Use the following script to start the Administration Manager:

```
%  <install_dir>/bin/manager
```

where <*install_dir*> is the OpenFusion installation directory.

When the OpenFusion Naming Service is used with Orbix 2000 on Unix, the Administration Manager should be started with a `-x` parameter as follows:

```
<install_dir>/bin/manager -x
```

Command Line Switches

The following command line switches can be used when starting the Administration Manager:

`-noorb`

Start the Administration Manager without a connection to an ORB. This does not allow Services to be started or stopped, and does not report any status information from Services. The Manager can be used purely as an XML configuration tool.

`-port`

The Administration Manager attempts to start on the port specified in the *Server Port* property on the Administration Manager properties panel.

`-remote`

The Administration Manager will only show configurations which can be managed from the node the manager is started from.

`-help`

Gives usage information on the command line switches. The switches `--help` and `-?` can also be used.

Domain Configuration Parameters

By default, configuration information for an OpenFusion CORBA Services installation is located under the OpenFusion installation directory. If this directory is changed (to allow configuration from a remote host, for example), environment variables must be set to specify the locations of the correct directories.

The three environment variables listed below point to the configuration directories at different levels of the OpenFusion hierarchy. (The hierarchy is explained in *Object Hierarchy* on page 14.) These variables must be expressed as valid *URL* strings.

Example:

WIN
```
set OF_DOMAINS_URL=file://C:\Program Files\PrismTech\OpenFusionV4\domains
```

UNIX
```
export OF_DOMAINS_URL=file:///usr/users/PrismTech/OpenFusionV4/domains
```

`OF_DOMAINS_URL`

The location of the top-level *`domains`* directory, expressed as a file-based URL. This defaults to:

```
file://<INSTALL>/domains
```

Where `<INSTALL>` is the OpenFusion installation directory.

`OF_DOMAIN_URL`

The location of the *`OpenFusion`* domain directory, expressed as a file-based URL. This defaults to:

```
file://<INSTALL>/domains/OpenFusion
```

`OF_NODE_URL`

The location of the *node* directory, expressed as a file-based URL. This defaults to:

```
file://<INSTALL>/domains/OpenFusion/localhost
```

2.3 Administration Manager

The Administration Manager is used to configure the OpenFusion Services and to manage (stop and start) the Services. The Administration Manager is extensible and can be used to configure user-created Java Objects.

The Administration Manager has two panes:

- The left-hand pane shows the *Object Hierarchy* of the OpenFusion installation.
- The right-hand pane shows the configurable properties for the object selected in the hierarchy.

Object Hierarchy

The left-hand pane of the Administration Manager shows all services, Singletons, and Java Objects, and the domains that contain them, in a tree structure, as shown in *Figure 2*.

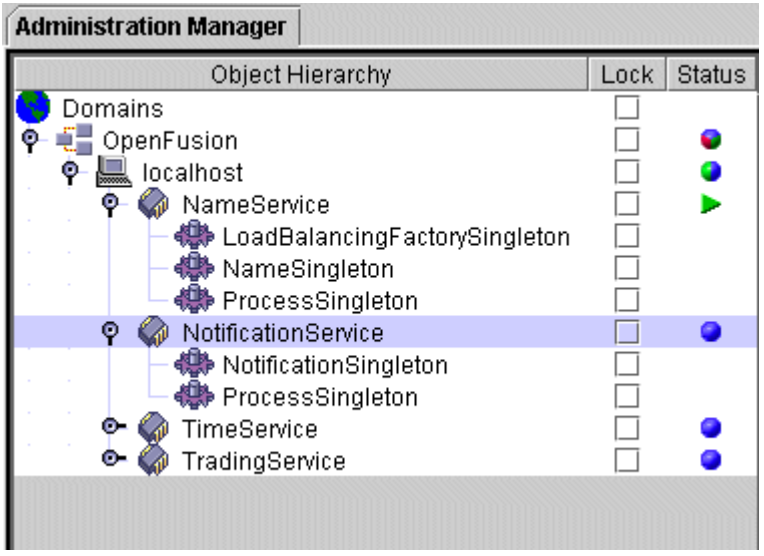

Figure 2 The Object Hierarchy

Domains are high-level organisational units. The default hierarchy shows all installed OpenFusion Services under a single domain. This hierarchy can be restructured and extended as required, as described in *Extending the Object Hierarchy* on page 19.

Nodes represent actual hardware devices within the domain and are given the name of the machine they represent. The default hierarchy shows the local machine as a node called *localhost*. This node can be deleted and replaced with a node which uses the computer name, if required. There is no difference, functionally, between using the machine name and the name *localhost*.

A Service is a logical group of Singletons and Java Objects that are controlled together. Groups of Services can be started together at the node or domain level.

Each of the OpenFusion Services is represented by a Service node in the Object Hierarchy. New Services can be created, which can contain different permutations of Singletons and Java Objects.

Every Service should contain a *ProcessSingleton*. The *ProcessSingleton* is the object which allows the Service to be controlled from the Administration Manager.

Every OpenFusion CORBA object is represented by a Singleton in the *Object Hierarchy*. Additional Singletons and Java Objects can be added to the hierarchy, either to existing OpenFusion Services or to user-created Services.

Java Objects should be co-located with CORBA Singletons in a Service. A Service which contains Java Objects and no Singletons cannot be monitored correctly for its status and cannot be controlled from the Administration Manager.

See *Extending the Object Hierarchy* on page 19 for details of adding nodes, Services, and Singletons. See Section 2.5, *Distributed Installation Configuration*, on page 36 for details of how multiple nodes can be managed from a central host.

Tool Tips

Every node in the *Object Hierarchy* has an associated tool tip which provides information about that node. To see the tool tip, hover the mouse pointer over the node, as shown in *Figure 3*.

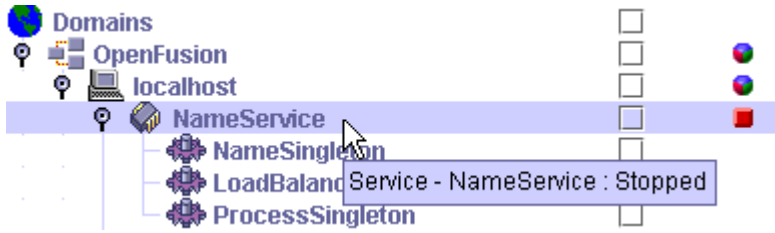

Figure 3 Viewing Tool Tips

The tool tip gives the type of node (Domain, Service, Singleton, etc.), the name of the node, and the status of the node (see *Status* on page 17).

Object Hierarchy Icons

Different nodes in the *Object Hierarchy* are identified by different icons. These icons are shown in *Table 1*.

Table 1 Object Hierarchy Icons

Icon	Object
	*Root node* No actions can be performed at this level of the hierarchy, other than adding new domains and saving. Administration Manager properties can be amended at this level.
	*Domain* An organisational grouping.
	*Node* A hardware device which runs OpenFusion Services. *localhost* is the default node. Other devices can be added as nodes.
	*Service* Singletons and Java Objects are grouped under Services and are started and stopped together at the Service level. Service-level properties can be set in the right-hand pane.
	*Singleton* Represents an underlying CORBA object. Properties can be set in the right-hand pane.
	*Java Object* Represents an underlying Java object. Properties can be set in the right-hand pane.
	*Unlicenced Singleton* Singletons cannot be started if a valid license file is not present.
	*Unlicenced Java Object* Java Objects cannot be started if a valid license file is not present.

Status

A coloured icon in the *Status* column of the tree view shows the current status of each service.

Parent nodes can show an indeterminate status. This is used when the node's child nodes have mixed status (for example, some are stopped and some are started). Status icons are shown in *Table 2*.

Table 2 Service Status Icons

Icon	Status
	*Running* The service is running normally.
	*Stopped* The service is stopped.
	*Starting* (yellow) The service is in the process of starting. This icon will be displayed while the browser is polling the server to determine if the service has started. The icon will eventually change to show *Running* (if the service starts normally) or the state of the service prior to the start command being issued (if the request times out without starting the service).
	*Unknown* (blue) The status of the object hierarchy node is unknown (has never been started). This icon is displayed when the browser is first loaded or a node is restored.
	*Indeterminate* (mixed red/green/blue) This icon is used for an object hierarchy node when its child nodes have a mix of different statuses, or when one of the child nodes has an indeterminate status.

Table 2 Service Status Icons (Continued)

Icon	Status
	*Indeterminate* (mixed red/green) This icon is used for an object hierarchy node when its child nodes are a mix of running and stopped status.
	*Indeterminate* (mixed green/blue) This icon is used for an object hierarchy node when its child nodes are a mix of running and unknown status.
	*Indeterminate* (mixed red/blue) This icon is used for an object hierarchy node when its child nodes are a mix of stopped and unknown status.

The status is also shown in the tool tip for each Service.

Starting the Services

Services can be started or stopped individually.

To start a service, right-click on the service name in the *Object Hierarchy* and select **Start** from the pop-up menu.

To stop a running service, right-click on the service name in the *Object Hierarchy* and select **Stop** from the pop-up menu.

Some properties cannot be modified after a service is started, so the service must be properly configured beforehand.

To start (or stop) a collection of services, right-click on the services' parent node and select **Start** (or **Stop**) from the pop-up menu. Services are started and stopped in the order they appear in the *Object Hierarchy.*

i

If an object hierarchy node cannot be started (that is, the option is disabled), then it is likely that it has an unlicenced child node or that the Node object hierarchy node is not valid for the hardware device that the Administration Manager is being invoked from, noting that `localhost` is always valid.

Before a service is started, the system automatically saves the service configuration. If there are mandatory properties for the service which have not been completed, the service will not start, a warning will be displayed, and the missing property will be noted in the browser log.

Extending the Object Hierarchy

The *Object Hierarchy* can be extended with new Domains, Nodes, Services, Singletons, and Java Objects.

The default Object Hierarchy shows all installed OpenFusion Services grouped under a single node under a single domain. Extending the Object Hierarchy provides a more flexible approach to managing the OpenFusion installation.

For example it may be necessary to run a particular Service in different configurations at different times. Instead of re-setting all the properties for the Service when it is run in another configuration, copies of the Service could be created under different domains. Each could be configured with the required properties. Then, to switch between configurations, simply stop one domain and start the other.

Another use for multiple domains would be to set up different combinations of Services that should be started together.

Nodes can be used to represent different servers within the domain.

New Services can be created to manage user-created Java Objects.

Other ways of grouping domains, nodes, and services will suggest themselves based on the way OpenFusion is used in a particular installation.

Adding Nodes

Domains and Services are simply organisational groupings and can be added without restriction.

Nodes represent hardware devices running CORBA Services. A node can only be managed through the Administration Manager if it is given a valid device name (`localhost` is always valid).

To add a node, right-click on the parent node and select **Add Domain**, **Add Node**, or **Add Service** (the command depends on the level you are adding to) from the pop-up menu, as shown in *Figure 4*.

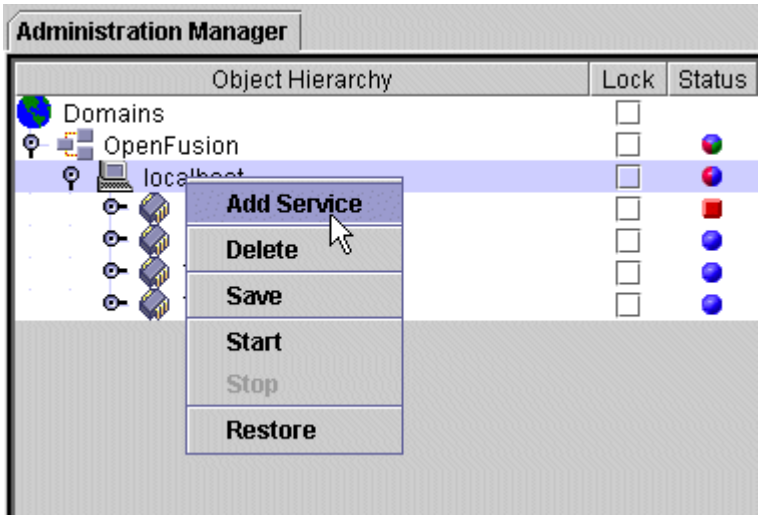

Figure 4 Adding a Node

Enter a name for the node. Node names must be unique within the scope of their parent nodes. You can re-use a name if it is under a different parent. Names can only contain alphanumeric characters.

When a new Service is added, a `ProcessSingleton` is automatically created beneath it. This allows the service to be managed by the Administration Manager: it is not recommended that users have a service without a `ProcessSingleton`.

Adding Singletons and Java Objects

Singletons and Java Objects can only be added under Service nodes of the *Object Hierarchy*.

Singletons and Java Objects in the *Object Hierarchy* are representations of underlying objects and so only objects which already exist are available for adding to a Service.

The Resolve Name of the Naming Service Singleton must be unique within the whole Domain, not just within the scope of the parent node (`localhost` by default). The Resolve Name must be unique to avoid the possibility of two objects attempting to register themselves in the NameService with the same name.

To add a Singleton or Java Object, right-click on a Service node and select **Add** from the pop-up menu. Select either *Singleton* or *Java Object* to see a menu of available objects. Objects that cannot be added to the Service are greyed-out. Select the required object from the list, as shown in *Figure 5*.

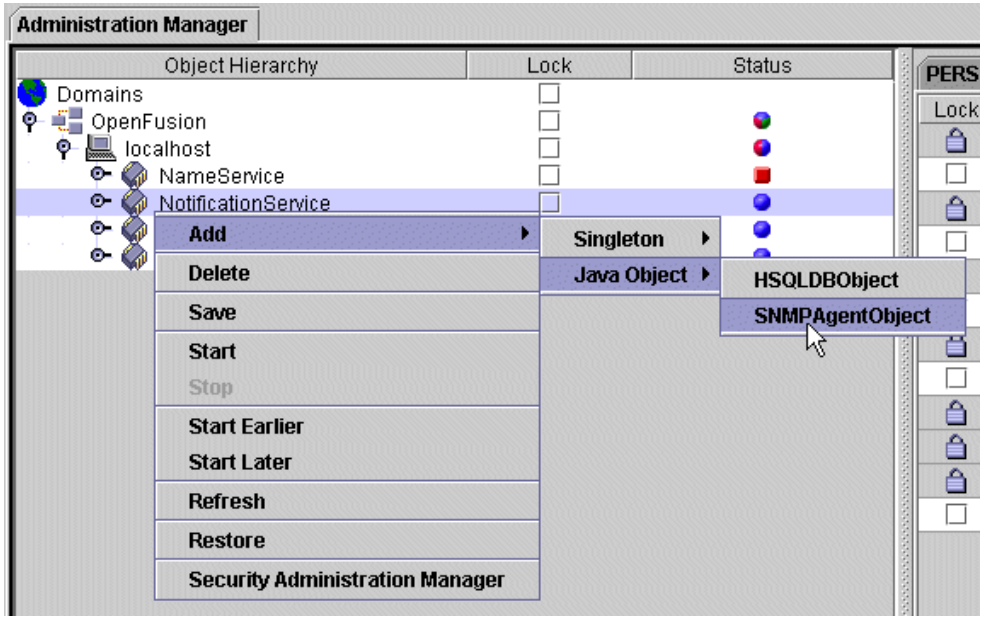

Figure 5 Adding a Java Object

It is not possible to have two instances of the same Singleton or Java Object under one Service.

If the same Singleton or Java Object is added under two different Services, they are two separate instances and properties changed in one instance will not affect the other instance.

Deleting Nodes

To delete a node from the *Object Hierarchy*, right-click the node and select **Delete** from the pop-up menu, as shown in *Figure 6*.

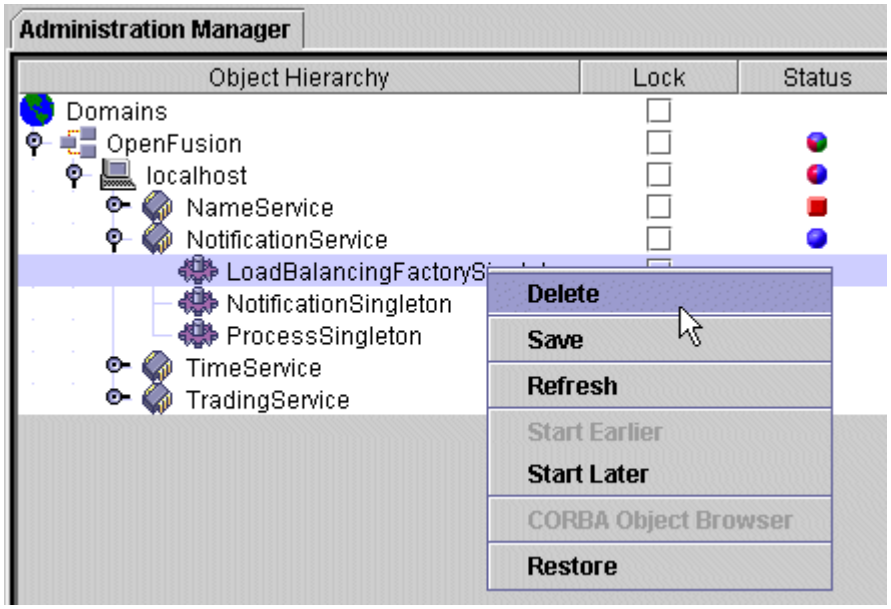

Figure 6 Deleting a Singleton

When a node is deleted, all children and all properties and settings of that node are also deleted.

A deleted node cannot be recovered, but a new node with the same name as the deleted node can be added later.

Changing the Ordering of Services and Singletons

The order of Services beneath nodes and Singletons/Java Objects beneath Services can be altered. The order is important as it determines the sequence in which Singletons and Java Objects will start when a node is started.

To move a Service, Singleton, or Java Object higher up the list, right-click on the node and select **Start Earlier** from the pop-up menu.

To move a Service, Singleton, or Java Object down the list, right-click on the node and select **Start Later** from the pop-up menu.

The *ProcessSingleton* is always the last Singleton in a Service and cannot be moved up.

Locking

Locking a property prevents that property from being updated. Single properties can be locked selectively, or an entire node in the *Object Hierarchy* can be locked.

This is not intended as a security measure. It is a simple matter to unlock a locked node or property. The purpose of the lock is to prevent accidental changes, and to prevent global changes from cascading through to a locked property.

Locking a property in the Administration Manager browser does not lock the property in the underlying CORBA object and does not prevent the property being changed programmatically.

The locked state of nodes and properties is saved when the browser is closed, so locks are restored when the browser is reloaded.

Locking Nodes

To lock a node in the *Object Hierarchy,* check the box in the *Lock* column for that node. To unlock the node, clear the checkbox.

Locks cascade to nodes lower in the hierarchy. If a Service is locked, the Singletons under that Service are also locked. If a domain is locked, the entire hierarchy under that domain is locked.

Nodes which have been locked by a cascade from a node higher in the hierarchy display a padlock icon in the *Lock* column. These nodes cannot be unlocked individually; the parent node much be unlocked first.

When a node is locked, all properties for the node are locked and cannot be individually unlocked. You can, however, selectively lock properties without locking an entire node. If a property is selectively locked and then a lock is applied to a higher node, the individual lock is retained if the higher lock is removed.

Starting a Service node can also cause some of the properties for that Service to be locked. Whether a property is locked or not when the Service starts is determined by the *Type* of the property (see *Type* on page 25).

Locking Properties

Properties can be locked for a number of reasons. Locked properties display a padlock icon in the *Lock* column and are coloured grey.

When a node is locked, all properties for that node automatically become locked. The only way to unlock these properties is to unlock the node.

Some properties are locked based on *Type* (see *Type* on page 25):

- *Fixed* properties are locked as soon as the Service is first activated (and cannot be unlocked, even if the Service is subsequently stopped).
- *Static* properties are locked while the Service is running and unlocked when the Service is stopped.

Some properties are locked based on the value of another property. For example, if the Security Enabled property for a service is checked, the other properties on the Security tab are unlocked and available. If the Security Enabled property is unchecked, then the other security properties are not needed and are all locked.

Any property can be locked individually, at the user's discretion. To lock a single property, check the box in the *Lock* column for that property. To unlock the property, clear the checkbox.

Restoring Services and Singletons

It is possible to restore all, or selected, Services and Singletons to their default (as supplied) states.

When a Singleton is restored, the IOR for the Singleton is deleted from the `domains` directory structure. (See *XML Configuration Files* on page 153 for details of how the `domains` directory structure maps the *Object Hierarchy*.)

When a Service is restored, the Service's data directory contents and log file are deleted and the IORs for each of that Service's Singletons are deleted.

The **Restore** command can be used at higher levels of the Object Hierarchy to restore all Services below the selected node.

Use this command with caution.

To restore a Service or a Singleton, right-click it in the *Object Hierarchy*.

1. Click **Restore** from the pop-up menu.
2. Set or clear the *Restore default properties* check-box:
   a) If is **not** checked, then the data directory contents and log file for the service node are deleted and the IOR files for each singleton is deleted for each service.
   b) If is **checked**, then all property values are returned to their default values.

Properties

The right-hand panel of the Administration Manager (*Figure 7*) shows the properties for the node selected in the Object Hierarchy.

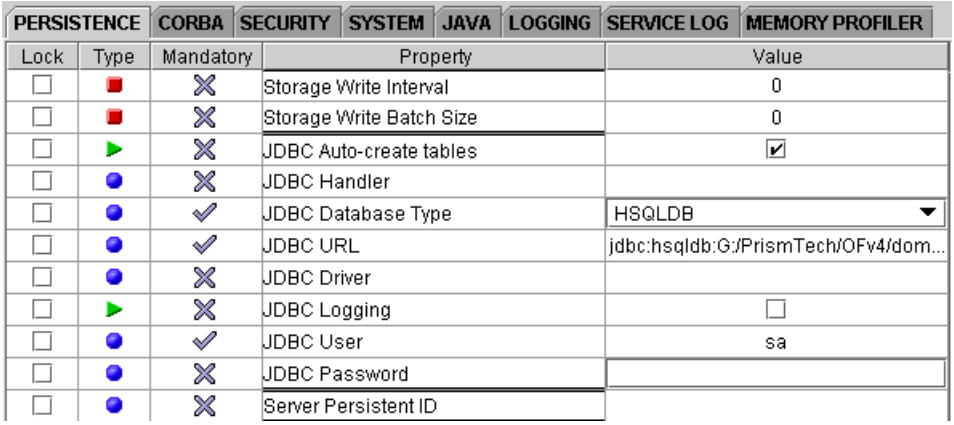

Figure 7 Administration Browser Properties Pane

Each service has properties arranged on tabbed panels. Utilities for service management are in the *Service Log* and *Memory Profiler* panels.

The following sections give basic instructions for working with properties. Details of how specific properties can be used for configuring individual Services are described in the sections dealing with each Service.

Type

Every property has a type, which defines how and when the property value can be changed. These types are identified by icons in the *Type* column, shown in *Table 3*.

Table 3 Property Types

Icon	Property Type
	*Fixed* The property can only be changed before the Service is started for the first time.
	*Dynamic* The property can be changed at any time, including while the Service is running.
	*Static* The property can only be changed when the Service is stopped.

Setting Properties Dynamically

If the value of a dynamic property is changed in the Administration Manager while the Service is running, the *Set* menu option ***must*** be used to update the property in the underlying CORBA object (see *Set* on page 29).

Mandatory

Some properties are defined as mandatory. A mandatory property is one which must be given a value before the Service is started and cannot be left blank. Zero (`0`) is a valid entry for a mandatory integer property.

The icons used in the *Mandatory* column to indicate mandatory properties are shown in *Table 4*.

Table 4 Mandatory Properties

Icon	Mandatory Status
	*Optional* The Service will start successfully with this property left blank.
	*Mandatory property* (blue tick) The Service will not start if this property is blank.
	*Incomplete mandatory property* (red tick) This icon is used for a mandatory property which has been left blank. A Service will not start if any of its properties show red ticks.

A node can be saved with mandatory properties left incomplete but a warning message will be displayed.

Accessibility

Properties can be read only or read/write. Read-only properties display information which can never be amended in the Administration Manager. Read/write properties can be amended (unless locked).

Read-only properties are indicated by grey shading. This is the same look as a locked property, but there is no icon in the *Lock* column for a read-only property.

Conditional Properties

Some properties will not appear on the Administration Manager property panel because they are *conditional* properties. These are properties which apply only to specific system configurations. For example, some properties relate to a specific ORB and will not appear on the screen if a different ORB is in use.

Assigning Values to Properties

A property with a boolean data type has a checkbox in the *Value* field. If the checkbox is ticked, the property is set to *true*. If the box is cleared, the property is set to *false*. To change the state of the checkbox, click it once.

A property with an enumerated data type has a drop-down list of valid values. To set the property, click the arrow at the right of the *Value* field and select a value from the list.

All other property values accept keyboard input. To set or edit the property, click in the *Value* field and type the required value.

Property Validation

Some property types are validated and will produce an error message if an invalid value is entered:

- *INTEGER* properties will only accept numeric input.
- *UUID* properties will only accept a string which is a valid UUID.
- *URL* properties will only accept a string which is a valid URL format. Only file, gopher, and http URL formats are accepted.
- *COUNTER* properties will only accept numeric input.

Entering Directory Paths

If the special characters $$are entered into a property field, the directory path of the current node is substituted. For example, if $$ is entered for a property of the NotificationSingleton, the following string is substituted:

```
<install_dir>/domains/OpenFusion/localhost/NotificationSer
vice/NotificationSingleton
```

Where *<install_dir>* is the OpenFusion installation directory and the directory path is entered as a continuous string (no carriage returns).

For example, enter the following to specify the location of the `NotificationSingleton.ior` file:

```
$$NotificationSingleton.ior
```

Note that the $$substitution includes the trailing slash of the directory path, so entering the following text would be incorrect (resulting in a double-slash):

```
$$/NotificationSingleton.ior
```

Actions that Can be Performed on Properties

Each type of property has a set of actions which can be performed on it. Right-click the property row to access a pop-up menu of actions.

The following actions are available.

Reset Counter

This action resets the counter to zero.

The action is only available for counter properties.

Refresh

This action retrieves the current value of the property from the underlying CORBA object and updates the *Value* field of the property.

If the value of an object's property is changed programmatically while the Service is running, the property displayed in the Administration Manager will not be updated unless this action is performed, and therefore can show a false value for dynamic properties.

This action can only be performed while the Service is running (and therefore is only available for dynamic properties).

Set

This action transfers the value of the property in the Administration Manager to the underlying CORBA object.

This action can only be performed while the Service is running (and therefore is only available for dynamic properties).

If the value of a dynamic property is changed in the Administration Manager while the service is running, the property in the underlying CORBA object is ***not*** automatically updated. The **Set** action ***must*** be used to update the CORBA object property.

Assign Value to Peers

This action copies the value of the selected property to all peers (all objects under the same parent node) which have a property with the same name.

If this action is performed on a Singleton property, the same property for all other Singletons and Java Objects under the same Service will be updated. If the action is performed on a Service property, the same property for all other Services under the same node will be updated.

If it is a dynamic property, the updated value is also set in the underlying CORBA object.

For example, the *Storage Type* property could be changed to *File* for the NotificationService and this command could be used to transfer that change to all other Services under the same node.

Properties which are locked (see *Locking* on page 22) are protected from being updated by this action.

Assign Value Globally

This action copies the value of the selected property to all properties in the *Object Hierarchy* which have the same name. This action is similar to **Assign Value to Peers** but the change is made over the entire Object Hierarchy.

If it is a dynamic property, the updated value is also set in the underlying CORBA object.

Properties which are locked (see *Locking* on page 22) are protected from being updated by this action.

New UUID

Assigns a valid UUID to the property.

This action is only available for UUID properties.

Signals

Signals are displayed as buttons in a Service's property list, as shown in *Figure 8*.

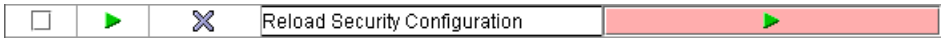

Figure 8 A Signal Button

When clicked, a signal button will trigger some action in the underlying Service. The action each signal button performs will depend on how the signal has been defined and will be described in the documentation for each Service.

A signal will only trigger an action when the underlying Service is running.

User Identity

To access secured services, a valid user identity must be provided.

The current user identity is displayed in the Administration Manager's status bar. To change the identity, use the *Enter user identity* tool bar button and enter a user name and password in the *User Identity Details* dialog box.

Service Log

Every Service has a log file that can be viewed on the *SERVICE LOG* tab for the Service. Only the last (most recent) 250Kb of the log file will be displayed.

The log file for each Service can be configured to specify log file location, maximum log file size, the level of information to be logged, and other factors. See *Logging Properties* on page 55 for details.

Use the **Refresh Log** button to refresh the display with the current contents of the log file (the display is not automatically updated when the file contents change).

Use the **Delete Log File** button to clear the Service Log. This clears the display and deletes the contents of the underlying log file. The Service Log can only be deleted if the Service is not running.

Memory Profiler

The Memory Profiler for each service shows the total available, used, and free memory in the Java Virtual Machine (JVM) that the service is running in. The total and used memory are also shown as a graph which shows changes over time. The graph is illustrated in *Figure 9*.

To start the Memory Profiler, select the reporting interval from the *Interval* drop-down list and click **Start**. The service must be running or the Memory Profiler will not start. To halt the Profiler, click **Stop**. Stopping the Profiler freezes the display but does not clear it. Re-starting a stopped graph, however, clears the display.

The scale of the *Memory* axis (y-axis) changes dynamically in order to effectively display changing amounts of memory.

The **Clean** button forces immediate garbage collection on the current process. The results of this will be seen as a drop in the Used JVM Memory and an increase in the Free JVM Memory on the Memory Profiler. This operation can be performed even when the Memory Profiler is stopped.

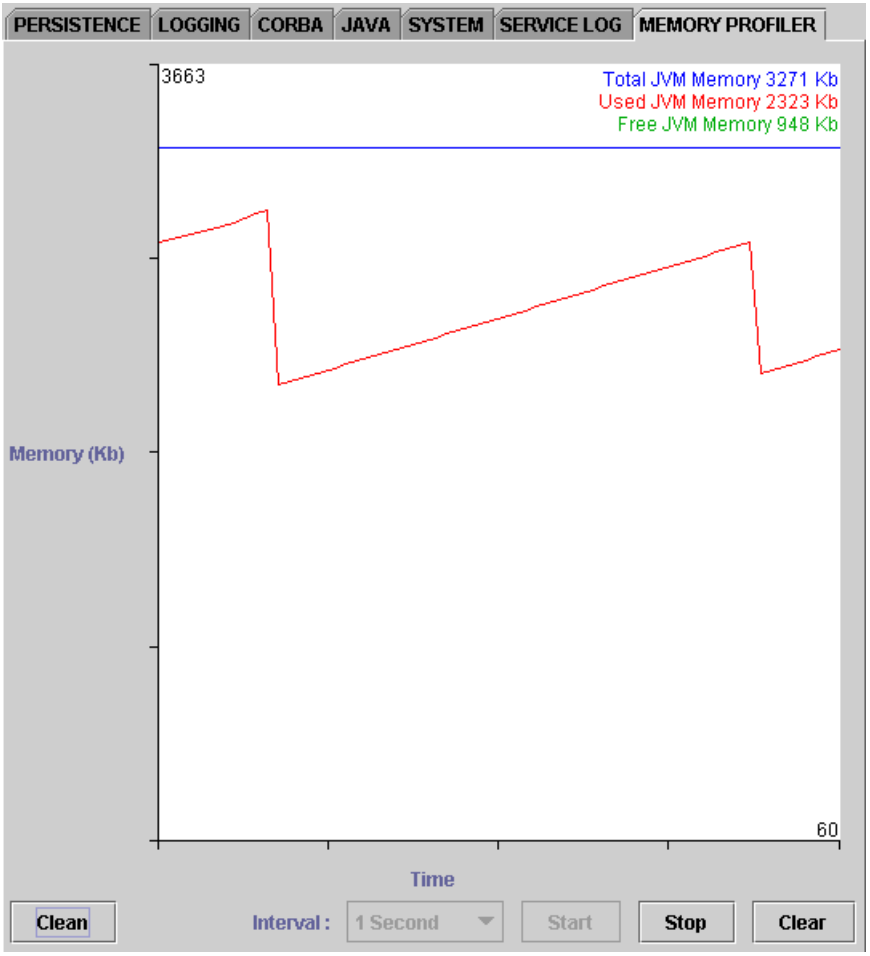

Figure 9 Memory Profiler

Tool Bar Options

The browser tool bar buttons provide access to a number of common features. These buttons are described in *Table 5*.

Many of these functions can be performed by using a key combination (control key plus a letter). These keyboard short cuts are also shown in *Table 5*.

Table 5 Tool Bar Buttons

Button	Function
	**Delete selected browser (Ctrl+D)** Removes the currently active browser from the browser framework. The browser configuration is not automatically saved when the browser is removed.
	**Refresh selected browser (Ctrl+R)** Refreshes the currently active browser. This does not refresh property values unless they are dynamic and the service is running.
	**Refresh the current node** Refreshes the view of the node currently selected in the *Object Hierarchy*. This does not refresh property values unless they are dynamic and the service is running. This button is only valid for the Administration Manager.
	**Stop current action** Aborts any action which has been initiated but has not yet completed.
	**Launch the file browser** Opens the file browser.
	**Save configuration** Save the current values of all the properties in the Object Hierarchy. This button is only valid for the Administration Manager.
	**View the browser log** Opens the browser's message log file.
	Enter user identity Opens the User Identity Details dialog box to allow user authentication.

If a function is not available in a particular browser, the corresponding button will be greyed-out while that browser is active.

When functions specific to a particular browser are added to the tool bar, the buttons will be described in the section of this Guide which deals with the relevant browser.

2.4 The CORBA Object Browser

Any Singleton of a running Service can be queried from the Administration Manager to reveal key information about the Singleton.

To query the Singleton, right-click the Singleton and select **CORBA Object Browser** from the pop-up menu, as shown in *Figure 10*.

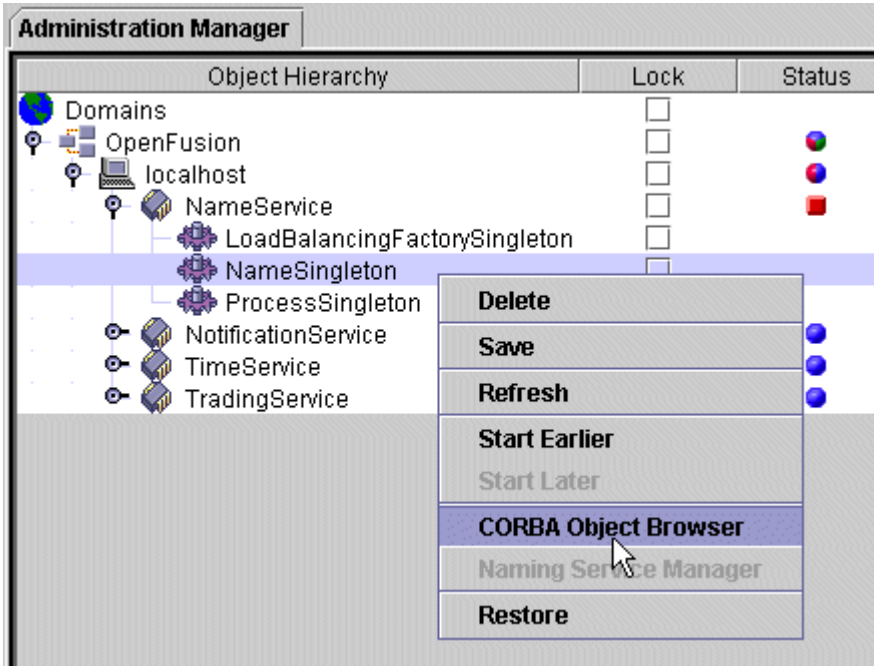

Figure 10 Starting the CORBA Object Browser

This action opens the CORBA Object Browser, as shown in *Figure 11*. The CORBA Object Browser can also be started from the command on the *Tools* menu.

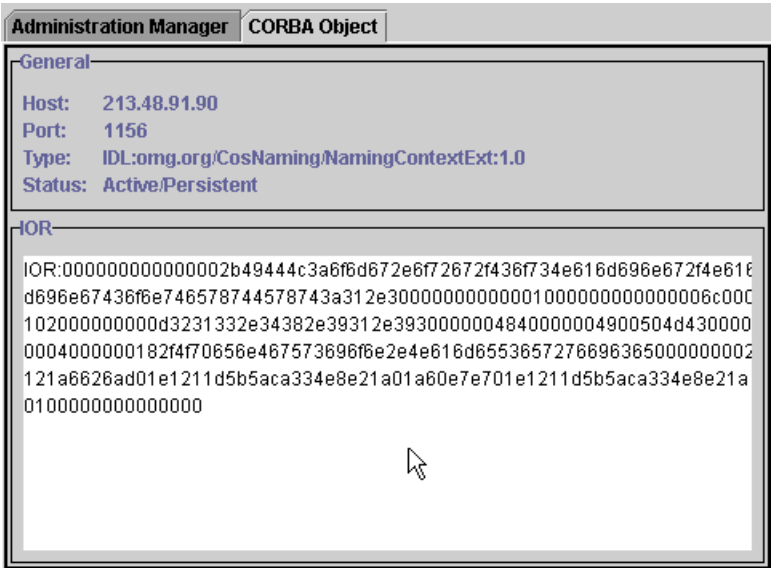

Figure 11 CORBA Object Browser

The CORBA Object Browser displays the following information about the Singleton object:

- Host IP address.
- Host port number.
- Object type.
- Status (active or inactive and persistent or non-persistent).
- IOR.

The displayed IOR can be selected and copied to the clipboard as a string.

When the CORBA Object Browser is active, two buttons are added to the tool bar. These buttons are shown in *Table 6*.

Table 6 CORBA Object Browser Tool Bar Buttons

Button	Function
	**Load IOR** Loads a previously-saved IOR from a text file. The file must contain a valid IOR as a string.
	**Save IOR** Writes the object's IOR as a string to a text file.

2.5 Distributed Installation Configuration

Multiple OpenFusion installations can be configured from a central host. This allows OpenFusion Services on different machines to share common configuration files and, if required, a common implementation repository.

The machines can be connected via a shared file system or by using a Web server running on the central host.

Note that the central configuration host and each remote machine must have a licensed OpenFusion installation.

The Central Configuration Host

The XML files used to configure the properties of each remote OpenFusion installation are all stored on the central configuration host under the `domains` directory (see *The Object Hierarchy* on page 153 for details). The central host must therefore be configured to store details of each remote installation.

Each remote installation should be set up as a separate node in the Administration Manager Object Hierarchy on the central host. (See *Adding Nodes* on page 19.)

Figure 12 shows the Object Hierarchy of a central configuration host managing four remote machines.

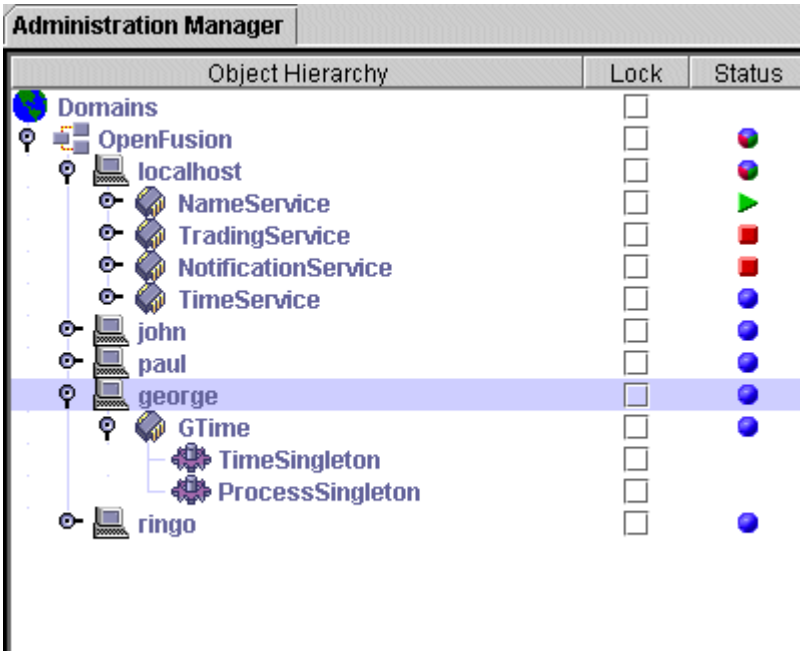

Figure 12 Remote Nodes

Each node should have the unique machine name of the remotely-managed computer.

Each remote machine displays the full Object Hierarchy from the central host, including all remote machine nodes and the *localhost* node, unless the Administration Manager is started with the `-remote` command-line option, for example:

```
% bin/manager -remote
```

The appearance of the *localhost* node could potentially cause confusion for remote users. It might be assumed that *localhost* refers to the remote machine, but it actually refers to the central host. To avoid the confusion, delete the *localhost* node from the central configuration host's Object Hierarchy and add a new node with the name of the host machine.

Add Services to each node and add Singletons and Java Objects to the Services, as described in *Adding Singletons and Java Objects* on page 20. *Figure 12* shows one Service with two Singletons added to a remote node. The Singletons and Java Objects must exist as valid, licensed objects on the remote machine.

Using a Shared File System

All hosts must have identical mappings to a common file system.

On Windows systems, network drives must be mapped so that all machines (including the central host) can refer to the central OpenFusion installation directory with the same drive letter. This ensures that a directory path (for example, `O:\OpenFusion\domains`) will always point to the same location on the central host regardless of which remote machine it is invoked from.

On Unix, use a soft link to achieve the same effect.

Note that it is not possible to use a common file system to link OpenFusion installations running on a mixture of Unix and Windows hosts. In a mixed operating system environment, central configuration can only be performed via a Web server (see *Using a Web Server* on page 39).

Set up the Central Host

On the central host, set up nodes in the Object Hierarchy with each node representing a remote machine. This is described in *The Central Configuration Host* on page 36.

Implementation Repository

It may be necessary to configure remote installations to share a common implementation repository. This is not necessary if the central host is only used for configuration purposes, but it is required if the clients need to communicate with Services running on other hosts (an example is when *load balancing* is being used, as described in the *Load Balancing Service Guide*).

The Servers running on each host must be configured to use the common implementation repository. With JacORB, for example, the `ORBInitRef.ImplementationRepository` property in the `jacorb.properties` file on each host must be set to point to the location of the common implementation repository's IMR file.

The common implementation repository can be running on any host.

In order to prevent conflicts when starting the Administration Manager with a common implementation repository, the *POA Name* property (found on the *CORBA* tab of the *Domains* node in the Administration Manager) must be unique for each host. To change this property, the Administration Manager must be started for each host with the `-noorb` option.

Environment Properties

Each remote host requires an `OF_DOMAINS_URL` environment property set to the `domains` directory on the remote host. If the shared file system has been mapped correctly, this property should be identical on every host. For example:

WIN
```
set OF_DOMAINS_URL=file://O:\Openfusion\domains
```

UNIX
```
export OF_DOMAINS_URL=file:///usr/users/central/OpenFusion/domains
```

Each remote host requires an `OF_Admin_URL` environment property set to the local `domains` directory. For example:

WIN
```
set OF_Admin_URL=file://C:\Openfusion\domains
```

UNIX
```
export OF_Admin_URL=file:///usr/users/node1/OpenFusion/domains
```

Using a Web Server

The central configuration host must be running Web server software. (Any third-party Web server will be suitable.)

The OpenFusion distribution includes the Tomcat Web server, but this ***should not*** be used to enable remote configuration.

Set up the Central Host

On the central host, set up nodes in the Object Hierarchy with each node representing a remote machine. This is described in *The Central Configuration Host* on page 36.

Configure Remote Singletons

When you add a Singleton to a remote node in the Object Hierarchy, it will have default data locations that apply to the central host. These locations must be changed to point to valid locations on the remote machine. As many of these properties are hidden from the Administration Manager GUI, the underlying XML files must be edited directly (***Note:*** this is normally not recommended, and care should be taken that no errors are introduced into the XML files).

Appendix A, *XML Configuration Files* on page 153, gives details of the structure and locations of the files which must be edited.

Every Singleton property value which is a directory path should be changed to point to a location on the remote machine. If the central host and the remote machine have exactly the same installation path and directory structure for their OpenFusion installations, these properties will be correct and do not need to be changed.

If hsqldb is used for persistence (see *hsqldb* on page 112), ensure that the `DB.WAL.DIR` property for each Service is set to point to an existing directory on the remote machine, otherwise the Service will not start.

Set the Central Host Properties

Set the properties of the `domains` node in the Object Hierarchy as described here. (These properties are described fully in *Configure Properties* on page 73.)

Central Configuration Host

This check box should be checked to indicate that the machine is the central configuration host.

OpenFusion Install URL

The URL that remote machines must use to access the central configuration host. This is a *http* URL which gives the host's machine name. This URL will be determined by the root document directory of the Web server.

For example, if the central configuration host is an NT Server named `central` with the Web server document directory set to `C:\`, and the OpenFusion installation on that machine is `C:\PrismTech\OpenFusion`, then the correct URL will be:

```
http://central/PrismTech/OpenFusion
```

If the Web server document directory is set to `C:\PrismTech\OpenFusion`, however, the correct URL will be:

```
http://central
```

Caution: entering an invalid URL will cause fatal problems! Take backups of the OpenFusion installation and be very careful when changing this property.

Configure from Remote Host

This check box should remain clear on the central host. The setting is only needed on remote machines.

Remote OpenFusion Install URL

This setting is not needed on the central host. The property should be locked, as *Configure from Remote Host* should not have been selected on the central host.

Set up the Remote Machine

To set up a remote machine to use central configuration, the central host must have been configured and the remote machine must have been set up as a node in the central host's Object Hierarchy.

The following properties must be configured for the `domains` node in the remote machine's Object Hierarchy. These properties are described fully in *Configure Properties* on page 73.

These properties are stored on the remote machine, not the central configuration host, which is why they must be set on each remote machine.

Central Configuration Host

This check box should remain clear on the remote machine. The setting is only needed on the central host.

OpenFusion Install URL

This setting is not needed on the remote machine. The property should be locked, as *Central Configuration Host* should not have been selected on the remote machine.

Configure from Remote Host

If this machine is to be configured from a central host, this check box must be checked.

Remote OpenFusion Domains URL

A URL which points to the location on the central host that stores the XML configuration files. This will be the OpenFusion installation directory.

This URL will be determined by the root document directory of the Web server.

For example, if the central configuration host is an NT Server named `central` with the Web server document directory set to `C:\`, and the OpenFusion installation on that machine is `C:\PrismTech\OpenFusion`, then the correct URL will be:

```
http://central/PrismTech/OpenFusion/domains
```

If the Web server document directory is set to `C:\PrismTech\OpenFusion`, however, the correct URL will be:

```
http://central/domains
```

Working with Central Configuration

When a remote machine is configured from a central host, all of the XML files which hold properties for Services and Singletons are stored on the central host. The remote machine can *read* from the configuration files but cannot *write* to them.

Because the remote machine cannot write to its own configuration files, it can never over-ride the configuration set by the remote configuration host. On the remote machine, most properties will be *locked*.

The only properties which remain unlocked are *Dynamic* properties (see *Type* on page 25). Changes to these properties will not be stored permanently when the Administration Manager is shut down.

Many main menu options, tool bar buttons, and right-click menu options are disabled on the remote machine. All actions which apply to changing the Object Hierarchy or modifying locked property values are disabled.

2.6 Tomcat Web Server Integration

It is possible to deploy the Java Tomcat Web server within an OpenFusion installation. The Tomcat server is used by the OpenFusion SOAP Bridge, as described in the *Messaging Bridges Guide*.

Tomcat is deployed as an embedded server and can be configured as a Java Object. As such, it can be deployed as a separate service or co-located with another OpenFusion service. See *Adding Singletons and Java Objects* on page 20 for details of deploying Java Objects in an OpenFusion installation.

See *http://jakarta.apache.org/* for further details of the Tomcat Web server.

Deployment of Web Archives

Each configured Tomcat object has its own `webapps` directory created within the configuration directory hierarchy. In addition, a global `webapps` directory is maintained at the root level of the OpenFusion installation.

By default, the ROOT Web archive file (`ROOT.war`) is deployed into every configured Tomcat instance. Additional Web archive files can be deployed by one of the following methods:

- Put the file into the specific `webapps` directory.
- Put the file into the global `webapps` directory and configure the *Tomcat WAR Files* property to include the file name.

Security

The Tomcat Web server has a security manager enabled by default. This uses the following security policy file:

```
<INSTALL>/etc/tomcat.policy
```

where `<INSTALL>` is the OpenFusion installation directory.

If more fine-grain security control is required, the file can be copied into the Tomcat home directory and edited as appropriate. The *Tomcat Security Policy File* property is used to locate this file.

Deploying Java Server Pages

In order to deploy your own Java Server Pages (JSPs) in the Tomcat Web server, a `.jar` file containing a Java compiler must be included in the CLASSPATH.

JSPs supplied with the OpenFusion CORBA Services distribution are pre-compiled and deployed within a `.war` file and therefore do not need access to a Java compiler.

Configuration

The following properties can be configured through the GUI manager for each embedded Tomcat object.

Tomcat Home Directory

The home directory of the Tomcat server. This defaults to the configuration directory for the Tomcat Java Object:

```
<INSTALL>/domains/<domain>/<node>/<service>/TomcatObject/
```

where `<INSTALL>` is the OpenFusion CORBA Services installation directory. See *The Object Hierarchy* on page 153 for details of the `domains` directory structure.

Property Name	`Tomcat.Home`
Property Type	`STATIC`
Data Type	`DIRECTORY`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Tomcat Work Directory

The Tomcat work directory. This defaults to the `work` directory under the default Tomcat home directory:

<INSTALL>/domains/<domain>/<node>/<service>/TomcatObject/work

where <INSTALL> is the OpenFusion CORBA Services installation directory. See *The Object Hierarchy* on page 153 for details of the domains directory structure.

This property is independent of the *Tomcat Home Directory* property and does not change if *Tomcat Home Directory* is changed.

Property Name	Tomcat.WorkDir
Property Type	STATIC
Data Type	DIRECTORY
Accessibility	READ/WRITE
Mandatory	YES

Tomcat WAR directory

The directory into which Tomcat deploys Web archive files. This defaults to the webapps directory under the default Tomcat home directory:

<INSTALL>/domains/<domain>/<node>/<service>/TomcatObject/webapps

where <INSTALL> is the OpenFusion CORBA Services installation directory. See *The Object Hierarchy* on page 153 for details of the domains directory structure.

This property is independent of the *Tomcat Home Directory* property and does not change if *Tomcat Home Directory* is changed.

Property Name	Tomcat.Context
Property Type	STATIC
Data Type	DIRECTORY
Accessibility	READ/WRITE
Mandatory	YES

Tomcat WAR Files

A colon-separated list of Web archive files to be deployed from the global webapps directory into the Tomcat WAR directory.

The ROOT.war file is always deployed and does not have to be included in the list.

Property Name	Tomcat.Archives
Property Type	STATIC

Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

Tomcat Security Policy File

The full path and name of the file which defines the security policies used by the Tomcat security manager. This defaults to:

```
<INSTALL>/etc/tomcat.policy
```

where <INSTALL> is the OpenFusion CORBA Services installation directory.

Property Name	Tomcat.PolicyFile
Property Type	STATIC
Data Type	FILE
Accessibility	READ/WRITE
Mandatory	YES

Tomcat Port

The port on which the Tomcat server listens for http requests. The default is 8080, but if this port in use by any other Web server deployed on the same system, a different port must be selected.

Property Name	Tomcat.Port
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES

Serve Root URL

This property should be checked (TRUE) if the Tomcat Object will be used to serve up the files used for remote system configuration. The default value for this property is TRUE. See Section 2.5, *Distributed Installation Configuration*, on page 36, for more details of remote system configuration.

Property Name	Tomcat.ServeRoot
Property Type	STATIC

Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES

Root URL

The directory that will be served up by the Tomcat Object. To allow remote system configuration, this should be the OpenFusion installation directory (which is the default value for the property). The directory should be specified as a URL of type *file://*.

This directory will be served up when a Web browser is used to access the following URL:

```
http://<server>:<port>/<context>
```

Where:

`server` is the machine which is running the Tomcat Object.
`port` is the port that Tomcat listens on, specified in the *Tomcat Port* property.
`context` is the path specified in the *Context Path* property.

If the directory contains a file called `index.html`, that file is returned to the browser. If `index.html` does not exist, a directory listing is returned instead.

This property is only enabled if the *Serve Root URL* property is checked.

Property Name	Tomcat.RootURL
Property Type	STATIC
Data Type	URL
Accessibility	READ/WRITE
Mandatory	YES

Context Path

The virtual directory that will be served up by the Tomcat Object. This defaults to the name of the Service that the Tomcat Object is a part of. See the *Root URL* property for details of how this property can be used.

This property is only enabled if the *Serve Root URL* property is checked.

Property Name	Tomcat.ContextPath
Property Type	STATIC

Data Type	STRING
Accessibility	READ/WRITE
Mandatory	YES

Testing the Tomcat Installation

Once the service containing the Tomcat Java Object has been started, the Tomcat deployment can be tested by attempting to connect to the default Web page. To connect to the default page, type the following into the address bar of a Web browser:

```
http://<server>:<port>
```

where <server> is the name of the machine running the Tomcat server and <port> is the port number specified in the *Tomcat Port* property (8080 by default).

If the server is working correctly, the Web page illustrated in *Figure 13* will be displayed.

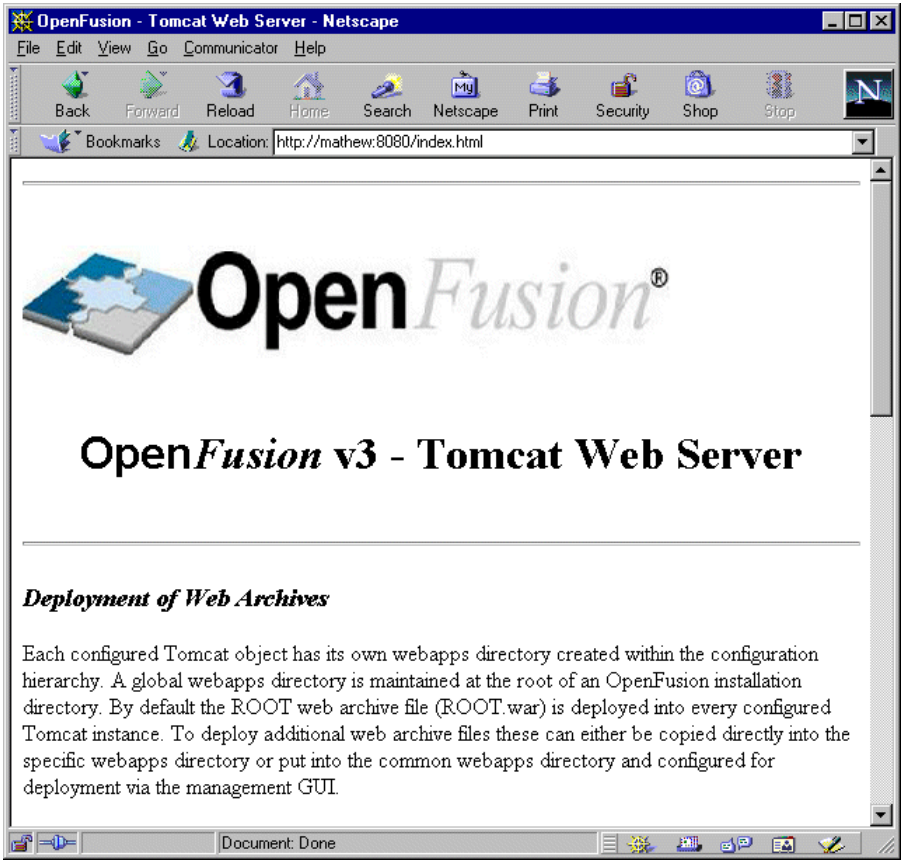

Figure 13 Tomcat Default Web Page

3 Common Configuration Properties

3.1 Overview

The properties described in this section are found in all Services. The properties must be set individually for each Service, although the commands *Assign Value to Peers* on page 29 and *Assign Value Globally* on page 30 can be used to set the same values for multiple Services.

The properties are grouped by function on different tabs of the properties pane.

i The `ENUM` data type has special meaning in that it represents a drop-down list in the GUI where a definitive list of values is allowed, rather than the usual meanings associated with the other data types (e.g. the INTEGER data type represents an integer).

3.2 Persistence Properties

The properties on the *Persistence* tab determine how and where the Service data is stored persistently. See Section 6, *Configuring Persistent Storage*, on page 107 for details of different persistent storage methods.

Storage Write Interval

This property specifies the delay (in seconds) between saving object state changes within a server and writing this information to persistent storage. This option is a performance optimization feature as it can be used to prevent the Service from 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). Increasing the write interval may improve performance when the data held by a service is changing rapidly.

Property Name	`DB.WriteInterval`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Storage Write Batch Size

The *Storage Write Batch Size* option specifies the maximum number of updates that will be buffered before the data is written to persistent storage. This option is a performance optimization feature.

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

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

Property Name	`DB.WriteBatch`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

JDBC Auto-create tables

If this property is checked (`true`), the Service will check for the presence of the JDBC tables required for persistent storage and automatically create the tables if they are not present.

The default value for this property is `true`.

Property Name	`DB.JDBC.AutoCreate`
Property Type	`FIXED`
Data Type	`BOOLEAN`
Accessibility	`READ-WRITE`
Mandatory	`NO`

JDBC Handler

The class name of the custom plug-in which will implement the JDBC `ExceptionHandler` interface.

The `ExceptionHandler` interface allows the customising of how an SQL exception will be handled. The interface is specified as follows:

```
public interface ExceptionHandler
{
   public static final int OK = 0;
   public static final int REPEAT = -1;
   public static final int FATAL = -2;
```

```
   public int handleException (java.sql.SQLException ex);
}
```

This operation should return a status indicating how an SQL exception should be handled. Possible return values are:

`OK`	The program should continue as normal.
`REPEAT`	The database operation should be re-tried immediately.
`FATAL`	The program should terminate.

A return value greater than `0` (zero) means that the database operation should be re-tried after the returned interval (in milliseconds).

Property Name	`DB.JDBC.Handler`
Property Type	`FIXED`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`NO`

JDBC Database Type

The *JDBC Database Type* option specifies the available, underlying relational database type. Select from one of the following supported databases:

- Oracle
- Sybase
- Informix
- SQL Server (Windows only)
- hsqldb

The default persistence option is hsqldb, which is installed with the OpenFusion CORBA Services distribution and will run with no additional configuration.

Property Name	`DB.JDBC.Type`
Property Type	`FIXED`
Data Type	`ENUM`
Accessibility	`READ/WRITE`
Mandatory	`YES`

JDBC URL

The *JDBC URL* option sets the location of the JDBC data source. The format of the URL depends on the type of data source being used.

Oracle

```
jdbc:oracle:thin:<data_source_name>
```

Where `<data_source_name>` is the name of the JDBC data source.

Sybase

```
jdbc:sybase:Tds:<data_source_name>
```

Where `<data_source_name>` is the name of the JDBC data source.

Informix

```
jdbc:informix-sqli:<data_source_name>
```

Where `<data_source_name>` is the name of the JDBC data source.

SQL Server

```
jdbc:odbc:<data_source_name>
```

Where `<data_source_name>` is the name of the JDBC data source.

hsqldb

There are three ways in which hsqldb can be run, each requiring a different URL format.

- Running on the local machine:

  ```
  jbbc:hsqldb:<database>
  ```

  Where `<database>` is the path to the hsqldb database. The default database location is a subdirectory of the Service directory, as follows:

  ```
  jdbc:hsqldb:<install_path>/domains/<domain>/<node>/
  <service>/data/hsqldb
  ```

  Where:
  `<install_path>` is the OpenFusion installation directory.
  `<domain>` is the name of the domain.
  `<node>` is the name of the node.
  `<service>` is the name of the OpenFusion Service.

  This default location can be changed if required.

- Running in local memory (in the same JVM as the Service being started):

```
jdbc:hsqldb:.
```

- Running on a remote machine:

  ```
  jdbc:hsqldb:hsql://<host>:<port>
  ```

  Where:
  `<host>` is the name of the remote machine.
  `<port>` is the port used to connect to hsqldb on the host machine. This is optional but will be required if the host machine runs more than one hsqldb server.

Property Name	`DB.JDBC.URL`
Property Type	`FIXED`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`

JDBC Driver

This is the class name of the JDBC driver used. A default driver based upon the type of database chosen will be used when this field is left blank, so it is not normally necessary to set this field.

Property Name	`DB.JBDC.Driver`
Property Type	`FIXED`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`NO`

JDBC Logging

Whether JDBC calls will be logged or not.

Property Name	`DB.JDBC.Logging`
Property Type	`DYNAMIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

JDBC User

Your Database Administrator will provide the user name for use in the *JDBC User* option.The default user is `sa` (the hsqldb system administrator user).

This user must have *create* rights on the database.

Property Name	`DB.JDBC.User`
Property Type	`FIXED`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`

JDBC Password

Your Database Administrator will provide the password for use in the *JDBC Password* option. The default password is blank (none is required for the default user, `sa`, in hsqldb).

Property Name	`DB.JDBC.Password`
Property Type	`FIXED`
Data Type	`PASSWORD`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Server Persistent ID

A unique identifier (*UUID*) associated with a specific server. Persistent storage databases use this ID to indicate which server persistent data belongs to. This allows different processes to share persistent data.

Property Name	`SID`
Property Type	`FIXED`
Data Type	`UUID`
Accessibility	`READ/WRITE`
Mandatory	`NO`

3.3 Logging Properties

All OpenFusion Services can produce logging information. This can be used to both track bugs and monitor server operation. OpenFusion uses four basic logging levels: *Error*, *Warning*, *Information* and *Debug*. The OpenFusion logging system uses the `log4j` logging package. (See *http://jakarta.apache.org/log4j* for more information.)

Log Pattern

The format used for the logging output. This property is only required if *Log Layout* is set to *Pattern*.

Details of setting log patterns can be found in *Log Messages* on page 163.

Property Name	`log4j.appender.Default.layout.ConversionPattern`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Log Layout

The layout used for the logging output. Choices are:

- *Simple*
- *Pattern*

If *Pattern* is selected, the *Log Pattern* property must be set.

Property Name	`log4j.appender.Default.layout`
Property Type	`STATIC`
Data Type	`ENUM`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Enabled LogFactor5 pattern layout

If `true`, log output is formatted for viewing with LogFactor5. The default is `false`. The property is only used if the Log Layout property is set to *Pattern*.

Property Name	`LogFactor5.enabled`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Syslog Facility

This is the UNIX Syslog facility to which logging is directed. See your UNIX documentation for more information on Syslog facility categories.

Property Name	`log4j.appender.Default.Facility`
Property Type	`STATIC`
Data Type	`ENUM`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Syslog Host

This option determines the name of the host to which logging is directed when the Syslog logging plug-in is selected. *Syslog* output is sent to the local host by default.

Property Name	`log4j.appender.Default.SyslogHost`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`

File Backup Number

This is the number of backup files that are retained after the value of *File Maximum Size* is exceeded and *RollingFile* is selected as the logging plug-in. The default is `1`.

Property Name	`log4j.appender.Default.MaxBackupIndex`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

File Append

This option controls whether the existing log file is replaced or new messages are appended to the file.

Property Name	`log4j.appender.Default.Append`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

File Maximum Size

This is the maximum size, in megabytes, of the log file created when *RollingFile* is selected as the logging plug-in. A new logging file will be created when the value of *File Maximum Size* is exceeded.

Property Name	`log4j.appender.Default.MaxFileSize`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Log File

The *File Name* specifies the file where diagnostic output is saved. A default value is used when this property is not set. This is:

```
<install_dir>/domains/<domain>/<node>/<service>/log/<service>.log
```

where <*install_dir*> is the OpenFusion installation path. See *The Object Hierarchy* on page 153 for details of the `domains` directory structure.

Property Name	`log4j.appender.Default.File`
Property Type	`STATIC`
Data Type	`FILE`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Log Plug-in

This property determines how diagnostic output will be logged. Options are:

- *File*
- *Rolling File*
- *Syslog*
- *Event Log*
- *Log Service*
- *None*

File

Selecting this option will direct all diagnostic output to a file specified by the *Log File* property.

Rolling File

This option directs diagnostic output to a file. The output file is backed up periodically when a specific size is reached. See also *File Backup Number* on page 57.

Syslog

UNIX This option directs diagnostic output to the UNIX syslog facility.

WIN *Event Log*

This option directs diagnostic output to the NT Event Log.

Log Service

This option redirects all diagnostic output to the OpenFusion Log Service. A notification-type log is used.

None

All diagnostic messages are disabled.

Property Name	`log4j.appender.Default`
Property Type	`STATIC`
Data Type	`ENUM`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Notify Log ID

This is the identity of the Notify Log that is used when logging to the Log Service is selected. A new log is created when a log with this identity does not already exist.

Property Name	`log4j.appender.Default.LogID`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Log Level

This property determines the level of diagnostic output that is logged in the log file. The logging level can be changed dynamically. Options are:

- Disable - No messages are logged; logging is disabled.
- *Error* - Only error messages are logged.
- *Warning* - Error and warning messages are logged.
- *Information* - Error, warning, and information messages are logged.
- *Debug* - Error, warning, information, and debugging messages are logged.

Caution: significant amounts of output may be generated when the *Debug* level of logging is selected.

Property Name	`log4j.rootLogger`
Property Type	`DYNAMIC`

Data Type	ENUM
Accessibility	READ/WRITE
Mandatory	YES

3.4 CORBA Properties

The properties on the *CORBA* tab provide a view of the CORBA-related state of the services.

INITIALIZE Exception Count

The current total of CORBA INITIALIZE exceptions thrown since the Service was started.

Property Name	CORBA.InitializeExceptions
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ ONLY
Mandatory	NO

BAD_PARAM Exception Count

The current total of CORBA BAD_PARAM exceptions thrown since the Service was started.

Property Name	CORBA.BadParamExceptions
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ ONLY
Mandatory	NO

INTERNAL Exception Count

The current total of CORBA INTERNAL exceptions thrown since the Service was started.

Property Name	CORBA.InternalExceptions
Property Type	DYNAMIC

Data Type	COUNTER
Accessibility	READ ONLY
Mandatory	NO

Number of active CORBA objects

Number of active CORBA objects currently in service.

Property Name	ObjectRegistry.Objects
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ ONLY
Mandatory	NO

Number of purged CORBA objects

Number of CORBA objects purged from memory.

Property Name	ObjectRegistry.Purges
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ ONLY
Mandatory	NO

CORBA Object Activity Timeout

Timeout for CORBA object activity check, in seconds.

Property Name	Timeout
Property Type	DYNAMIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	NO

Incoming Call Count

The current total of CORBA operations invoked.

Property Name	CORBA.Calls
Property Type	DYNAMIC
Data Type	COUNTER
Accessibility	READ ONLY
Mandatory	NO

Load CORBA Singletons on Startup

Whether to load Singletons on server startup or on demand.

Property Name	LoadOnStart
Property Type	STATIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	NO

Enable Dynamic Portable Interceptors

If set to TRUE (checked), this property enables the use of OpenFusion Dynamic Portable Interceptors for the Service.

Dynamic Portable Interceptors are required by an OpenFusion Service using adaptive load balancing with load shedding. See the *Load Balancing Service Guide* for details.

Property Name	EnableDynamicInterceptors
Property Type	STATIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	NO

Object Purging

When set to *TRUE* this option enables the purging (deactivation) of objects from the server, limiting the amount of object references that are stored by the ORB. Objects may be purged at a given interval and/or when a maximum number of object references has been exceeded. Objects are purged using a least-recently-used algorithm.

The properties *Object Cache Maximum Size* and *Object Cache Minimum Size* are used to control object purging behaviour. These properties set upper and lower limits for the number of object references that the ObjectRegistry is expected to manage. Object purging will be triggered when the number of object references exceeds the *Object Cache Maximum Size* limit. The purging algorithm will attempt to destroy sufficient object references to reduce the number held in the ObjectRegistry to that specified by the *Object Cache Minimum Size* property.

For example, with the properties `ObjectRegistry.MaxSize=1000` and `ObjectRegistry.MinSize=100`, purging will be triggered when the 1001st object reference is created. The purging algorithm will attempt to destroy 901 object references to reduce the numbr of references held in the ObjectRegistry to 100.

Note that memory usage does not correlate directly to the number of objects.

Naming Service: When the OpenFusion Naming Service is being used with object purging enabled, clients ***must*** always perform operations from the root context. Otherwise, problems will occur if the parents have been purged from memory.

Property Name	`ObjectRegistry.Purge`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Object Cache Maximum Size

This is the maximum number of objects that can be created in a server before purging occurs. When the object references handled by the ObjectRegistry exceeds the value of this property, objects are removed using a least-recently-used algorithm.

Objects will only be purged if Object Purging has be set `TRUE`. For full details of using this property, see *Object Purging* on page 62.

Property Name	`ObjectRegistry.MaxSize`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Object Cache Minimum Size

The minimum cache size for persistent CORBA objects. When objects are purged from the server, this number of objects will be left. For full details of using this property, see *Object Purging* on page 62.

Property Name	`ObjectRegistry.MinSize`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Object Cache Purge Interval

This is the interval, in minutes, between object purge operations.

Objects will only be purged if *Object Purging* has be set *TRUE*.

Property Name	`ObjectRegistry.Interval`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

ORB Initialization Arguments

This is a space separated list of arguments passed to the ORB at initialization.

Property Name	`ORB.Arguments`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`NO`

POA Name

This is the name of the POA (Portable Object Adaptor) created for the server. This property is only used by the VisiBroker 4.x and Orbix2000 distributions (but see below for information pertaining to JacORB). Every server should have a unique POA name. The server UUID is used as the POA name when this field is left blank.

JacORB and the POA Name

On JacORB, the *POA Name* property is used to set *Implementation Name* property used by the Naming Service.

To federate two separately-installed Naming Services running on JacORB, each service must have different Implementation Name. The following parameter can be passed to override the Implementation Name when the service is started:

```
-Djacorb.implname=<name>
```

where `<name>` is the required Implementation Name. This does *not* override the *POA Name*.

Property Name	`POA.Name`
Property Type	`FIXED`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Server Port

The server will run on a fixed port number when this option is set. The port number is that which the server will use to listen for requests.

A fixed port number allows for inter-ORB interoperability and enables servers to run without a daemon. Fixed ports also make it easier to implement security measures such as firewalls.

Property Name	`Port`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Server Process ID

A unique identifier (*UUID*) associated with a specific server process. This ID is used to identify every object belonging to the process.

Property Name	`PID`
Property Type	`FIXED`

Data Type	UUID
Accessibility	READ/WRITE
Mandatory	NO

3.5 Security Properties

The properties on the *Security* tab relate to securing OpenFusion Services.

For convenience, these properties have been placed in a separate section. See Section 10, *Security Configuration*, on page 133.

3.6 Java Properties

These properties relate to the Java Virtual Machine (JVM) that runs the OpenFusion Services.

JVM Information

This property displays information about the Java Virtual Machine that the Service is running in, for example:

```
build JDK-1.2.2_006, native threads, symcjit
```

This information is only displayed while the service is running.

Property Name	JVM.Info
Property Type	DYNAMIC
Data Type	STRING
Accessibility	READ ONLY
Mandatory	NO

JVM Flags

These flags are passed to the Java Virtual Machine used to run the Service.

Property Name	JVM.Flags
Property Type	STATIC
Data Type	STRING
Accessibility	READ/WRITE
Mandatory	NO

JVM Free Memory

Displays the free memory available to the Java Virtual Machine that the Service is running in.

This information can only be refreshed while the Service is running.

Property Name	`JVM.FreeMemory`
Property Type	`DYNAMIC`
Data Type	`INTEGER`
Accessibility	`READ ONLY`
Mandatory	`NO`

JVM Total Memory

Displays the total memory available to the Java Virtual Machine that the Service is running in.

This information can only be refreshed while the Service is running.

Property Name	`JVM.TotalMemory`
Property Type	`DYNAMIC`
Data Type	`INTEGER`
Accessibility	`READ ONLY`
Mandatory	`NO`

Use Xbootclasspath

If this property is checked, the JVM is passed the `-Xbootclasspath` flag when the service is started.

The `-Xbootclasspath` flag causes the service to use the OpenFusion classes and not those supplied with the JRE. A side-effect of using `-Xbootclasspath` is the inability of the JVM to find shared libraries.

For a Singleton to register itself with a running Naming Service when it is started, this property must be checked.

Property Name	`JVM.XBoot`
Property Type	`STATIC`

Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	NO

3.7 System Properties

These properties relate to the system that OpenFusion runs on.

User Name

Displays the name of the user running the process.

This information is only displayed while the Service is running.

Property Name	User.Name
Property Type	DYNAMIC
Data Type	STRING
Accessibility	READ ONLY
Mandatory	NO

System Type

Displays the operating system type.

This information is only displayed while the Service is running.

Property Name	System.Type
Property Type	DYNAMIC
Data Type	STRING
Accessibility	READ ONLY
Mandatory	NO

System Name

Displays the name of the system running the Service (the computer name). This information is only displayed while the Service is running.

Property Name	System.Name
Property Type	DYNAMIC

Data Type	STRING
Accessibility	READ ONLY
Mandatory	NO

3.8 Common Singleton Properties

These properties are used to specify the location for reading and writing the Singleton's IOR.

The method used to read and write the IOR file will depend on which properties have been completed.

Reading the IOR

The rules for reading the IOR are, in order of precedence:

1. The IOR will be read from the location specified in the *IOR URL* property.
2. If the *IOR URL* property is blank, the IOR will be read from the naming service specified in *IOR Name Service*, under the name specified in *IOR Name Service Entry*.
3. If *IOR Name Service Entry* is blank, the IOR will be read from the location specified in the *IOR File Name* property.

Writing the IOR

The rules for writing the IOR are, in order of precedence:

1. The IOR will be written to the location specified in the *IOR File Name* property.
2. If *IOR Name Service Entry* is not blank the IOR will be written to the naming service specified in *IOR Name Service*, under the name specified in *IOR Name Service Entry*.

IOR Name Service

The name of the Naming Service which will be used to resolve the Singleton object. This defaults to *NameService*, which is the resolve name of the OpenFusion Naming Service, and should only be changed if the name service is being resolved using a different name.

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

IOR Name Service Entry

The naming service entry for the Singleton, in INS format (Interoperable Naming Service stringified name). This name will be written to the naming service specified in the *IOR Name Service* property.

Any intermediary naming context must already exist in the naming service. For example, to write Singleton "b" to the naming service as follows:

```
R/a/b
```

the context "a" must already exist.

This property has no default value, and if it is left blank the Singleton will not be written to the naming service.

Property Name	IOR.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.

Currently, only *http* and *file* URLs are supported.

This property defaults to:

```
file:/<install>/domains/<domain>/<node>/<service>/<singleton>/<singleton>.ior
```

where `<install>` is the OpenFusion CORBA Services installation path. See *The Object Hierarchy* on page 153 for details of the `domains` directory structure.

The *IOR URL* can only be used when reading the IOR. The IOR cannot be written to a location specified in a URL; the *IOR File Name* property should be used instead.

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. This defaults to:

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

where `<install>` is the OpenFusion CORBA Services installation path. See *The Object Hierarchy* on page 153 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 name used to locate the Singleton using `resolve_initial_references`.

The *Resolve Name* of the Naming Service Singleton must be unique within the whole Domain.

ProcessSingletons do not have this property.

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

3.9 Administration Manager Properties

Properties set at the root level of the *Object Hierarchy* govern the operation of the Administration Manager.

Although administration properties are shown as dynamic in the Administration Manager, in order for changes to those properties to take affect they must be saved, and the Administration Manager must be shut down and re-started.

To set Administration Manager Properties, select the *Domains* node (root node) in the Object Hierarchy. The properties, described below, are shown on the following tabs:

- *CORBA* contains properties that relate to the CORBA ORB.
- *LOGGING* contains the logging properties (see Section 3.3, *Logging Properties*, on page 55).
- *GENERAL* contains properties specific to the Administration Manager.
- *CONFIGURE* contains the properties for setting up OpenFusion to run remotely from a central configuration server (see Section 2.5, *Distributed Installation Configuration*, on page 36).
- *SERVICE LOG* tab displays the Browser Log as described in *Service Log* on page 30.

CORBA Properties

POA Name

This is the name of the POA (Portable Object Adaptor) created for the server. Every server should have a unique POA name. The server UUID is used as the POA name when this field is left blank.

The default value is `OpenFusion.Manager`.

Property Name	`POA.Name`
Property Type	`DYNAMIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Server Port

The Server port that the Administration Manager attempts to use when started with the `-port` command line switch. See *-port* on page 12.

Property Name	`Port`
Property Type	`DYNAMIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Configure Properties

These properties allow the configuration of OpenFusion installations from a central host. This is described in Section 2.5, *Distributed Installation Configuration* on page 36.

Central Configuration Host

If this server is to act as the central configuration manager for remote hosts, then this check box must be checked. The location of the configuration file on the local host must be entered in the *OpenFusion Install URL* field.

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

OpenFusion Install URL

The URL of the OpenFusion installation on a central configuration host.

Caution: entering an invalid URL will cause fatal problems! Take backups of the OpenFusion installation and be very careful when changing this property.

Property Name	`OpenFusionInstallURL`
Property Type	`DYNAMIC`
Data Type	`URL`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Remote OpenFusion Domains URL

The URL of the OpenFusion configuration file on a central configuration host.

Property Name	RunOpenFusionDomainsURL
Property Type	DYNAMIC
Data Type	URL
Accessibility	READ/WRITE
Mandatory	YES

Configure From Remote Host

If the OpenFusion configuration is to read from a central host, then this check box must be checked. The location of the configuration file on the remote host must be entered in the *Remote OpenFusion Install URL* field.

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

General Properties

Pre-load Properties

If this option is selected, the performance of the browser will improve. The disadvantage is that the browser takes slightly longer to load when first started. For best performance, we recommend that this option is always selected (which is the default value).

Property Name	PreLoadProperties
Property Type	DYNAMIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	NO

Service Timeout

The timeout interval (in seconds) when starting Services, after which the Administration Manager stops checking the Service node to see if it is started. If the service has not started, then it is flagged as “failed to start”. The default value is 60 seconds.

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

Status Timeout

The timeout interval (in seconds) which is allowed for a response when checking checking the status of servers. The default value is 2 seconds.

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

4 Instrumentation

4.1 Overview

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 at run time using the Administration Manager, SNMP or CORBA `Process` interface.

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

Manageable Resources

An OpenFusion manageable resource is a CORBA Singleton or Java Object that can be managed at runtime via SNMP (see *SNMP Agent* on page 78) or using the CORBA `Process` interface (*CORBA Process Interface* on page 84).

The CORBA Singletons listed below are manageable resources:

- ProcessSingleton (the default Singleton within each Service)
- NameSingleton
- LoadBalancingFactorySingleton
- TradingSingleton
- ServiceTypeRepositorySingleton
- NotificationSingleton
- TimeSingleton

The following Java Objects are manageable resources:

- SNMPAgentObject (the SNMP agent can be managed via SNMP)

Object Counters

The Object Counters provided for each managed Singleton or Java Object (for example, the *Number of Event Channels* property of the *NotificationSingleton*) give a count of the number of objects in existence. The counter is incremented when an object is created and decremented when the object is destroyed.

The destruction of an object occurs during garbage collection, not when the object is de-referenced. Therefore, there will be a delay between an object being de-referenced and the counter registering that it has been destroyed.

4.2 SNMP Agent

The SNMP agent is a Java Object that enables SNMP management applications to access the properties of manageable resources at runtime via SNMP. The OpenFusion SNMP agent implements SNMPv1 and uses UDP as the underlying transport protocol for sending and receiving SNMP messages.

To use the SNMP agent, the *SNMPAgentObject* must be added to a Service in the Administration Manager. Adding Java Objects to a Service is described in *Adding Singletons and Java Objects* on page 20. The SNMP agent enables all manageable resources that are co-located with it to be managed via SNMP.

Configuring the SNMP Agent

The following properties of the *SNMPAgentObject* can be configured from the Administration Manager.

Port

The port used by the agent to listen for SNMP requests. The standard port for listening for SNMP requests is port 161.

Property Name	`Port`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	161

Max Packet Size

The maximum packet size (in bytes) of an SNMP message.

Warning: If the packet size is configured to be too small then the SNMP agent may fail with an exception when attempting to process an SNMP message whose size exceeds the maximum packet size.

Property Name	MaxPacketSize
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES
Default Value	1300

Max Active Clients

The maximum number of clients that can access the agent concurrently. A value less than one is interpreted to mean that there is no limit to the number of clients that can access the SNMP agent concurrently.

Property Name	MaxActiveClients
Property Type	STATIC
Data Type	INTEGER
Accessibility	READ/WRITE
Mandatory	YES
Default Value	10

Enable Traps

A flag indicating if the agent will send SNMP traps. By default the SNMP agent does not send traps.

Property Name	EnableTraps
Property Type	STATIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	YES
Default Value	FALSE

Trap Hosts File

An XML file defining hosts to receive traps sent by the agent. See *Trap Hosts File* on page 82 for a description of this file.

Property Name	`TrapHostsFile`
Property Type	`STATIC`
Data Type	`FILE`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	none

Default Trap Port

The port to send traps to when a port is not specified in the XML. The standard port for listening for SNMP traps is port 162.

Property Name	`DefaultTrapPort`
Property Type	`STATIC`
Data Type	`INTEGER`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	162

Default Trap Community

The community name used for sending traps when a community name is not specified in the XML. For security reasons this property cannot be monitored via SNMP.

Property Name	`DefaultTrapCommunity`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	none

Trap On Authentication Failure

A flag indicating if the agent sends a trap when an authentication failure occurs. Regardless of the value of this property, the SNMP agent will only send traps if the *EnableTraps* property is set to TRUE.

Property Name	`TrapOnAuthenticationFailure`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	TRUE

Read-only Community

The community name of the agent providing read-only access to the MIB view. For security reasons this property cannot be monitored via SNMP.

Property Name	`ReadOnlyCommunity`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	none

Enable Write Access

A flag indicating if the agent will allow write access. By default the SNMP agent does not allow write access.

Property Name	`EnableWriteAccess`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	FALSE

Read-write Community

The community name of the agent providing read-write access to the MIB view. For security reasons this property cannot be monitored via SNMP. Requests that use this community when the *EnableWriteAccess* property is set to FALSE will result in an authentication failure.

Property Name	`ReadWriteCommunity`
Property Type	`STATIC`
Data Type	`STRING`
Accessibility	`READ/WRITE`
Mandatory	`YES`
Default Value	none

Notifications

The following notifications are sent by the SNMP agent:

- `SnmpAgentStartup`

  This notification signals that the SNMP agent has started and is listening for SNMP requests.

- `AuthenticationFailure`

  This notification signals that an authentication failure has occurred while processing an SNMP message.

- `SnmpAgentShutdown`

  This notification signals that the SNMP agent has stopped and is no longer listening for SNMP requests.

Notifications are sent to SNMP management applications as SNMPv1 traps. These traps are enterprise specific and do not belong to the *snmp* group in *MIB-II*. Consult the SNMP agent MIB for details of all traps sent by the SNMP agent.

Trap Hosts File

The trap hosts file is an XML file that defines the hosts that are to receive SNMPv1 traps emitted by the SNMP agent. The full path to this file is specified by the *TrapHostsFile* property (page 80). If the *EnableTraps* property (page 79) is set to FALSE then the *TrapHostsFile* property is ignored by the SNMP agent.

Each host is defined in the XML by a host name and optionally a port and community name. If the port is not specified then the value of the *DefaultTrapPort* property (page 80) is used. Similarly, if the community name is not specified then the value of the *DefaultTrapCommunity* property (page 80) is used. The host name must be specified and can either be the name or the IP address of the host.

The DTD that specifies the format of the XML is defined in the following file:

```
<INSTALL>/xml/schema/TrapHosts.dtd
```

where `<INSTALL>` is the OpenFusion installation directory.

A `DOCTYPE` declaration referencing `TrapHosts.dtd` must be included in every XML file so that the XML can be validated. If this declaration is not included then the SNMP agent will fail to start.

Starting the SNMP Agent

Once an *SNMPAgentObject* has been added to a Service, starting the Service automatically starts the SNMP agent. Immediately after the agent has started it will send an `SnmpAgentStartup` trap (if traps are enabled) and it will begin listening for incoming SNMP requests.

Manageable resources that are co-located with the *SNMPAgentObject* must be started before the *SNMPAgentObject* in order to be managed via SNMP. This can be accomplished from within the Administration Manager by positioning all co-located resources above the *SNMPAgentObject* in the Service definition. The exception to this starting rule is the *ProcessSingleton*, which is normally the last resource in a Service definition.

Stopping the SNMP Agent

Stopping the Service containing the *SNMPAgentObject* automatically stops the SNMP agent. Immediately before the agent has stopped it will send an `SnmpAgentShutdown` trap (if traps are enabled) and it will stop listening for incoming SNMP requests.

OpenFusion MIBs

The OpenFusion MIBs are contained in the following directory:

```
<INSTALL>/mibs
```

where `<INSTALL>` is the OpenFusion installation directory.

There is one MIB for each OpenFusion manageable resource. The name of each MIB is prefixed with the name of the resource it describes (minus the *Singleton* or *Object* suffix). For example, the MIB representing the *TradingSingleton* is named `TRADING-MIB.txt`, and the MIB representing the *SNMPAgentObject* is named `SNMPAGENT-MIB.txt`. The exception to this naming rule is the *ProcessSingleton* MIB, which is named `SERVER-MIB.txt`.

The OpenFusion MIBs fully conform to SMIv1. Management applications connected to the SNMP agent will be able to access the objects defined in the MIBs of co-located resources by their OIDs. The OID of the root node of the PrismTech OpenFusion MIB tree is `1.3.6.1.4.1.5510.1`.

4.3 CORBA Process Interface

The CORBA *`Process`* interface can be used to programmatically monitor the system by accessing the instrumentation properties of individual service instances.

The services in this release which support the use of the `Process` interface are listed below in *Table 7*. The table also lists the names used by `Process`' methods to access the services.

Table 7 Services' Access Names

Service Singleton	Access Name
`ProcessSingleton` *`(default service singleton)`*	Server
`NotificationSingleton`	Notification

Using the Process Interface

An instance of the Process interface can be used to programmatically obtain property values for any instrumentation-enabled singletons which are co-located with the `Process` object.

The following steps describe how to use the Process interface in a program or module for obtaining instrumentation property values for a service instance.

Step 1: Ensure the program module imports the following packages:

```
com.prismt.orb.ObjectAdapter
com.prismt.openfusion.Server.Process
com.prismt.openfusion.Server.ProcessHelper
```

Step 2: Perform the standard ORB initialisation, for example:

```
static org.omg.CORBA.ORB orb = ObjectAdapter.init (new String[0]);
```

Step 3: Obtain a reference to the local Process interface instance by retrieving the instance's IOR from a file called *ProcessSingleton.ior* (located in the *<install_dir>/domains/OpenFusion/localhost/<service>/ProcessSingleto*n directory). The ORB's *string_to_object()* method is used to convert the stringified version of the IOR, which the file contains, to the needed object reference.[1]

Example

```
String iorPathName = "ProcessSingleton.ior";
BufferedReader in = new BufferedReader (new FileReader (iorPathName));
String iorString = in.readLine ();
in.close ();
org.omg.CORBA.Object obj = orb.string_to_object (iorString);
Process processObject = ProcessHelper.narrow (obj);
```

The example code above assigns the needed `Process` object to *processObject*.

Step 4: Use the `Process` object's *getValue()* method to retrieve the desired property values for a service instance. Please note that each service instance is referenced as a named singleton object (see *Table 7* on page 84).

The *getValue()* method is given (as `Strings`) the *access name* of the service's singleton object as well as the name of the desired instrumentation property:

- the access name for each service's singleton object is given in Table 7, *Services' Access Names*, on page 84
- the instrumentation property names are listed under *Instrumentation Properties* in the *Configuration* section of the service's user guide; for example, those for the Notification Service are listed under *Notification Service Configuration, Instrumentation Properties* of the *Notification Service Guide*

The `getValue()` method returns an *any* which contains the property value. The contained value will be of the type (`String`, `long`, `ulonglong`, etc.) specified for the property, as listed under the *Instrumentation Properties* for the service referred to above: the value must be retrieved from the any using the appropriate `Any` extraction method, for example *extract_string()* for `Strings`, *extract_long()* for `longs`, *extract_longlong()* for `longlongs`, etc.

1. The code examples shown here use *BufferedReader* and *FileReader* for simplicity, although other file reading approaches could be used; modules using these packages must import the standard *java.io.BufferedReader* and *java.io.FileReader* packages.

Example

```
String service = "Notification";
String propertyName = "Channels";

org.omg.CORBA.Any any = processObject.getValue(propertyName, service);
long channelValue = any.extract_longlong();
System.out.println ("The value of the Channels property is: " + channelValue);
```

Example Program

The following example shows how instrumentation value can be displayed using a stand-alone program.

```
import com.prismt.orb.ObjectAdapter;
import com.prismt.openfusion.Server.Process;
import com.prismt.openfusion.Server.ProcessHelper;

import java.io.BufferedReader;
import java.io.FileReader;

// display instrumentation values using the CORBA Process interface
public class InstrumentationAccessor
{
   private org.omg.CORBA.ORB orb = null;
   private Process localProcess = null;

   // Constructor, where
   // iorPathName is the location of the file (pathname) containing the Process IOR
   public InstrumentationAccessor (String iorPathName)
   {
      orb = ObjectAdapter.init (new String[0]);

      // Obtain reference to the Process object using the stringified
      // IOR stored in the file defined in iorPathName
      try
      {
         BufferedReader in = new BufferedReader (new FileReader (iorPathName));
         String iorString = in.readLine ();
         in.close ();
         org.omg.CORBA.Object object = orb.string_to_object (iorString);
         localProcess = ProcessHelper.narrow (object);
      }
      catch (Exception e)
      {
         System.out.println ("Failed to obtain process.");
      }
   }

   // Obtain a property value using the Process.getValue() method, where
   // propertyName is the name of the property
   // service is the name of the service containing the property
   public org.omg.CORBA.Any getPropertyValue (String propertyName, String service)
      throws Exception
   {
      try
      {
         org.omg.CORBA.Any any = localProcess.getValue (propertyName, service);
         return any;
      }
      catch (Exception ex)
      {
         throw new Exception ("Failed to retrieve value of " +  propertyName
                              + " from " + service);
      }
```

```
   }

   // main ////////////////////////////
   public static void main (String[] args)
   {
      // check that pathname of Process IOR file provided by user
      if (args.length != 1)
      {
         System.out.println ("Please supply pathname of Process IOR file");
         System.exit (1);
      }

      // InstrumentationAccessor's constructor obtains a reference to the
      // local process using the stringified IOR stored in file provided by
      // the user as a command line parameter
      InstrumentationAccessor accessor = new InstrumentationAccessor (args [0]);

      // display instrumentation property values
      try
      {
         System.out.println ("\nDisplaying instrumentation property values.\n");

         // Server object properties values
         org.omg.CORBA.Any any = accessor.getPropertyValue ("JVM.FreeMemory", "Server");
         long freeMem = any.extract_long ();
         System.out.println ("JVM Free mem: " + freeMem);

         any = accessor.getPropertyValue ("JVM.Info", "Server");
         String info = any.extract_string ();
         System.out.println ("JVM info: " + info);

         // Service object properties values for the Notification Service
         any = accessor.getPropertyValue ("Channels", "Notification");
         long chans = any.extract_longlong ();
         System.out.println ("Channels: " + chans);

         any = accessor.getPropertyValue ("ProxyPushConsumers", "Notification");
         long ppc = any.extract_longlong ();
         System.out.println ("ProxyPushConsumers: " + ppc);

         any = accessor.getPropertyValue ("EventsDelivered", "Notification");
         long evsd = any.extract_longlong ();
         System.out.println ("Events delivered: " + evsd);
         Thread.sleep (2000);
      }
      catch (Exception ex)
      {
         ex.printStackTrace ();
      }
   }
}
```

5 Service Portability

The OpenFusion CORBA Services conform with the OMG defined Java bindings. However, there are some ORB and platform differences that must be taken into account when developing and running clients and servers. The following sections cover these issues:

- Portability Classes

  The OpenFusion framework supports a number of CORBA portability classes that normalise access to the underlying ORB and related classes.

- Running User Defined Clients and Servers

  This section covers both vendor and platform issues concerning the content of `CLASSPATH`, `PATH` and different parameters required for the command line when executing user defined clients and servers.

- OpenFusion IDL Compilation Issues

  This section covers both vendor and platform issues concerning the compilation of OpenFusion IDL when creating user defined servers, for example, creating event suppliers for the Notification Service.

- C++ Support

  This section gives some basic guidelines for the development of C++ clients for the OpenFusion CORBA services.

5.1 Portability Classes

The OpenFusion framework supports a number of CORBA portability classes that normalise access to the underlying ORB and related classes. There are a number of reasons why this has been done:

- Prior to the CORBA 2.3 specification, server side object mappings were not standardised and the generated server side support classes were different for each ORB vendor.
- It hides some of the complexity in using the ORB native object adapters (particularly with respect to the POA) and proprietary loaders.
- It supports the deployment of both transient and persistent objects and simplifies the management of a persistent object's state.
- It normalises the creation of and access to dynamic `Any` classes. This is required as these were repackaged in the CORBA 2.3 specification.

Three classes are used to support the development of ORB portable code:

- *ORBAdapter*: This provides a client side abstraction layer for initilializing and accessing the ORB and running client applications.
- *ObjectAdapter*: This provides a server side abstraction layer for managing server objects.
- *DynAnyFactory*: This provides a factory class for creating dynamic any objects. This class returns implementations of the dynamic any classes which conform with CORBA 2.3.

These three portability classes are in the `com.prismt.orb` package. The following sections describe each of these three classes in detail.

The ORBAdapter Class

The `ORBAdapter` class contains operations for ORB initialization and a number of utility operations to return information about object references.

ORB Initialization

Two static initialization operations are supported. The first takes an array of `String` arguments and is intended to be called from the `main` operation so that any arguments passed to an application may be passed onto the ORB when it is initialized. The second form takes an ORB parameter and should be called to initialize from a pre-existing full ORB implementation. These operations are defined as follows:

```
public static synchronized org.omg.CORBA.ORB init (String[] args)
   throws org.omg.CORBA.INITIALIZE
public static synchronized void init (org.omg.CORBA.ORB existing)
   throws org.omg.CORBA.INITIALIZE
```

These operations both return an instance of a full ORB. An operation is also provided to return the initialized ORB instance:

```
public static synchronized org.omg.CORBA.ORB getORB ()
```

A limited functionality singleton ORB will be returned when an `init` operation has not been previously called. Most of the other operations defined on this class will throw the `INITIALIZE` exception when the class has not been initialized via one of the `init` operations.

ORB Shutdown

A single operation is provided to shut down the ORB. The `ORBAdapter` class should not be used after this has been called.

```
public static synchronized void shutdown ()
```

Object Information

A number of operations are provided to query the status of an object reference. These operations are guaranteed to work only with objects created using the `ObjectAdapter` class.

```
public static boolean isProcessLocal (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean isNodeLocal (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean isActive (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static Boolean isActive (org.omg.CORBA.Object obj, int timeout)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean isPersistent (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean isTransient (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean isValid (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

A CORBA object reference is defined to be valid when it is either active or non transient.

When checking for whether an object is active, an ORB implementation may block for some time. A timeout value for the `isActive` operation is supported via the following two operations:

```
public static int getActiveTimeout ()
public static void setActiveTimeout (int timeout)
```

Object Stringification

Two operations are provided to convert object references to strings and vice versa. These operations are similar to the ORB operations `string_to_object` and `object_to_string` except that INS (Interoperable Name Service) format strings are also supported.

```
public static org.omg.CORBA.Object stringToObject (String str) throws
   org.omg.CORBA.INITIALIZE,
   org.omg.CORBA.BAD_PARAM
public static String objectToString (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE
```

Service Resolution

Two operations support the dynamic resolution of services from names:

```
public static org.omg.CORBA.Object resolve (String name)
   throws org.omg.CORBA.BAD_PARAM, org.omg.CORBA.INITIALIZE
```

The `resolve` operation resolves a CORBA object or service by name in a similar way to the ORB `resolve_initial_references` operation.

The ObjectAdapter Class

This class supports operations for the management of server objects. This adapter logically layers over either a BOA or POA depending on the ORB implementation.

Initialization

Two static initialization operations are supported. The first takes an array of `String` arguments and is intended to be called from the `main` operation so that any arguments passed to an application may be passed onto the ORB when it is initialized. The second form takes an ORB parameter and should be called to initialize from a pre-existing full ORB implementation.

These initialization operations also initialize the ORB through the corresponding operations defined in the `ORBAdapter` class. These operations are defined as follows:

```
public static synchronized org.omg.CORBA.ORB init (String[] args)
   throws org.omg.CORBA.INITIALIZE
public static synchronized void init (org.omg.CORBA.ORB orb)
   throws org.omg.CORBA.INITIALIZE
```

Object Creation

Object implementations should implement the `Operations` interface generated for the IDL interface that is being implemented. An object implementation must also implement the `java.io.Serializable` interface when it may be used as a persistent object. A serializable implementation may be used to create either transient or persistent objects. However, a non-serializable implementation may only be used to create transient objects.

Objects are created using either the `createPersistent` or `createTransient` operations on the `ObjectAdapter` class, as appropriate. Both operations return a CORBA object reference (`org.omg.CORBA.Object`) that may be narrowed to the appropriate type using the appropriate generated helper class.

A number of overloaded creation operations are supported but there are essentially two forms. The first, `createPersistent`, is used to create persistent CORBA objects and the second, `createTransient`, to create transient CORBA objects. Both operations have the general form:

```
public static org.omg.CORBA.Object createTransient
(
   java.lang.Object obj
   [,java.lang.Class opsClass]
   [,int flags]
   [,UUID id]
)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM

public static org.omg.CORBA.Object createPersistent
```

```
(
   Serializable obj
   [,java.lang.Class opsClass]
   [,int flags]
   [,UUID id]
)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

All the parameters, apart from the object implementation, are optional and are defined as follows:

- *`obj`*: The object implementation. This must be serializable for persistent objects.
- *`opsClass`*: The Operations class being implemented. By default, the most derived Operations class for an implementation will be discovered via reflection. However, the type of the operations class being supported must be specified when an implementation supports several, possibly unrelated, interfaces.
- *`flags`*: Creation option flags. A number of flags are supported that provide additional semantic behaviour for the created object. Currently this includes purging and activation policies and whether multiple CORBA objects can be created for a single implementation. The flag values are supplied as `final static int`s for the `ObjectAdapter` class and are intended to be combined using the `and` operator. These flags are described in Table 8, *ObjectAdapter Object Creation Flags* below.
- *`id`*: The identity of the created object. All OpenFusion CORBA objects use UUIDs for identity. By default, a new UUID is assigned for created objects. This parameter uses the provided UUID as the object identity for the created CORBA object.

Table 8 ObjectAdapter Object Creation Flags

Flag	Persistent	Transient	Description
DISABLE_AUTO_ACTIVATION	✓	X	This flag disables the auto-activation of persistent objects, i.e. on demand. A persistent object must be explicitly reactivated via the `ObjectAdapter reactivate` operation when this flag is set.
DISABLE_PURGE	✓	D	This flag disables the purging of persistent objects. By default, transient objects will not be purged as they cannot be reactivated
ENABLE_PURGE	D	✓	This flag enables the automatic purging of objects based upon the configurable purging options. A transient object is effectively destroyed when it is purged.
ENABLE_DUPLICATES	✓	✓	This flag allows a transient object implementation to have multiple CORBA object identities.

D = Default behaviour, ✓= Supported, X = Not supported.

Object Identity

All object implementations registered with the OpenFusion object adapter have an identity based upon the DCE UUID and are encapsulated within the `com.prismt.util.UUID` class. The `getId` operation returns an object identity from an object reference. The `getIds` operation returns an array of these identities when an implementation has multiple object identities. These operations are defined as follows:

```
public static UUID getId (java.lang.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

The `ObjectAdapter` class provides a number of overloaded operations that can use either an object identity or an object reference to identify a particular implementation (`deactivate`, `destroy`, `reactivate`, `exists`, and `getImplementation`).

Multiple Object Identity

A single Java object instance may implement any number of CORBA objects. This is supported by means of the `ENABLE_DUPLICATES` flag. An object implementation must implement the `com.prismt.orb.Multiplexable` interface when it is to be used for multiple CORBA objects. This interface consists of two operations:

```
public UUID getGroupId (Class opsClass);
public UUID getPrimaryGroupId ( );
```

An implementation must maintain, as part of its state, a unique identifier for each of the operations interfaces that it supports. These unique identifiers must be made available through the `getGroupId` operation. The first such identifier issued must be recorded and returned by subsequent calls to `getPrimaryGroupId`.

An implementation may need to determine the identity of the object being called when the implementation has been used to create multiple CORBA objects. The following operation supports this functionality:

```
public static UUID getCallerId ()
```

This operation should only be called within the context of an invoked operation on an implementation.

Object Deactivation

Transient object references are only valid while their implementations exist so deactivating a transient object is equivalent to destroying it. Any references to a deactivated transient object become invalid. Persistent objects store their state thus allowing implementations to be activated and deactivated any number of times (when deactivated, an object's implementation has been deleted but its state remains).

```
public static void deactivate (java.lang.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static void deactivate (UUID id)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static void deactivate (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

Object Destruction

All objects can be destroyed so that any references to them are no longer valid. For persistent objects, the object's state is also destroyed. The `ObjectAdapter` operations that support this are defined as follows:

```
public static void destroy (java.lang.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static void destroy (UUID id)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static void destroy (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

Object Reactivation

All persistent objects can be deactivated and reactivated as required. Either an object reference or an id can be used to reactivate an object implementation. The `ObjectAdapter` operations that support this are defined as follows:

```
public static Serializable reactivate (UUID id) throws
   org.omg.CORBA.INITIALIZE,
   org.omg.CORBA.OBJECT_NOT_EXIST,
   org.omg.CORBA.BAD_PARAM
public static Serializable reactivate (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.OBJECT_NOT_EXIST
```

The `OBJECT_NOT_EXIST` exception is thrown when the object cannot be reactivated, e.g. when the object is not persistent or has been destroyed.

Object Existence

Two operations are provided to determine whether or not an object implementation exists for a given object reference or identity:

```
public static boolean exists (UUID id)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static boolean exists (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

These operations will return `true` when a persistent object exists, whether or not it is currently active.

Object References

Object implementations may be associated with one or more CORBA object references. Two operations are supported to return the reference(s) associated with a particular implementation:

```
public static org.omg.CORBA.Object getObject (java.lang.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

Object Implementations

Two operations are provided to return an object implementation class given an object identity or reference:

```
public static java.lang.Object getImplementation (org.omg.CORBA.Object obj)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static java.lang.Object getImplementation (UUID id)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

These operations are typically used where co-located persistent object implementations may need to refer to each other.

Persistent Object State

Persistent object implementations may have state and, by default, this is managed by OpenFusion Object Adapter using serialization (persistent object implementations must implement *Serializable*). The state of a persistent object is stored when it is first created so the state can be restored whenever it is reactivated. A persistent object must ensure that its state is saved whenever a persistent implementation's state changes (typically through a client invoking some operation). Two operations, `save` and `write`, are supported on the `ObjectAdapter` class to support this. These operations are defined as follows:

```
public static void save (Serializable entity)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
public static void write (Serializable entity)
   throws org.omg.CORBA.INITIALIZE, org.omg.CORBA.BAD_PARAM
```

Both these operations write out the implementation objects state via serialization. The `save` operation may be buffered due to configured caching policies, i.e. an asynchronous operation, whereas the `write` operation ensures that the state has been written out to persistent store, i.e. a synchronous operation.

All the non-transient attributes of a persistent implementation must be serializable. However, the PrismTech object adapter can deal with a number of non-serializable CORBA data types through the use of specialized input and output streams. The following CORBA data types may be held as attributes of a persistent implementation:

- `org.omg.CORBA.Any`, and
- `org.omg.CORBA.TypeCode`.

Two approaches are possible when a persistent object wishes to store persistent references to other CORBA objects. The object identity (UUID) can be stored and then used to remap back to the server object and narrowed to the correct type on restoration. Alternatively, the object reference can be stringified when stored and destringified and narrowed when restored.

Persistent object implementations may implement `javax.ejb.EntityBean`. This interface specifies a number of callback operations that are invoked by the object adapter on a persistent implementation when a state change has occurred or is about to occur. Callbacks are invoked as follows:

- The `ejbStore` operation is called just before an implementation is saved.
- The `ejbLoad` operation is called just after an implementation has been restored.
- The `ejbActivate` operation is called just after an implementation has been created or reactivated.

- The `ejbPassivate` operation is called just before an implementation is deactivated.
- The `ejbRemove` operation is called just before an implementation is destroyed.

The `com.prismt.ejb.EntityBeanAdapter` abstract class provides default implementations of all these operations. Persistent implementations can simply extend this class and only re-implement any callbacks they wish to use.

Running a Server

Incoming requests from clients are not processed until the server is started when object implementations have been registered with the `ObjectAdapter`. A server is started with the `ready` operation and stopped with the `shutdown` operation, defined as follows:

```
public static void ready ()
   throws  org.omg.CORBA.INITIALIZE
public static void ready (boolean block)
   throws  org.omg.CORBA.INITIALIZE
public static void shutdown ()
```

The first form of the `ready` operation blocks; the second takes a boolean parameter that determines whether the operation should block or not.

Restrictions

Most operations on the `ObjectAdapter` class should not be called until after it has been initialized with the `init` operation or an `org.omg.CORBA.INITIALIZE` system exception will be thrown.

Most operations on the `ObjectAdapter` class will throw `org.omg.CORBA.BAD_PARAM` when invalid parameters or unexpected null parameters are passed.

A persistent object should not use any of the object related operations on `ObjectAdapter` from within the implementation of the `readObject` operation used in object serialization. A persistent implementation that needs to use ObjectAdapter operations on restoration, e.g. when mapping from persistent object UUIDs to implementations, should implement `javax.ejb.EntityBean` and put this functionality in the `ejbLoad` operation.

Recommendations

Use the factory design pattern to create all CORBA object implementations, i.e locate the `create` operation in a factory rather than in an implementation constructor. This allows:

- the object implementation to be deployed where appropriate as an EJB rather than a CORBA object
- a factory to create an implementation as either a transient or persistent object (as long as it implements `Serializable`)
- one implementation to extend another

Try not to use any object-based `ObjectAdapter` operations in:

- constructors
- serialization operations (`readObject` and `writeObject`)

When persistent implementation classes need to reference each other, they should:

- store the object id for the related implementation when saved
- use the `reactivate` and `getImplementation` operations to retrieve a reference to the related implementation class when restored

Persistent implementations should re-implement the `ejbActivate` operation, which is called by the object adapter after a persistent implementation has been reactivated and its state restored, when they need to do some further initialization after reactivation.

Most of the `ObjectAdapter` class operations can throw CORBA system exceptions. These are all derived from `RuntimeException` and so no try-catch block is required in the calling code. However, users of the `ObjectAdapter` class should be aware that these may be thrown in some circumstances. The causes and types of exception thrown are described fully in the javadoc documentation for the class.

The DynAnyFactory Class

This class supports creation operations for the CORBA 2.3 defined dynamic any classes.

Creation Operations

Six operations are provided for the creation of dynamic any objects from a given type code. The `InconsistentTypeCode` exception is thrown when the provided type code does not correspond to the type of the requested dynamic object.

```
1: public static org.omg.DynamicAny.DynAny createBasicDynAny (TypeCode tc)
2:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
3: public static org.omg.DynamicAny.DynStruct createDynStruct (TypeCode tc)
4:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
5: public static org.omg.DynamicAny.DynSequence createDynSequence (TypeCode tc)
6:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
7: public static org.omg.DynamicAny.DynArray createDynArray (TypeCode tc)
8:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
```

```
 9: public static org.omg.DynamicAny.DynUnion createDynUnion (TypeCode tc)
10:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
11: public static org.omg.DynamicAny.DynEnum createDynEnum (TypeCode tc)
12:    throws org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
```

Implementing an Interface

A simple example is presented to demonstrate the use of the OpenFusion object adapter. This consists of two interfaces: a `Counter` interface that implements simple counter functionality, and, a `CounterFactory` interface that is used to create `Counter` objects with the ability to set their initial count. These interfaces are defined as follows:

```
 1: #pragma prefix "prismt.com/cos/CosLifeCycle/examples"
 2:
 3: module Counters
 4: {
 5:    interface Counter
 6:    {
 7:       attribute long count;
 8:    };
 9:
10:    interface CounterFactory
11:    {
12:       Counter createCounter (in long initial);
13:    };
14: };
```

The OpenFusion object adapter supports both persistent and transient objects. To demonstrate the usage of both, the `Counter` interface is implemented as a persistent object whereas the `CounterFactory` interface is implemented as a transient. The `Counter` interface is implemented by the `CounterImpl` class as follows:

```
 1: package com.prismt.cos.CosLifeCycle.examples.Counters;
 2:
 3: import java.io.Serializable;
 4: import com.prismt.orb.ObjectAdapter;
 5:
 6: public class CounterImpl implements CounterOperations, Serializable
 7: {
 8:    public CounterImpl ()
 9:    {
10:       value = 0;
11:    }
12:
13:    public CounterImpl (int val)
14:    {
15:       value = val;
16:    }
17:
18:    public int count ()
19:    {
20:       return value;
21:    }
22:
23:    public void count (int count)
24:    {
25:       value = count;
26:       ObjectAdapter.save (this);
27:    }
28:
```

```
29:    private int value;
30: }
```

This implementation simply implements the operations defined in the `CounterOperations` class and uses the `ObjectAdapter save` operation to save its state when the count changes. The `save` operation will have no effect when an implementation is created as a non-persistent, i.e. transient, object.

The `CounterFactory` interface is implemented by the `FactoryImpl` class as follows:

```
 1: package com.prismt.cos.CosLifeCycle.examples.Counters;
 2:
 3: import org.omg.CosLifeCycle.*;
 4: import com.prismt.orb.ObjectAdapter;
 5:
 6: public class FactoryImpl implements CounterFactoryOperations
 7: {
 8:    public static final void main (String[] args)
 9:    {
10:       ObjectAdapter.init (args);
11:
12:       ObjectAdapter.createTransient (new FactoryImpl ());
13:
14:       ObjectAdapter.ready ();
15:    }
16:
17:    public Counter createCounter (int count)
18:    {
19:       org.omg.CORBA.Object ref;
20:
21:       ref = ObjectAdapter.createPersistent (new CounterImpl (count));
22:
23:       return (CounterHelper.narrow (ref));
24:    }
25: }
```

This implementation implements the `createCounter` operation defined in the `CounterFactoryOperations` interface. This operation simply creates a new `Counter` implementation as a persistent CORBA object. The factory object itself is created as a transient CORBA object in `main`.

Persistent Servers

The OpenFusion ORB portability framework uses UUIDs to identify both persistent objects and persistent servers that contain these objects. Persistent objects are created within the context of a persistent server. When a server is running, persistent objects may be deactivated and reactivated, on demand, any number of times until they are explicitly destroyed. Persistent objects can also maintain their state across the cycle of starting and stopping persistent servers. To do this, a persistent server must be coded, or configured, so that it has the same identity each time it is started.

Servers must be registered with the ORB as `OpenFusion.<uuid>`, where `<uuid>` is the server's UUID.

Server identity is encapsulated within the `com.prismt.util.PID` class. This has a `setPID` operation that can be used to hard code the identity of a server. This must be done before the ORB is initialised.

```
package com.prismt.orb.examples;

public class Server
{
   public static void main (String args[])
   {
      com.prismt.util.PID.setPID ("43fe0080-9b6c-11d4-9727-af67c68e5b18");
      com.prismt.orb.ObjectAdapter.init (args);
      com.prismt.orb.ObjectAdapter.ready ();
   }
}
```

Alternatively, the Java system property `Process.PID` can be set as a JVM command line parameter to determine the id of a server. This has the advantage over the hard coded approach in that it is possible to run multiple instances of the same server class with different identities.

```
% run -DProcess.PID=43fe0080-9b6c-11d4-9727-af67c68e5b18
  MyServer
```

Persistent state must also be configured in order to use persistent servers and objects. (See Section 6, *Configuring Persistent Storage*, on page 107.)

5.2 Running User Defined Clients and Servers

Resolving Services

The OpenFusion Java examples use the `resolve_initial_references` method to access the individual CORBA services. The examples and other clients must be run in a manner that is specific to each ORB vendor in order for this mechanism to be correctly initialised. The following sections assume JacORB 1.4.

Services can be resolved either via static configuration or dynamically via an ORB initialisation class.

Configuration

IORs created for persistent services may be configured as initial references. The following example shows how this can be done directly via a configuration file entry.

```
ORBInitRef.TimerEventService=IOR:000000000000003049444c3a6f6d672e6f72672f43
6f7354696d65724576656e742f54696d65724576656e74536572766963653a312e300000000
0010000000000000a2000102000000000756c74726135000062b80000000000463a3e0232
3106756c74726135174f70656e467573696f6e2e54696d655365727669636500212105240520
0b3ef11d5b154d7b9de5c8185028f7790b3ef11d5b154d7b9de5c818500000000000030000000
0000000080000000049545f4100000001000000180000000000010001000000000000101040
000001000101090000000600000006000000000023
```

Alternatively, a utility class has been provided to help in the generation of a configuration file from the generated IOR files. This can be run as follows:

```
% java com.prismt.openfusion.orb.ConfigGen jacorb
  domain_xml out_file
```

This reads in all the configured services within an OpenFusion domain identified by the `domain_xml` file or URL, and writes out a JacORB configuration file, `out_file`, for these services. This file can then be appended to the standard JacORB properties configuration file.

Dynamic Registration

Services may be registered using a portable interceptor ORB initialisation class. The following Java system property should be defined:

```
-Dorg.omg.PortableInterceptor.ORBInitializerClass.com.
prismt.orb.portable.jacorb14.Initializer
```

Jar Files

The `CLASSPATH` must contain the following jar files when using OpenFusion services or examples:

Table 9 Jar Files

ORB	Required Jar Files	ORB Subdirectory
JacORB 1.4	*jacorb.jar*	*lib*

Using OpenFusion Run Scripts

A script file named **run** has been provided on UNIX systems (named **run.bat** on NT) to simplify the command line execution of classes used in conjunction with OpenFusion. The script will add the required standard properties to the command line; the user has only to consider properties and parameters required for the execution of the target class.

Command Line Format

Example command line formats when using the scripts, where *classname* is replaced by the required Java client class:

Classname only

```
%  run classname
```

Classname with parameters and/or properties

```
%  run [-x] [-d] [-s] –DmyDef=adef classname myParam
```

The `-x` parameter runs the JVM using the `-Xbootclasspath` flag to avoid problems with CosNaming classes supplied as part of the JDK.

The `-d` parameter runs the class in debug mode. This mode displays debug information on the console for any client class which implements `debug.debug` from the `log4j` logging package.

The `-s` parameter enables security for the client. The run script sets the property `-DSecurityEnabled=true`. The OpenFusion Security Service is described in *Section 8* on page 123.

5.3 OpenFusion Java IDL Compilation

There are a number of issues to consider when compiling interfaces using any of the OpenFusion defined services. These are demonstrated in the following Notification Service example, *News.idl*.

```
 1: #include <CosNotifyComm.idl>
 2:
 3: #pragma prefix "prismt.com/cos/CosNotification/examples"
 4:
 5: module News
 6: {
 7:    interface Bureau : CosNotifyComm::StructuredPushSupplier
 8:    {
 9:       void broadcast ();
10:       void stop ();
11:    };
12:
13:    interface Listener : CosNotifyComm::StructuredPushConsumer
14:    {
15:       void select (in string bureau);
16:       void print ();
17:    };
18: };
```

The elements to be considered when compiling *News.idl* and other similar IDL files are:

- IDL Compilers and Definitions

  Each ORB vendor has its own IDL to Java compiler and the code generated may be ORB specific. All ORB vendor specific includes, definitions and fixups are handled in a `orbdefs.idl` file. This is included where required by all other IDL files supplied as part of the distribution. When using the IDL compiler to compile any of these files, the appropriate ORB must be identified with a **`-D`** parameter. The values currently supported are.

Table 10 ORB Definitions

ORB	Definition
`-DJACORB14`	JacORB 1.4

- Package Specification

  The compiler has to be provided with the specifications for the OpenFusion packages that will be used in this module. The example below illustrates the required format for an included package.

```
-i2jpackage CosNotifyComm:org.omg.CosNotifyComm
```

- The compiler needs to be informed of the location of the package being compiled as well as provided with details of the location of external packages.
- Include Directories

  An include path must be provided with the **`-I`** flag for any included IDL file. All OpenFusion service IDL files are provided in the idl subdirectory of a distribution. ORB vendor IDL files may also be required.

Table 11 IDL Includes

ORB	Include Subdirectory
JacORB 1.4	*`idl`*

- Output Directory

  An output directory, where the compiler can place the generated Java files, is usually specified:

```
-d outputDir
```

5.4 C++ Support

OpenFusion services may be used from C++ clients. The client side stubs must be compiled from the provided IDL service definitions in order to do this. Please see your ORB vendor's documentation and examples for full details on how to do this. Some general guidelines are provided below.

```
#include <CosPropertyService.idl>

module Example
{
   interface MyServer : CosPropertyService::PropertySet
   {
      void printAllProperties();
   };
};
```

The elements to be considered when compiling IDL files using C++ are:

- IDL Compilers and Definitions

  Each ORB vendor has its own IDL to C++ compiler and the code generated is ORB specific. Any OpenFusion IDL specifying definitions must also be compiled for the ORB and language used, i.e. with a **`-D`** parameter. A set of orb/platform specific **`.mk`** files can be found in the `<OPENFUSION>/etc` directory where the appropriate idl compiler and **`-D`** parameters are already set. The Makefiles within the examples include the appropriate file from this directory.

- Include Directories

  An include path to the location of the standard OMG IDL files must be provided as an OpenFusion interface is inherited. This will be the same for any vendor but the format will vary depending upon the platform.

  For Solaris:

```
INCS += -I<OPENFUSION>/idl
```

  Again, the **`.mk`** files found in the `<OPENFUSION>/etc` directory include the standard OMG and OpenFusion IDL. Only user defined IDL needs to be specifically included in Makefiles when the `/etc/*.mk` files are included.

Should you require further assistance with developing C++ clients or code examples, see the OpenFusion support page at: *http://www.prismtechnologies.com/Contacts*

6 Configuring Persistent Storage

OpenFusion CORBA Services supports persistent storage via JDBC access to a relational database. Oracle, Sybase, Informix, and hsqldb are supported on both Unix and Windows platforms. Microsoft SQL Server is supported on Windows. See the *Product Guide* for details of supported versions.

Persistent storage is configured for each Service using the following properties on the *Persistence* tab of the Administration Manager:

- Storage Write Interval
- Storage Write Batch Size
- JDBC Handler
- JDBC Database Type
- JDBC URL
- JDBC Driver
- JDBC Logging
- JDBC User
- JDBC Password

See *Persistence Properties* on page 49 for details of these properties.

The default persistence database is hsqldb, which is installed with the OpenFusion CORBA Services distribution and will run with no additional configuration.

Configuring a JDBC Data Source

JDBC stands for Java Database Connectivity and is a Java implementation of the Open Database Connectivity standard (ODBC). JDBC specifies a standard interface to allow Java applications to access a relational database. All JDBC drivers support this interface, thus allowing applications to be written against the interface and isolating the developer from the different database vendors' APIs.

The JDBC data source must be pre-configured prior to running a Service. The process of configuring a JDBC database source for the databases supported by the OpenFusion CORBA Services is described below.

Your Database Administrator should configure the JDBC data source when using Oracle, Sybase, Informix or SQL Server. The hsqldb database runs locally and is installed with the OpenFusion CORBA Services distribution. No additional configuration should be required.

A set of SQL scripts that will generate all the necessary tables and stored procedures needed by the Services, within the database, have been provided. The scripts create all the necessary indexes for the tables, thus making searching through the database faster.

When an OpenFusion Service starts, it will check for the existence of the required tables (the common tables and the tables specific to that Service) in the directory indicated by the *JDBC URL* property. If the tables do not exist, OpenFusion will attempt to create them.

The scripts involved in creating the required tables and indexes are:

Table 12 SQL Scripts

Script	Description
CreateCommonTables.sql	Creates common shared tables (required by all services).
DropCommonTables.sql	Drops the common shared tables.
CreateJNDITables.sql	Creates tables for the Naming Service.
DropJNDITables.sql	Drops the tables created for the Naming Service.
CreateTraderTables.sql	Creates tables for the Trading Service.
DropTraderTables.sql	Drops the tables created for the Trading Service.
CreateNotificationTables.sql	Creates tables for the Notification Service.
DropNotificationTables.sql	Drops the tables created for the Notification Service.
CreateLogTables.sql	Creates tables for the Log Service.
DropLogTables.sql	Drops the tables created for the Log Service.

These scripts can be found in the `/admin/database/` sub-directory of the OpenFusion installation. Scripts are provided for each supported database.

The common tables must always be created. If the tables for each Service are held in Service-specific directories, a separate set of common tables must exist in each directory. If all the tables are held in a single location, only one set of common tables will be required in that location.

All the `nameof` created tables are prefixed with **`OF`**_ and all stored procedures with **`of`**_.

Oracle

The various OpenFusion tables and stored procedures can be added to an Oracle database using the `sqlplus` application, as follows:

```
%  sqlplus <USER>/<PASSWORD> < <FILE.SQL>
```

Alternatively, the generic OpenFusion JDBC `Loader` class can be used, as follows:

```
>  run com.prismt.jdbc.Loader -dt oracle -db <URL>
   -dr oracle.jdbc.driver.OracleDriver -u <USER>
   -p <PASSWORD> -s <SCRIPT> [-v]
```

Where:

`<URL>` is the database URL pointing to the directory created in Step 1.
`<USER>` is the database owner.
`<PASSWORD>` is the owner's database password.
`<SCRIPT>` is the SQL script from *Table 12* on page 108.
`-v` is an optional switch which will cause Loader to produce verbose output (listing each individual command from the script file before it executes it).

The Oracle user must have the rights to create tables and procedures in the database in order to run the SQL scripts successfully using either of the above methods.

Oracle Thin Drivers

The *type 4 thin* (all Java) Oracle JDBC drivers are supplied as part of the OpenFusion distribution. The Oracle JDBC drivers are from the Oracle 8.1.7 distribution and are backwardly compatible to Oracle version 7.3.4. The Oracle JDK 1.2 JDBC driver is supplied in the `lib` directory of the distribution in the zip file `classes12.zip`.

Oracle OCI Drivers

To use OCI Drivers with OpenFusion Services, the *oci7* or *oci8* drivers must be obtained from Oracle. These can be downloaded from *http://technet.oracle.com* (these drivers are not included as part of the OpenFusion distribution). Complete the following steps to configure the system to use these drivers.

Step 1: Install `classes12.zip` into the directory:

`<INSTALL>/lib`

where `<INSTALL>` is the OpenFusion installation directory.

Step 2: Install the appropriate driver file (`liboci73jdbc.so` or `libocijdbc8.so`, depending on the version of Oracle being used) into the directory:

`<INSTALL>/lib`

where `<INSTALL>` is the OpenFusion installation directory.

Step 3: Edit the file `<INSTALL>/bin/.javaenv` (where `<INSTALL>` is the OpenFusion installation directory) to remove this line:

```
unset LD_LIBRARY_PATH
```

and add the following two lines:

```
LD_LIBRARY_PATH=/lib
export LD_LIBRARY_PATH
```

Step 4: In the Administration Manager, set the *JDBC URL* property to:

- For Oracle 7:

  `jdbc:oracle:oci7:@`

- For Oracle 8:

  `jdbc:oracle:oci8:@`

Sybase

The various OpenFusion tables and stored procedures can be added to a Sybase database using a generic OpenFusion loader class. This can be done using:

```
% run com.prismt.jdbc.Loader -dt sybase -db <URL>
  -dr com.sybase.jdbc2.jdbc.SybDriver -u <USER>
  -p <PASSWORD> -s <SCRIPT> [-v]
```

Where:

`<URL>` is the database URL

<USER> is the database owner.
<PASSWORD> is the owner's database password.
<SCRIPT> is the SQL script from *Table 12* on page 108.
-v is an optional switch which will cause Loader to produce verbose output (listing each individual command from the script file before it executes it).

The Sybase user must have the rights to create tables and procedures in the database in order to run the SQL scripts successfully using the above method.

The SQL scripts assume that the tables and procedures are being added to the user's default database. The `use database` command should be added to the start of the scripts when this is not the case.

Informix

The various OpenFusion tables and stored procedures can be added to an Informix database using the `dbaccess` application that is provided with Informix, as follows:

```
%  dbaccess <DATABASE> <SCRIPT>
```

Where:
<DATABASE> is the name of the Informix database.
<SCRIPT> is the SQL script from *Table 12* on page 108.

Alternatively, the generic OpenFusion JDBC `Loader` class can be used, as follows:

```
>  run com.prismt.jdbc.Loader -dt informix -db <URL>
   -dr com.informix.jdbc.IfxDriver -u <USER>
   -p <PASSWORD> -s <SCRIPT> [-v]
```

Where:
<URL> is the database URL
<USER> is the database owner.
<PASSWORD> is the owner's database password.
<SCRIPT> is the SQL script from *Table 12* on page 108.
-v is an optional switch which will cause Loader to produce verbose output (listing each individual command from the script file before it executes it).

The Informix user must have the rights to create tables and procedures in the database in order to run the SQL scripts successfully using either of the above methods.

The appropriate access rights must also be granted in order for the OpenFusion services to access the database. See the Informix documentation for details of how to do this.

SQL Server

The various OpenFusion tables and stored procedures can be added to an SQL Server database using a generic OpenFusion JDBC `Loader` class. Use the following command to run this using the JDBC/OBDC driver supplied as part of the Java Runtime Environment on NT:

```
>  run com.prismt.jdbc.Loader -dt sqlserver -db <URL>
   -dr com.microsoft.jdbc.sqlserver.SQLServerDriver
   -u <USER> -p <PASSWORD> -s <SCRIPT> [-v]
```

Where:

`<URL>` is the database URL
`<USER>` is the database owner.
`<PASSWORD>` is the owner's database password.
`<SCRIPT>` is the SQL script from *Table 12* on page 108.
`-v` is an optional switch which will cause Loader to produce verbose output (listing each individual command from the script file before it executes it).

hsqldb

The hsqldb database is installed with the OpenFusion CORBA Services and configured automatically to run in standalone mode. Additional configuration is required to use hsqldb in client/server mode. To configure a new instance of hsqldb, the following steps should be used.

Create an hsqldb Instance

Step 1: Create a directory to contain the database.

Step 2: Use the following command to add OpenFusion tables to the database:

```
%  run com.prismt.jdbc.Loader -dt hsqldb -db <URL>
   -dr org.hsqldb.jdbcDriver -u <USER> -p <PASSWORD>
   -s <SCRIPT> [-v]
```

Where:

`<URL>` is the database URL pointing to the directory created in Step 1.
`<USER>` is the database owner.
`<PASSWORD>` is the owner's database password.
`<SCRIPT>` is the SQL script from *Table 12* on page 108.
`-v` is an optional switch which will cause Loader to produce verbose output (listing each individual command from the script file before it executes it).

Note that OpenFusion's default behaviour is to create the tables required by a Service in the `data` sub-directory of the Service's directory.

Configure OpenFusion Services to Run with hsqldb Persistence

Set the following properties in the Administration Manager. See Section 3.2, *Persistence Properties* on page 49, for full details of these properties.

- In order to establish a connection to hsqldb, set the *JDBC URL* property to the value supplied to the `-db` switch in Step 2, above.
- To use the default administrator account, set the *JDBC User* property to `sa` and the *JDBC Password* property to blank.

Alternatively, a new hsqldb user can be created. This user must have admin privileges in order to close down the database automatically and must be granted appropriate access permissions for the database tables.

hsqldb in Client/Server Mode

The default version of hsqldb used when a Service started is in standalone mode. This means that only one application can access the database at a time. To access the same database simultaneously from multiple JVMs, the Client/Server version of hsqldb must be used. This is highly recommended when using hsqldb in a production environment.

A Java Object for the configuration and management of hsqldb instances, *HSQLDBObject*, is included with the OpenFusion product distribution. To use the Java Object, add a Service to the OpenFusion Object Hierarchy (calling it, for example, HSQLDBService) and add the Java Object to it, as described in *Extending the Object Hierarchy* on page 19. When the Service is started, the hsqldb database will be started.

The database must have its tables loaded using the `com.prismt.jdbc.Loader` class, as described in step 2 of *Create an hsqldb Instance* on page 112.

Use the Administration Manager to configure the properties of the HSQLDBObject. The properties are described below.

Name

The name and full path of the database. This identifies where data files will be stored. The default location is:

`<INSTALL>/domains/<DOMAIN>/<NODE>/<SERVICE>/data/HSQLDBServer`

Where:

`<INSTALL>` is the OpenFusion installation directory.

<DOMAIN> is the name of the domain in the Object Hierarchy.
<NODE> is the name of the Node in the Object Hierarchy.
<SERVICE> is the name of the Service that has been created to hold the HSQLDBObject.

Property Name	Name
Property Type	FIXED
Data Type	FILE
Accessibility	READ/WRITE
Mandatory	YES

Timeout

When the Service containing the HSQLDBObject is started, it polls hsqldb to determine whether it can establish a connection. When the Service is stopped, it polls hsqldb to see if the shutdown statement has completed execution. The *Timeout* property is used by the Administration Manager to determine how long (in seconds) the Service will spend polling the database.

The default value is 30 seconds.

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

Trace

This property toggles the JDBC Trace on and off. If the property is checked (true), The JDBC Trace is switched on. The JDBC Trace is off by default.

Property Name	Trace
Property Type	STATIC
Data Type	BOOLEAN
Accessibility	READ/WRITE
Mandatory	NO

No System Exit

If this property is checked (`true`), it directs hsqldb to avoid `System.exit()` calls when the shutdown command is issued to the database. This is of particular importance when running an hsqldb server inside another application. The property is mainly used to ensure interoperability with application servers.

Property Name	`NoSystemExit`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Silent

If this property is checked (`true`), it allows the database to display all queries.

Property Name	`Silent`
Property Type	`FIXED`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`NO`

Port

The port that the hsqldb database listens on. If more than one instance of hsqldb is running on a machine, each must be set to use a different port.

For each OpenFusion Service that will use the hsqldb instance, set the *JBDC URL* property to include the specified port (see *JDBC URL* on page 52).

The default port is `9001`. If this default is used, it does not need to be added to the JDBC URL.

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

Restoring Data

If the **Restore** command is used to restore a Service to its default state (see *Restoring Services and Singletons* on page 24), all hsqldb tables and data for the Service being restored will be deleted.

This will only happen if the default *JDBC URL* is used. If the tables are in an alternative location (not a subdirectory of the Service being restored), the tables will not be deleted. The procedures given above can be used to configure an instance of hsqldb in a different location.

7 Command Line Tools

All command line tools are found in the `<install_dir>/bin` directory, where `<install_dir>` is the OpenFusion installation directory.

7.1 IOR Decoder

OpenFusion provides a tool that will decode an Interoperable Object Reference (IOR) and display its component parts or alternatively will generate a `corbaloc:iiop` URL that can be used in place of an IOR for obtaining a reference to a CORBA object.

The IOR Decoder is started with the following command:

```
% iorDecoder [-corbaloc] (-ior <IOR> | -file <file>)
  [-h | -? | -help]
```

The parameters are defined as follows:

-corbaloc	This option specifies that the tool should generate a `corbaloc:iiop` URL for the IOR.
-ior <IOR>	`<IOR>` is the IOR to decode.
-file <file>	`<file>` is a file containing the IOR to decode.
-h -? -help	These options are equivalent and print usage information for the tool.

If the IOR Decoder is invoked with the `-corbaloc` option then a `corbaloc:iiop` URL is generated for the IOR. Otherwise, the IOR is decoded and its type, port, host and stringified object key are displayed.

7.2 Administration Manager Tool

The XML configuration files in an OpenFusion installation can be fully populated from within the Administration Manager GUI (as described in *Completing the XML File Installation* on page 154). As an alternative to running the Administration Manager GUI, OpenFusion provides a command line Administration Manager tool, *`adminMgrTool`*, that will fully populate the XML configuration files.

The Administration Manager tool is started with the following command:

```
% adminMgrTool <parameter>
```

The following parameters can be used with the tool to provide different functionality:

`-p`

Causes the tool to fully populate the XML configuration files.

`-r <restore_name>`

Restores a named object in the Object Hierarchy. `<restore_name>` is a hierarchical name (using dots to separate the name components) which identifies the object to be restored. For example, to restore the *NotificationService* from the *localhost* node of the *OpenFusion* domain:

```
adminMgrTool -r OpenFusion.localhost.NotificationService
```

`-d <node_name>`

Deletes a named object in the Object Hierarchy. `<node_name>` is a hierarchical name (using dots to separate the name components) which identifies the object to be deleted. For example, to delete the *NotificationService* from the *localhost* node of the *OpenFusion* domain:

```
adminMgrTool -d OpenFusion.localhost.NotificationService
```

`-aDom <domain_name>`

Adds a Domain to the Object Hierarchy. For example:

```
adminMgrTool -aDom OpenFusion
```

`-aNode <domain_name> <node_name>`

Adds a node to a domain, where `<domain_name>` is the name of the domain that the node will be added to. For example, to add the *localhost* node to the *Openfusion* domain:

```
adminMgrTool -aNode Openfusion localhost
```

`-aServ <node_name> <service_name>`

Adds a Service to a node in the Object Hierarchy. `<node_name>` is a hierarchical name (using dots to separate the name components) which identifies the node that the Service will be added to. For example, to add *NamingService* to the *localhost* node of the *OpenFusion* domain:

```
adminMgrTool -aServ OpenFusion.localhost NameService
```

`-aSing <service_name> <singleton_name>`

Adds a Singleton to a Service. `<singleton_name>` is a hierarchical name (using dots to separate the name components) which identifies the Service that the Singleton will be added to. For example, to add the *NameSingleton* to the *NameService*:

```
adminMgrTool -aSing OpenFusion.localhost.NameService NameSingleton
```

`-aJO <service_name> <java_object_name>`

Adds a Java Object to a Service. `<service_name>` is a hierarchical name (using dots to separate the name components) which identifies the Service that the object will be added to. For example, to add the *SNMPAgentObject* to the *NameService*:

```
adminMgrTool -aJO OpenFusion,localhost.NameService SNMPAgentObject
```

`-ap <object_name> <property_name> <new_property_value>`

Changes a property value for an object in the Object Hierarchy. `<object_name>` is a hierarchical name (using dots to separate the name components) which identifies the object. For example, to change the value of the Naming Service *log4j.rootLogger* property to "*Debug*":

```
adminMgrTool -ap OpenFusion.localhost.NameService log4j.rootLogger Debug
```

`-h`
`-?`
`-help`

These options are equivalent and print usage information for the tool.

A single command line can only include one parameter. So to add a domain, a node, and two Services to the Object Hierarchy, for example, the Administration Manager tool must be invoked four times with a different object added in each invocation.

The Administration Manager tool must be invoked with the `-p` option before any Services can be started from the command line using the `server -start` script (as described in *Starting Servers from the Command Line* on page 8).

7.3 Configuration Generator

OpenFusion provides a tool that will parse an XML configuration hierarchy and output references to configured objects in a form suitable either for inclusion in an ORB configuration file or as arguments that can be passed to an ORB at initialization.

The Configuration Generator is started with the following command:

```
% configGen -orb (orbix2000 | orbacus | jacorb |
  portable | visibroker) -domain_xml <XML>
  [-out_file <file>] [-h | -? | -help]
```

The parameters are defined as follows:

-orb	This option specifies the type of ORB for which output is to be generated.
-domain_xml <XML>	*<XML>* is the XML configuraton file for the domain to be parsed by the tool.
-out_file <file>	*<file>* is the file to contain the generated output.
-h -? -help	These options are equivalent and print usage information for the tool.

If the Configuration Generator is invoked without the `-out_file` parameter then the output is written to the console. If the `portable` type is specified with the `-orb` option then the output is in the format `-ORBInitRef <ObjectID>=<ObjectURL>` described in the CORBA specification.

Security Service

8 Description

8.1 Overview

The OpenFusion Security Service provides the ability to apply access control to CORBA Services and Java Objects.

Access control is based upon clients' identities being verified by plug-in authentication modules. The OpenFusion Security Service is independent of the authentication technology in use, allowing the flexibility of a range of authentication systems from simple username/password entry to voice or fingerprint verification.

The Security Service can be used to control access to individual Object instances or even specific methods. This level of granularity provides an extremely flexible and configurable security model. For example, individual Notification Service message queues can be secured so that only clients which provide a valid username and password combination can access a particular queue. This is of great value in messaging applications where sensitive data is involved; in the banking sector, for example.

8.2 Concepts and Architecture

The OpenFusion Security Service supports the use of *Pluggable Authentication Modules* (PAM) which conform to Sun's *Java Authentication and Authorisation Service* (JAAS) API specification. Pluggable authentication modules allow the Security Service to remain independent of the underlying authentication technologies. Typical authentication modules may prompt for and verify a username and password, for example.

Securable Objects

The Security Service allows security to be applied to CORBA Objects and Java Objects. Any Java Object which implements the `Identifiable` interface, and any CORBA Object, is a securable object.

For a CORBA Object, only operations interfaces and their methods are securable.

For a Java Object, only the interfaces from the class implementing the `Identifiable` interface (its interfaces and the interfaces of its superclasses) are securable.

Running the Security Service does not automatically secure all objects and Services. Nothing is secured in a default OpenFusion installation. It is necessary to identify which resources will be secured and to establish access lists for each resource.

Authentication

The purpose of the authentication process is to associate a *Principal* with a *Subject*.

A Subject is any client or user of a Service or resource. A Principal is a name that represents an identity attribute. For example, a Principal could be the user ID of the client user, the user's role within an organisation, or the name of a group of users.

Authentication can be performed on either the client or the server side of any transaction, or on both sides of the transaction. If successful, the Subject will be associated with zero or more Principals. One Subject may have several Principals, representing the names by which it is identified to different Services.

Once a Subject has been authenticated, the Security Service propagates the Subject's identity (Principal) to all subsequent actions attempted by that Subject.

ACLs

An *ACL* (Access Control List) is a list of Principals which will be allowed access to a particular resource. When a subject attempts to access a resource, the Security Service checks each of the Subject's Principals against the resource's ACLs to determine whether access should be allowed.

Each given resource can have two different ACLs in effect:

- The ACL specific to this particular method on this particular type of object. If an ACL is not defined at the method level, the ACL for the type of object is used, if defined.
- The ACL specific to this particular method on this particular object instance. If an ACL is not defined at the method level, the ACL for the object instance is used, if defined.

If either or both ACLs are defined for a resource, Principals which are listed in either or both ACLs will be allowed access the resource. Principals which do not appear on either list will be denied access.

If no ACLs are defined, the resource will be considered unrestricted and access will be permitted to any Subject.

If both ACLs are defined and both are empty lists, or if only one ACL is defined and it is an empty list, all access to the resource is prohibited.

If one ACL grants a Principal access to a resource, that access cannot be revoked by the resource's other ACL. So if a Principal is granted access to a particular type of object, those access rights extend to all instances of that object even where the instance has a blank ACL (which would normally deny access to all Principals).

ACLs are created and maintained through the Security Administration Manager (see Section 11, *Security Administration Manager* on page 147) and are held as XML files. It is possible, but not recommended, to edit the XML files using some method other than the Security Administration Manager.

The UML model in *Figure 14* shows the relationship between Principals and ACLs.

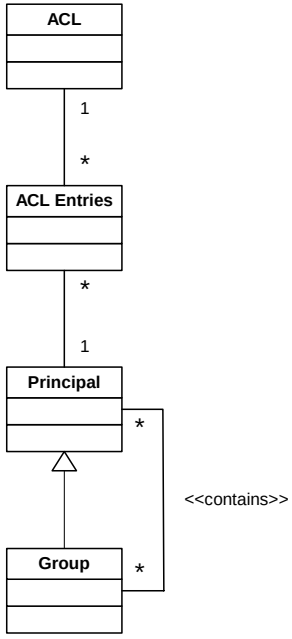

Figure 14 ACL UML Model

Groups

Groups represent collections of Principals. Groups cannot be assigned to a Subject, but they can be used in ACLs in order to simplify the construction of the ACL.

It is possible to construct Groups of Groups, recursively, with the resulting potential for cyclic references. Cyclic references will be rejected from the ACL with unpredictable results, and so should be avoided when Groups are constructed.

When the Security Service evaluates an ACL to determine if a Principal should be granted access to a resource, all Groups in the ACL will be examined to see if the Principal is a member. If the Principal is a member of a Group in the ACL, it will be granted access.

ACLs are examined to determine Group membership each time an attempt is made to access a resource.

Group details are defined in an XML file, as described in Section 9.2, *Creating ACL Groups* on page 129.

Mapping Principals

The OpenFusion Security Service `LoginModule` includes a mechanism for mapping Principals to a Subject at the point of authentication. The `LoginModule` examines the Principals which other login modules have associated with a Subject and determines whether mappings exist between those Principals and any other Principals. If mappings exist, the mapped Principals are also associated with the Subject.

Principal mappings are defined in an XML file, as described in Section 9.3, *Creating Principal Mappings* on page 130.

Example:

A user logs on to the system, providing his user name `joe` and password `secret` to the login module. These are authenticated and found to be correct. The login module assigns a Principal, `joe`, to represent his authenticated identity.

In the Principal Mappings file, a mapping exists between the Principal `joe` and the two Principals `administrator` and `boss`. These Principals are added to the Subject `joe`. Furthermore, the Principal `administrator` is also mapped to the Principal `guest`. This mapping is resolved, and `guest` is also assigned to `joe`.

After login, `joe` will be able to access any resource whose ACL includes any of the Principals `joe`, `administrator`, `boss`, or `guest`, or any Groups which had any of these Principals as direct or indirect members.

LoginModule

The supplied `LoginModule` includes a simple Generic Security Service Username and Password (GSSUP) authentication mechanism. It holds user names against passwords in plain text in an XML file and will successfully authenticate any user which supplies a correct username and password pair.

Other Pluggable Authentication Modules can be used, as long as they conform to the JAAS specification. A discussion of Pluggable Authentication Modules is outside the scope of this document.

When different authentication modules are used, the last stage of authentication is always performed by the Security Service `LoginModule`, which performs Principal mapping as described in *Mapping Principals* on page 126.

9 Using Specific Features

This section describes the main procedures for securing OpenFusion Services. The procedures are the same for any OpenFusion Service.

9.1 Securing an Interface or Method

This procedure describes how an object can be secured using the Administration Manager. Each object and method for which security access has been set is stored persistently in an XML file. It is possible, but not recommended, to add entries directly to the XML file without using the Security Administration Manager.

See Section 11, *Security Administration Manager*, on page 147, for a further details of using the Security Administration Manager.

Step 1: Ensure that the Service is **Stopped**. Note that it is possible to configure a running service, but see *Step 10:* on page 128.

Step 2: Select the Service in the Administration Manager's Object Hierarchy and select the *SECURITY* tab in the properties panel.

Step 3: Enable security for the Service by clicking the *Security Enabled* check box.

Step 4: Enter the location of the following security configuration files:

- *XML Group Persistence*
- *XML Principal Persistence*
- *JAAS Configuration File*
- *XML ACL Persistence*
- *Security Credentials File*
- *Security Configuration File*

If these properties are unavailable (grey), it means that the *Security Enabled* property has not been set.

Default locations are supplied for all of these files, and can be accepted without change if desired.

Step 5: **Start** the Service and then start the Service Manager.

Step 6: Right-click on an object in the Service Manager hierarchy and select **Security Administration Manager** from the popup menu. This will start the Security Administration Manager with the security object hierarchy populated with the interfaces and methods of the selected object.

The Security Administration Manager is fully described in Section 11, *Security Administration Manager*, on page 147.

Step 7: Select an interface or method in the security object hierarchy. This is the operation which we will secure so that only authorised clients can access it.

The security Principals that are associated with the operation will be shown in the *Access Entry Details* list. At this point, the list should be blank.

Step 8: Type a Principal name in the *Enter principal to be added* box and click the **Add** button. This will add the principal to the *Access Entry Details* list.

Only clients which supply valid credentials for the listed Principal will be able to access the operation. Note that by default (before any security Principals are added), any client could have accessed the operation. The act of adding a Principal effectively denies access to everybody except that Principal.

Repeat the previous step with additional Principals, if required.

Step 9: Click the **Save Changes to Security Access Entries** tool bar button. This action saves the security configuration to persistent storage.

Step 10: If the Service is running, go back to the *SECURITY* tab of the Administration Manager and click the *Reload Security Configuration* signal button to force the Service to reload the security configuration from persistent storage (XML files).

A Service reads its security configuration on start-up. Any changes made in the Security Administration Manager while the service is running will not be automatically implemented. This signal must be used to implement the changes in the running Service.

This step is only required if the Service is running when the properties are changed. If the Service is stopped first, there is no need to force a configuration reload.

The signal button reloads the configuration from the XML files, *not* from the current state of the Security Administration Manager. Any changes made in the Security Administration Manager *must* be explicitly saved (using the tool bar button) before the signal button is used to reload the configuration. Otherwise, the changes will be lost.

Excluding Methods from the Security Manager

In some circumstances, it is only possible to secure an object at the object level, not at the method level. This is because the methods are never called on an instantiated object and therefore can never be interecpted. (This only alllies to instantiated objects, never to interfaces.) In this situation, it is useful to exclude the object's

methods from the object hierarchy of the Security Administration Manager. This avoids the mistaken belief that an object has been made secure because its methods appear secure, when in fact the security should have been set at the object level.

To exclude an object from the hierarchy, add an entry for the object to the `SecurityObjectLevel.xml` file, using a suitable XML editor or plain text editor. This file is located in the `<install>/xml/security` directory (where `<install>` is the OpenFusion CORBA Services installation path).

i A working knowledge of XML is required to edit the `SecurityObjectLevel` file. An example of this file is given below.

Example SecurityObjectLevel File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Singleton SYSTEM "file:///D:/openfusion/xml/schema/SecurityObjectLevel.dtd">
<SecurityObjectLevel>
  <ClassName>com.prismt.j2ee.jms.QueueImpl</ClassName>
  <ClassName>com.prismt.j2ee.jms.TopicImpl</ClassName>
</SecurityObjectLevel>
```

9.2 Creating ACL Groups

Groups represent collections of Principals. Groups can be placed into an ACL in order to simplify the construction of ACLs. ACL Group details are defined in an XML file.

i A working knowledge of XML is required to create and maintain ACL Groups. An example of the XML Group Persistence File is given below.

Step 1: Select the Service in the Administration Manager's Object Hierarchy and select the *SECURITY* tab in the properties panel.

Step 2: Enable security for the Service by clicking the *Security Enabled* check box.

Step 3: Enter the location of the *XML Group Persistence* file. The default location can be used if required.

Services can share a single Group persistence file, or a different file can be specified for each Service. The default is for all Services to store their Group persistence files in a common location.

Step 4: Locate the XML Group Persistence File in the directory identified in Step 3.

Step 5: Use a suitable XML editor or plain text editor to create or modify the XML Group Persistence file. The file must conform to the following schema:

`<install>/xml/schema/of-security-groups.xsd`

where <install> is the OpenFusion CORBA Services installation path.

Step 6: Click the *Reload Security Configuration* signal button on the *SECURITY* tab to force the underlying Service to implement the changed security configuration.

Example XML Group Persistence File:

```
<?xml version="1.0" encoding="UTF-8"?>
<securityGroups xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="d:openfusion/xml/schema/of-security-groups.xsd">
    <group>
      <groupName>All Users</groupName>
      <memberPrincipal>Administrator</memberPrincipal>
      <memberPrincipal>Default User</memberPrincipal>
      <memberPrincipal>Guest</memberPrincipal>
    </group>
</securityGroups>
```

9.3 Creating Principal Mappings

Mappings between Principals can be used to assign additional Principals to a subject at the point of authentication. Principal mappings are defined in an XML file.

i A working knowledge of XML is required to create and maintain Principal mappings. An example of the XML Principal Persistence File is given below.

Step 1: Select the Service in the Administration Manager's Object Hierarchy and select the *SECURITY* tab in the properties panel.

Step 2: Enable security for the Service by clicking the *Security Enabled* check box.

Step 3: Enter the location of the *XML Principal Persistence* file. The default location can be used if required.

Services can share a single Principal mapping file, or a different file can be specified for each Service. The default is for all Services to store their Principal mapping files in a common location.

Step 4: Locate the XML Principal mapping file in the directory identified in Step 3.

Step 5: Use a suitable XML editor or plain text editor to create or modify the XML Principal mapping file. The file must conform to the following schema:

```
<install>/xml/schema/of-security-principal-map.xsd
```

where <install> is the OpenFusion CORBA Services installation path.

Step 6: Click the *Reload Security Configuration* signal button on the *SECURITY* tab to force the underlying Service to implement the changed security configuration.

Example XML Principal Persistence File:

```
<?xml version="1.0" encoding="UTF-8"?>
<principalMappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="openfusion/xml/schema/of-security-principal-map.xsd">
   <principal>
      <principalName>root</principalName>
      <impliesPrincipal>admin</impliesPrincipal>
   </principal>
   <principal>
      <principalName>user</principalName>
      <impliesPrincipal>canRead</impliesPrincipal>
      <impliesPrincipal>canPrint</impliesPrincipal>
      <impliesPrincipal>canExecute</impliesPrincipal>
   </principal>
   <principal>
      <principalName>admin</principalName>
      <impliesPrincipal>user</impliesPrincipal>
      <impliesPrincipal>canWrite</impliesPrincipal>
   </principal>
   <principal>
      <principalName>canRead</principalName>
   </principal>
   <principal>
      <principalName>canPrint</principalName>
   </principal>
   <principal>
      <principalName>canExecute</principalName>
   </principal>
   <principal>
      <principalName>canWrite</principalName>
   </principal>
</principalMappings>
```

9.4 Supplying Authorised Credentials

Authorised credentials must be supplied by any client code which attempts to use a secured operation. Authentication is carried out by a Pluggable Authentication Module (PAM). The login module supplied with OpenFusion provides a Generic Security Service Username and Password (GSSUP) authentication mechanism. The supplied module is:

`com.prismt.openfusion.security.login.LoginModule`

The default LoginModule compares supplied credentials with the list of credentials held in the Security Credentials file to determine validity. This is a plain-text XML file stored in a location identified by the *Security Credentials File* property in the Administration Manager. The location can also be set in the `gssupUsers` element of the Security Configuration file or in the system property `security.UserDataFile`. See *Security Configuration File Properties* on page 142 for details.

The default location of the Security Credentials file is:

`<install>/Security/etc/security/userdata.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

i A working knowledge of XML or familiarity with an XML editor is required to create and maintain the Security Credentials file. An example of this file is given below.

Example Security Credentials File:

```
<Users>
   <User>
      <UserName>adminuser</UserName>
      <Password>adminPass</Password>
   </User>
   <User>
      <UserName>guest</UserName>
      <Password>guestPass</Password>
   </User>
</Users>
```

10 Security Configuration

Security must be configured separately for a Service and for the clients of that Service. Service configuration is performed through the Administration Manager. Client configuration comes from a combination of system properties and details stored in an XML file.

10.1 Configuring a Secure OpenFusion Service

Security properties for a service are configured through the Administration Manager, as follows.

Step 1: Select a Service in the Administration Manager's *Object Hierarchy.*

Step 2: Select the *SECURITY* tab.

Step 3: Set the following properties:

- *Security Enabled*
- *XML Group Persistence*
- *XML Principal Persistence*
- *JAAS Configuration File*
- *XML ACL Persistence*
- *Security Credentials File*
- *Security Configuration File*

These properties are fully described in *Security Administration Manager Properties*, below.

i These properties should be configured separately for each OpenFusion Service.

Security Administration Manager Properties

Services can share common persistence files, or different file locations can be specified for each Service. The default is for all Services to store their persistence files in a common location, which means that by default the above properties are configured identically for each Service.

Security Enabled

If this property is checked, security is enabled for the Service. If security is not enabled, the remaining properties on this tab are unavailable.

Property Name	`security.Enabled`
Property Type	`STATIC`
Data Type	`BOOLEAN`
Accessibility	`READ/WRITE`
Mandatory	`YES`

XML Group Persistence

The name and location of the XML group persistence file, given as either a *file* or *http* URL. This defaults to:

`file:<install>/etc/security/grouppersistence.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

This file is described in 9.2, *Creating ACL Groups* on page 129.

Property Name	`security.XMLGroupPersistenceFile`
Property Type	`STATIC`
Data Type	`URL`
Accessibility	`READ/WRITE`
Mandatory	`YES`

XML Principal Persistence

The name and location of the XML principal persistence file, given as either a *file* or *http* URL. This defaults to:

`file:<install>/etc/security/principalpersistence.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

This file is described in described in 9.3, *Creating Principal Mappings* on page 130

Property Name	`security.XMLPrincipalPersistenceFile`
Property Type	`STATIC`
Data Type	`URL`
Accessibility	`READ/WRITE`
Mandatory	`YES`

JAAS Configuration File

The name and location of the Java Authentication and Authorisation Service (JAAS) configuration file, given as either a *file* or *http* URL. This defaults to:

`file:<install>/etc/security/jaas.config`

where `<install>` is the OpenFusion CORBA Services installation path.

Property Name	`java.security.auth.login.config`
Property Type	`STATIC`
Data Type	`URL`
Accessibility	`READ/WRITE`
Mandatory	`YES`

XML ACL Persistence

The name and location of the XML ACL persistence file, given as either a *file* or *http* URL. This defaults to:

`file:<install>/etc/security/accessentry.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

Property Name	`security.XMLACLPersistenceFile`
Property Type	`STATIC`
Data Type	`URL`
Accessibility	`READ/WRITE`
Mandatory	`YES`

Security Credentials File

The name and location of the file holding user credentials, given as either a *file* or a *http* URL. This defaults to:

`file:<install>/Security/etc/security/userdata.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

This file is described in *Supplying Authorised Credentials* on page 131.

Property Name	`security.UserDataFile`
Property Type	`STATIC`

Data Type	URL
Accessibility	READ/WRITE
Mandatory	YES

Security Configuration File

The name and location of the security configuration file, given as either a *file* or a *http* URL. This defaults to:

`file:<install>/etc/security/SecurityProperties.xml`

where `<install>` is the OpenFusion CORBA Services installation path.

This file is described in *Security Configuration File Properties* on page 136.

This property will be over-ridden by the system property `security.ConfigurationFile`, if it is set.

Property Name	security.ConfigFile
Property Type	STATIC
Data Type	URL
Accessibility	READ/WRITE
Mandatory	YES

10.2 Configuring a Secure Client

A secure client is configured from properties held as elements in an XML file. The location of this file is given by the system property `security.ConfigurationFile`. If this has not been set, the location will be taken from the *Security Configation File* property set in the Administration Manager.

i A working knowledge of XML or familiarity with an XML editor is required to create the Security Configuration file.

Security Configuration File Properties

The following properties can be set in the Security Configuration file. Some of these properties can also be set (or overridden) in other ways, as noted.

securityEnabled

This property determines whether security will be enabled or disabled and can take the values `true` (enabled) or `false` (disabled). It defaults to `false` if not explicitly set.

To disable security, neither this property nor the *Security Enabled* property in the Administration Manager must be set to `true`. Either one of the two properties set to `true` is sufficient to enable security.

gssupCredential

This includes two properties: `user` and `password`, which are the GSSUP credentials that will be used for the Subject.

There are three ways that these properties can be set. In order of precedence, these are:

1. As the following system properties:

   `OF.Security.UserName`

   `OF.Security.Password`

2. Programatically, by invoking the following methods:

```
com.prismt.openfusion.security.util.Configuration.getInstance().setGSSUPUserName(name)
com.prismt.openfusion.security.util.Configuration.getInstance().setGSSUPPassword(password)
```

3. In the Security Configuration file, as in the following example fragment:

```
    <gssupCredential>
      <user>administrator</user>
      <password>my_password</password>
    </gssupCredential>
```

fileLocations

This group of properties defines the locations of up to five different files used by the Security Service:

- `principalMappings`

  This property gives the location of the Principal Mappings file (described in 9.3, *Creating Principal Mappings* on page 130). If this property is set, it will override the *XML Principal Persistence File* property set in the Administration Manager. The system property `security.XMLPrincipalPersistenceFile` can be used to override the location set by this property.

- `acls`

  This property gives the location of the ACL Persistence file. If this property is set, it will override the *XML ACL Persistence File* property set in the Administration Manager. The system property `security.XMLACLPersistenceFile` can be used to override the location set by this property.

- `groups`

  This property gives the location of the Group Persistence file (described in 9.2, *Creating ACL Groups* on page 129). If this property is set, it will override the *XML Group Persistence File* property set in the Administration Manager. The system property `security.XMLGroupPersistenceFile` can be used to override the location set by this property.

- `gssupUsers`

  This property gives the location of the file that holds user names and passwords for the default LoginModule. If this property is set, it will override the *Security Credentials File* property set in the Administration Manager. The system property `security.UserDataFile` can be used to override the location set by this property.

- `jaasLoginConfig`

  If this property is present, its value will be used to set the `java.security.auth.login.config` system property. It is used by the `com.sun.security.auth.login.ConfigFile` object, which handles runtime login configuration. For more details, consult the JAAS documentation.

jaasLoginConfigName

If this property is present, it will override the default key used to identify the configured LoginModules. The default value of this key is *`OpenFusion`*. For more details, consult the JAAS documentation.

clientSideLogin

If this property is set to `true`, LoginModules will be triggered on the client side of a call. The property defaults to `false` if not explicitly set.

serverSideLogin

If this property is set to `true`, LoginModules will be triggered if this is the server side of a call. The property defaults to `true` if not explicitly set.

Example Security Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<securityConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         NamespaceSchemaLocation="/openfusion/xml/schema/of-security-config.xsd">
    <gssupCredential>
      <user>administrator</user>
      <password>my_password</password>
    </gssupCredential>
    <fileLocations>
      <principalMappings>http://configserver/openfusion/mapping.xml</principalMappings>
      <acls>http://configserver/openfusion/acls.xml</acls>
      <groups>http://configserver/openfusion/groups.xml</groups>
      <gssupUsers>http://configserver/openfusion/usersfile.xml</gssupUsers>
      <jaasLoginConfig>http://configserver/openfusion/jaas.config</jaasLoginConfig>
    </fileLocations>
    <jaasLoginConfigName>LoginConfig_Name</jaasLoginConfigName>
    <clientSideLogin>false</clientSideLogin>
    <serverSideLogin>true</serverSideLogin>
</securityConfig>
```

11 Security Administration Manager

Use the Security Administration Manager to create and maintain ACLs for the OpenFusion Services. The Manager allows Principals to be added to individual objects, interfaces, or methods.

11.1 Starting the Security Administration Manager

The Security Administration Manager cannot be started from the command line. It must be started from within the Administration Manager, either from a Service node or from an instantiated object.

- Starting from a Service node

  To start the Security Administration Manager, right-click on a Service node in the Administration Manager's *Object Hierarchy* and select **Security Administration Manager** from the pop-up menu.

  When the Security Administration Manager is started from a Service node, the browser will not be initially populated with entries in the security object hierarchy.

- Starting from an instantiated object

  The Security Administration Manager can be started from any node in a Service Manager's object hierarchy which represents a securable object.

  To start the Security Administration Manager, right-click on the selected node and select **Security Administration Manager** from the pop-up menu.

  When the Security Administration Manager is started from an instantiated object, the security object hierarchy will be populated automatically with the object's interfaces and the methods it implements (including methods inherited from its interfaces).

Only one instance of the Security Administration Manager can be loaded in any one session.

11.2 Using the Security Administration Manager

The left-hand panel of the Security Administration Manager (the security object hierarchy) shows the securable objects that have been loaded into the Security Administration Manager. The right-hand panel shows details of the object selected in the hierarchy. See Figure 15, *The Security Administration Manager* on page 142.

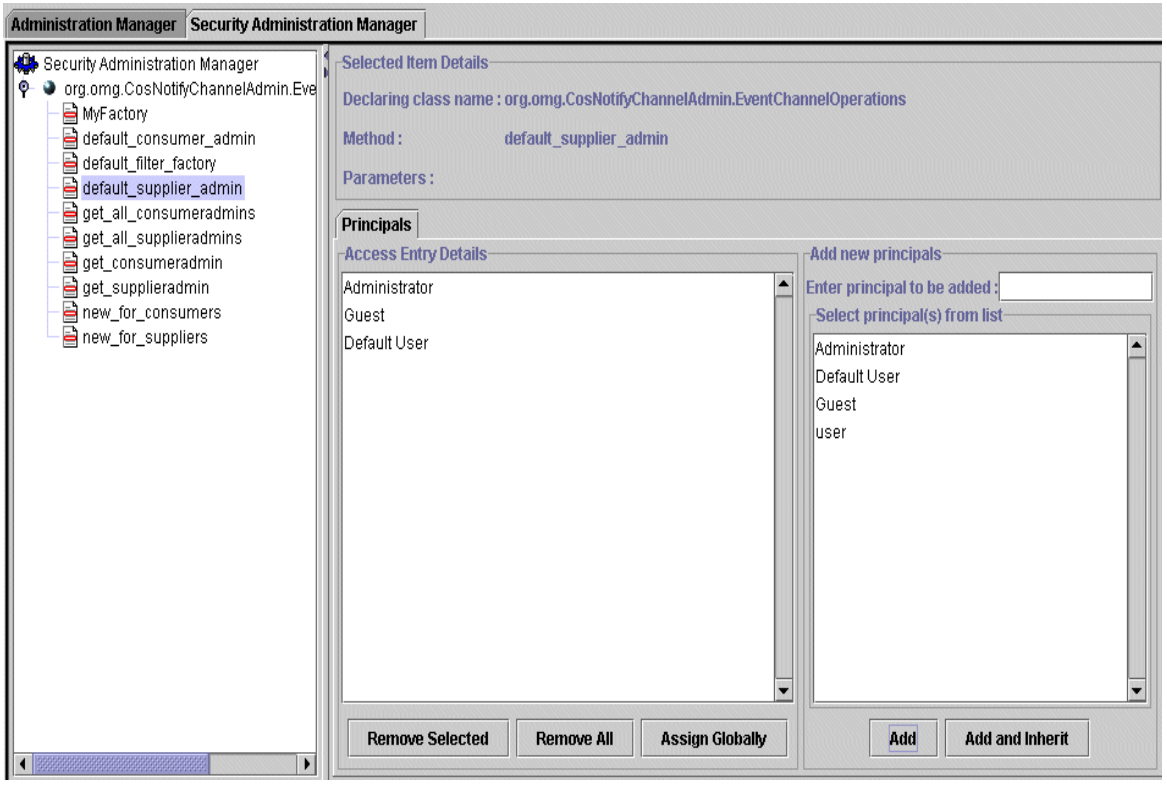

Figure 15 The Security Administration Manager

Object Hierarchy

The security object hierarchy shows the securable objects that have been loaded into the Security Administration Manager.

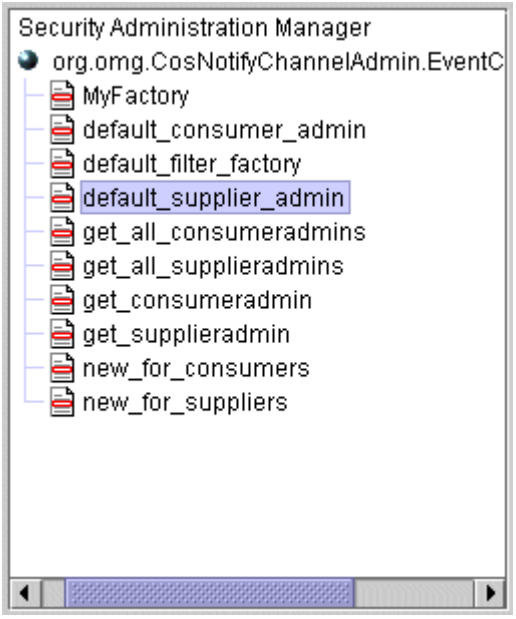

Figure 16 The Security Object Hierarchy

If the Security Administration Manager is launched from an instantiated object, the security object hierarchy will be automatically populated with objects, interfaces, and methods. If these objects are then assigned security access entries, they will be added to persistent storage.

When the Security Administration Manager is launched from a Service node, the security object hierarchy is empty. To populate it with entries from persistent storage, right-click on the *root node* and use one of the pop-up menu options, described below.

Different objects in the Security Administration Manager are identified by different icons in the security object hierarchy tree view. These icons are shown in *Table 13*.

Table 13 Security Object Icons

Icon	Node
	*Root Node* The object hierarchy root node.
	*Object* Represents a CORBA object or Java object. When this node is selected, security access information for the object is shown in the right-hand panel. When this node is expanded, all methods applicable to the object are shown. Methods inherited from any operations type class are also shown.
	*Type* Represents an object's operations type class, to allow security access controls to be set against either the object or the type.
	*Method* Represents a method, which is the lowest level at which security access controls can be set. When this node is selected, security access information for the method is shown in the right-hand panel.

Security Hierarchy Options

The following options are used to populate the security object hierarchy. These options are accessed by right-clicking on the *root node* of the security object hierarchy.

- Get First 100 Security Access Entries

  This option retrieves the first 100 entries in the XML file. For performance reasons, the number of entries displayed at any one time is limited to 100.

- Get Next 100 Security Access Entries

  This option is enabled after the **Get First 100 Security Access Entries** option has been used. This option retrieves the next 100 security access entries from the XML file. The previous 100 entries are removed from the hierarchy, so that a maximum of 100 entries are displayed at one time.

- Search

  This option allows a single interface to be loaded from persistent storage. Enter an interface name in the dialog box displayed when this option is selected. If an entry exists in persistent storage for the object, or a method or interface relating to the object, the details are retrieved and added to the security object hierarchy.

- Add New Security Access Entry

  This option allows a new security access entry to be added for an interface. Enter an interface name in the dialog box displayed when this option is selected. If an entry exists in persistent storage for the object, or a method or interface relating to the object, the details are retrieved and added to the security object hierarchy. If it does not exist, the details will be added to the security object hierarchy and a persistent storage entry will be created if Principals are added and saved.

Excluding Methods from the Object Hierarchy

In some circumstances, it is only possible to secure an object at the object level, not at the method level. In this situation, it is useful to exclude the object's methods from the object hierarchy. See *Excluding Methods from the Security Manager* on page 128 for details.

Tool Bar Buttons

The Security Administration Manager adds a new button to the tool bar. This button is shown in *Table 14*.

Table 14 Security Administration Manager Tool Bar

Button	Function
	**Save Changes to Security Access Entries** Saves security access entries to XML files for persistent storage.

Principals Panel

The *Principals* panel controls security access for the object, interface, or method selected in the security hierarchy. It consists of two sections: *Add new principals* and *Access Entry Details*.

The *Access Entry Details* list box lists all Principals who have been granted access to the class or method. The *Add new principals* list box lists all Principals that are available for adding to a class or method. This list is built dynamically as each node is selected. It is *not* a definitive list of all known Principals.

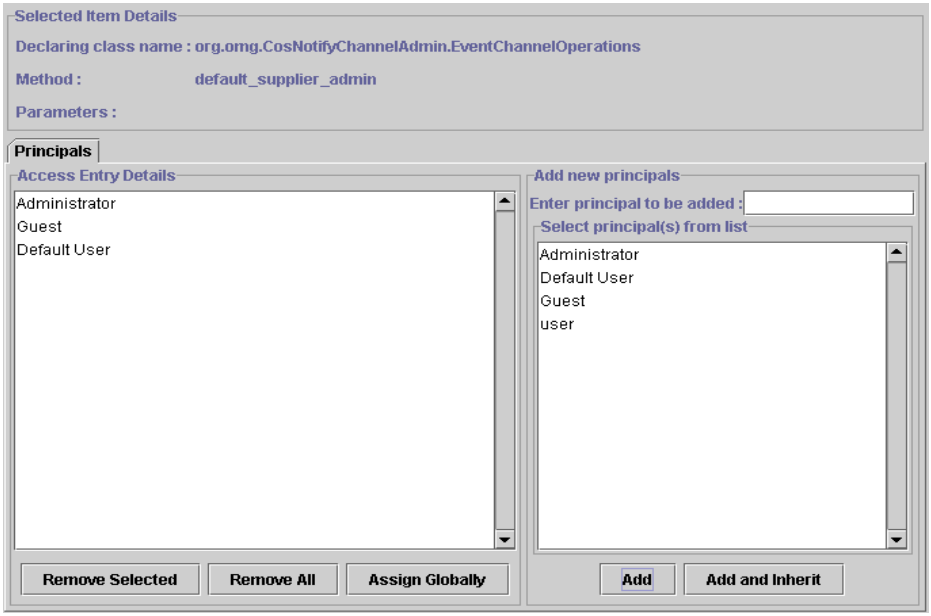

Figure 17 The Principals Panel

Operations

The following operations can be performed from the Principals panel of the Security Administration Manager:

- Add a new Principal
- Assign a Principal to a class or method
- Add and inherit
- Remove a Principal from a Class or Method
- Remove All Principals from a Class or Method
- Delete Access Entries Globally
- Assign Principals Globally

These operations are described in the following sections.

Add a New Principal

Step 1: In the security object hierarchy, select the class or method that the Principal will be added to.

Step 2: Enter the Principal name in the text box under *Add new principles*.

Step 3: Click the **Add** button. The Principal will be added to the *Access Entry Details* list.

Step 4: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

A Principal must be added to a specific class or method. It is not possible to add a Principal to the list of Principals without also assigning it to a class or method.

A new Principal will be added to persistent storage with the ACL entry for the object, interface, or method it is added to.

Assign a Principal to a Class or Method

Once a Principal has been added to one class or method, it is available to add to other classes and methods.

Step 1: Select the class or method from the security object hierarchy.

Step 2: Click the Principal name in the list of Principals. Use shift+click to select a range of Principals, ctrl+click to select a non-contiguous range.

Step 3: Click the **Add** button. The Principal will be added to the selected class or method.

Step 4: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Add and Inherit

Principals applied to an object or interface are not automatically applied to every method of that object or interface. The following procedure should be used to cause a method to inherit its parent's security Principals.

Step 1: Select a method from the security object hierarchy.

Step 2: Click the **Add and Inherit** button. (This button is not enabled until a method is selected in the security object hierarchy.)

Step 3: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Remove a Principal from a Class or Method

Step 1: Select the class or method from the security object hierarchy.

Step 2: Click the Principal name in the *Access Entry Details* list. Use shift+click to select a range of Principals, ctrl+click to select a non-contiguous range.

Step 3: Click the **Remove Selected** button.

Step 4: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Note that when a Principal has been removed from all classes and methods and the changes saved, it is no longer held in persistent storage. It remains in the Principals list until the Security Administration Manager is shut down.

Remove All Principals from a Class or Method

There are two ways in which all Principals can be removed from a class or method. The results of the two procedures are significantly different because of how empty ACLs are treated. See *ACLs* on page 124 for more details of this.

Remove Principals and deny all access

This will leave the class or method's ACL with no Principals recorded against it. This has the effect of denying all access to the class (if Principals are removed at the class level) or method (if Principals are removed at the method level).

Step 1: Select the class or method from the security object hierarchy.

Step 2: Click the **Remove All** button.

Step 3: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Remove Principals and allow free access

This will remove the class or method's ACL. This has the effect of removing all security from the class (if Principals are removed at the class level) or method (if Principals are removed at the method level) and allowing anyone access to it.

Step 1: Right-click the class or method in the security object hierarchy.

Step 2: Select **Delete Access Entry** from the pop-up menu.

Step 3: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Delete Access Entries Globally

This procedure will delete all security access entries for an object or interface, and all security access entries for methods of that object or interface.

Step 1: Right-click the object or interface in the security object hierarchy.

Step 2: Select **Global Delete** from the pop-up menu.

Note that the deleted Principals remain in the Principals list until another node in the security object hierarchy is selected.

Step 3: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Assign Principals Globally

Step 1: Assign Principals to a class or method, using the steps in either *Add a New Principal* on page 147 or *Assign a Principal to a Class or Method* on page 147.

Step 2: Click the **Assign Globally** button. Every Principal in the *Access Entry Details* list is assigned to all methods of the parent object or interface.

Step 3: Click the **Save Changes to Security Access Entries** tool bar button to commit the changes.

Implementing Security Configuration Changes

Changes to the security configuration for a Service can be performed while the Service is running or halted, but changes made while the Service is running will not be immediately implemented. There are two ways in which security changes can be passed to a running Service:

- If the Service is stopped and re-started, it will read and implement the new security configuration.
- If the *Reload Security Configuration* signal button is clicked, the Service re-reads the security configuration and implements any changes.

Interfaces

This panel is only displayed when the Security Administration Manager has been invoked from a Service node (see *Starting from a Service node* on page 141).

The *Interfaces* panel lists the interfaces for the class selected in the security object hierarchy. These interfaces may have their own security access settings, and so can be loaded into the security object hierarchy.

To load an interface class into the hierarchy:

Step 1: Select an object in the security hierarchy. The *Interfaces* panel will not be available if a method is selected.

Step 2: Select the *Interfaces* tab in the right-hand panel of the Security Administration Manager.

Step 3: Select the required interface from the list in the *Interfaces* panel.

Step 4: Click the **Load Selected Class** button.

The class (including all of its methods) is loaded as a separate node in the security object hierarchy, and if it has access details in persistent storage they are retrieved and loaded also.

Appendices

Appendix A

XML Configuration Files

Overview

All properties for the OpenFusion CORBA Services are stored in and controlled from XML files, making the list of properties flexible and extensible.

It is possible to directly edit property values in the XML files, although it is recommended that the Administration Manager interface be used. The Administration Manager provides proper validation of input and reduces errors.

Manually editing the XML configuration files is not recommended, but it would be possible to programmatically alter the properties. The XML files are described here for developers who wish to do that. The OpenFusion graphical tools include an XML editor which performs validation against XML DTDs. A command-line utility for setting properties is also provided; this is described in *Command-line Configuration* on page 161.

All configuration files are stored under the OpenFusion installation directory.

Take great care when manually editing XML files as errors can seriously interfere with the functioning of the OpenFusion graphical tools and Services.

WIN Directory paths given as examples in this section use Unix conventions. Users of OpenFusion on Windows NT should make the appropriate substitutions.

The Object Hierarchy

The `domains` directory under the OpenFusion installation directory contains the XML files that record the current configuration of the OpenFusion installation. The `domains` directory structure maps directly to the *Object Hierarchy* in the Administration Manager, so a directory exists for each domain, node, Service, Singleton, and Java Object. The directory must have exactly the same name as the domain, node, Service, Singleton, or Java Object it represents.

For example, the *localhost* node in the Administration Manager's default *Object Hierarchy* is represented by the following directory structure:

```
<INSTALL>/domains/OpenFusion/localhost
```

where `<INSTALL>` is the OpenFusion installation directory.

If the *Object Hierarchy* is altered or added to through the Administration Manager (see *Extending the Object Hierarchy* on page 19), new directories and XML files are created to reflect the new structure.

Figure 18: shows the correlation between the *Object Hierarchy* and the `domains` directory structure (the illustration is from Windows NT, however the same structure is used on UNIX)..

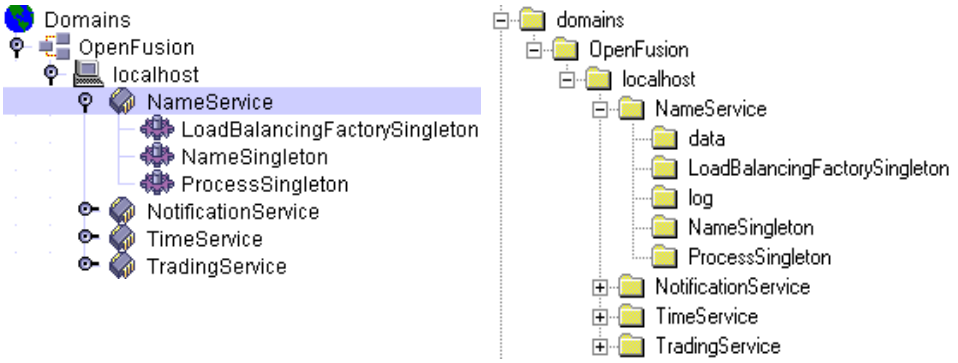

Figure 18: Object Hierarchy and Directory Structure

Completing the XML File Installation

The normal installation of OpenFusion CORBA Services creates minimal XML configuration files in the `domains` directory structure. These files only contain configuration information for properties which differ from the default values. To fully populate these XML files with property information, you must run the Administration Manager and save the configuration.

As an alternative to running the Administration Manager GUI, the configuration can be completed using the Administration Manager command line tool, described in 7.2, *Administration Manager Tool* on page 117.

This *must* be performed before any Services can be started from the command line using the `server -start` script (as described in *Starting Servers from the Command Line* on page 8).

Directory Tree

The structure of the `domains` directory is shown in *Figure 19:*.

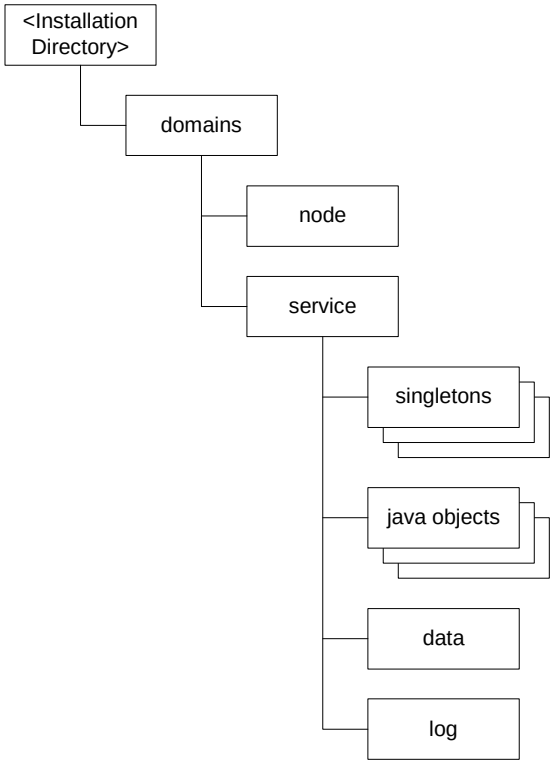

Figure 19: Domains Directory Tree

Configuration information is stored in XML files at each level of the directory tree.

XML Files

Domains and Nodes

Each domain and node directory must contain a single file, `<name>.xml`, where <name> is the name of the domain or node.

The domain and node files list all the children of that domain or node. They also show whether the *Object Hierarchy* has been locked at that level (see *Locking Nodes* on page 23).

These files are located and named as follows:

```
<INSTALL>/domains/<domain>/<domain>.xml
<INSTALL>/domains/<domain>/<node>/<node>.xml
```

where <INSTALL> is the OpenFusion installation directory, <domain> is the name of the domain, and <node> is the name of the node.

For example, the *localhost* node in the Administration Manager's *Object Hierarchy* is defined in the following XML file:

```
<INSTALL>/domains/OpenFusion/localhost/localhost.xml
```

The XML file also records whether or not the node is locked (see *Locking Nodes* on page 23).

The format of the XML files for domains and nodes is defined in the following DTD files:

```
<INSTALL>/xml/schema/Domain.dtd
<INSTALL>/xml/schema/Node.dtd
```

Services

Each Service directory must contain a single file, <service>.xml, where <service> is the name of the Service.

The service file lists the Singletons and Java Objects under that Service. They also show whether the *Object Hierarchy* has been locked at that level (see *Locking Nodes* on page 23) and store any run time properties for the service.

These files are located as follows:

```
<INSTALL>/domains/<domain>/<node>/<service>/<service>.xml
```

where <INSTALL> is the OpenFusion installation directory, <domain> is the name of the domain, <node> is the name of the node, and <service> is the name of the Service.

For example, the *NameService* node in the Administration Manager's *Object Hierarchy* is defined in the following XML file:

```
<INSTALL>/domains/OpenFusion/localhost/NameService/NameService.xml
```

The XML file also records the current value and locking status of each property belonging to the Service. See *Common Configuration Properties* on page 49 for details of Service properties.

The format of the XML files for Services is defined in the following DTD file:

```
<INSTALL>/xml/schema/Service.dtd
```

Singletons

Each Singleton directory must contain a file, `<singleton>.xml`, where `<singleton>` is the name of the Singleton. The directory also contains the Singleton's IOR file (after the Service has been started).

These files are located as follows:

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

where `<INSTALL>` is the OpenFusion installation directory, `<domain>` is the name of the domain, `<node>` is the name of the node, and `<service>` is the name of the Service that contains the Singleton.

For example, the *NameSingleton* Singleton in the Administration Manager's *Object Hierarchy* is represented by the following XML file:

```
<INSTALL>/domains/OpenFusion/localhost/NameService/NameSingleton/
     NameSingleton.xml
```

The `XML` file records the current value and locking status of each property belonging to the Singleton. See *Common Configuration Properties* on page 49 for details of properties.

The format of the XML files for Singletons is defined in the following DTD file:

```
<INSTALL>/xml/schema/Singleton.dtd
```

Java Objects

Each Java Object directory must contain a file, `<javaobject>.xml`, where `<javaobject>` is the name of the Java Object.

This file is located as follows:

```
<INSTALL>/domains/<domain>/<node>/<service>/<javaobject>/<javaobject>.xml
```

where `<INSTALL>` is the OpenFusion installation directory, `<domain>` is the name of the domain, `<node>` is the name of the node, and `<service>` is the name of the Service that contains the Java Object.

For example, the *ChannelConfiguratorObject* Java Object in the Administration Manager's *Object Hierarchy* is represented by the following XML file:

```
<INSTALL>/domains/OpenFusion/localhost/NotificationService/
     ChannelConfiguratorObject/ChannelConfiguratorObject.xml
```

The `XML` file records the current value and locking status of each property belonging to the Java Object. See *Common Configuration Properties* on page 49 for details of properties.

The format of the XML files for Java Objects is defined in the following DTD file:

```
<INSTALL>/xml/schema/JavaObject.dtd
```

See Appendix C, on page 167, for details of how to create and configure Java Objects.

XML Templates

The `xml/templates` directory under the OpenFusion installation directory contains the XML files that define the properties for every object in the Administration Manager's *Object Hierarchy*.

To add a property to a Service, Singleton, or Java Object in the *Object Hierarchy*, the property must be fully defined in the appropriate XML file, as described below.

See *Properties* on page 25 for details of how properties are displayed in the Administration Manager.

Directory Tree

The structure of the `templates` directory is shown in *Figure 20:*

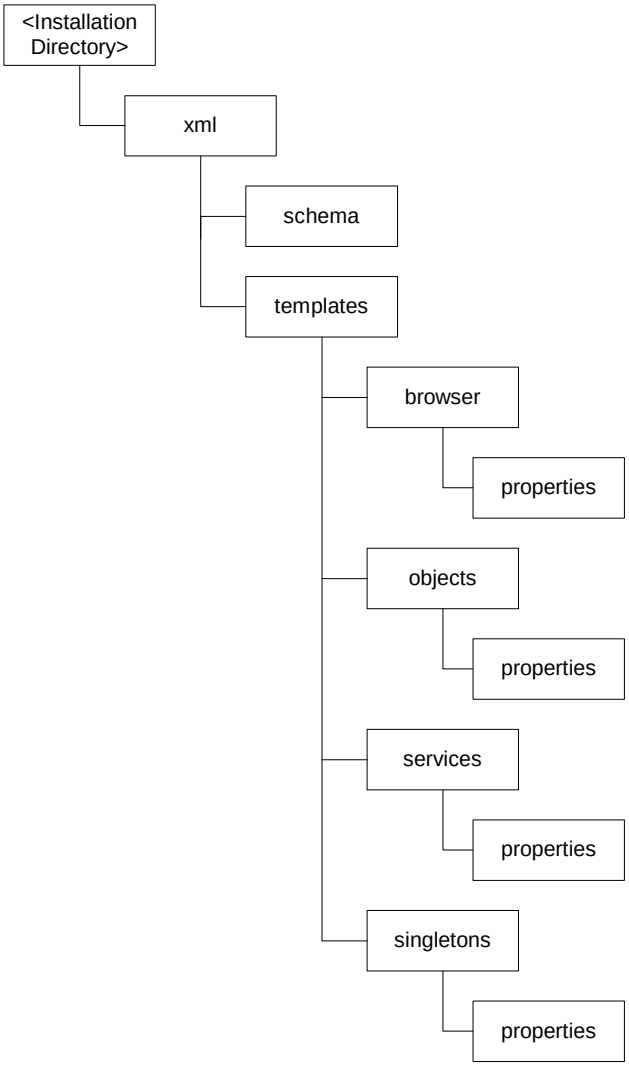

Figure 20: Templates Directory Tree

Defining a Property in the XML File

The XML DTDs in the `schema` directory (see *XML Schema* on page 160) define how a new property must be created in the XML files. The following notes give further explanations of the XML elements.

GroupName

In the Administration Manager, each group is assigned a different pane and identified by a named tab. The *GroupName* element identifies which tab the property will appear on.

CategoryName

This is an organisational element. Properties on the same tab which also have the same *CategoryName* are grouped together on the tab.

Dependencies

Dependencies describe the relationship between different set of properties. If setting a property to a specific value will disable (lock) or enable (unlock) other properties, the *Dependencies* element should be used to show that relationship.

Conditional

Some properties apply only to specific system configurations. For example, some properties relate to a specific ORB and will not appear on the Administration Manager screens if a different ORB is in use. *Conditional* elements, if present, show which configurations are required for the property to be valid. The conditions currently supported are CORBA version, Java version, operating system, and ORB version. Permitted values are shown in the DTD.

XML Schema

All XML files used to configure the managers and browsers must conform to the DTDs in the directory:

```
<INSTALL>/xml/schema
```

where `<INSTALL>` is the OpenFusion installation directory.

The configuration DTD files are listed in *Table 15*.

Table 15 DTD Files

File	Function
`Domain.dtd`	Describes a domain object the *Object Hierarchy*.
`JavaObject.dtd`	Describes a Java Object in the *Object Hierarchy*.
`Node.dtd`	Describes a node object in the *Object Hierarchy*.

Table 15 DTD Files (Continued)

File	Function
`Properties.dtd`	Describes the properties of an object and controls how they are displayed in the property pane of the Administration Manager.
`Service.dtd`	Describes a service process object in the *Object Hierarchy*.
`Singleton.dtd`	Describes a Singleton object in the *Object Hierarchy*.

Warning: under no circumstances alter any DTD files in the schema directory.

Command-line Configuration

The OpenFusion CORBA Services distribution includes a command-line utility `com.prismt.openfusion.tools.ChangeSettings` which can be used to set properties in the XML configuration files. This provides an alternative to using the Administration Manager and can be useful when performing a command-line or script-driven install of OpenFusion.

This utility is run as follows:

```
% java com.prismt.openfusion.tools.ChangeSettings <dir>
  <property> <value> [ <property> <value> ... ]
```

Where:

`<dir>` is the directory that contains the XML files.
`<property>` is the name of the property which is to be set.
`<value>` is the value that the property is to be set to.

The utility will search all XML files in the specified directory, and recursively in all directories below that directory, for incidences of the specified property. Wherever an incidence of that property is located, it is set to the specified value.

Multiple property-value arguments can be specified, allowing several properties to be set in a single operation.

Properties can only be set in XML files which conform to the DTD for OpenFusion property files.

Because directories are searched recursively, care must be taken when specifying the directory argument. If a property exists in multiple different services but should be set to a different value in each service (`IOR.URL`, for example), it would be a bad idea to set that property by running the utility at the `domains` directory level.

i All Service properties are named and described in the Configuration and Management section of each Service guide. Properties common to all Services are documented in Section 3, *Common Configuration Properties*, on page 49 of this guide.

Appendix B

Log Messages

Overview

OpenFusion uses the log4j package to support error diagnostics and logging. This is a public domain logging package. Further details can be found at *http://jakarta.apache.org/log4j*.

Conceptually, the log4j package supports appenders and layout managers. Appenders direct output to a particular destination such as file or system log. Layout managers can be used to format the generated log message. Every appender has an associated layout manager.

The layout of the log messages in OpenFusion is set to the default pattern layout of the log4j package, which means only the message and severity appear. The format of the output message can be customised by the use of a pattern layout manager and an associated conversion pattern. See the on-line log4j documentation for details on how this can be done.

Using a Pattern Layout

To use a pattern layout for a Service, the *Log Layout* property for that Service must be set to *Pattern*. See *Log Layout* on page 55 for details of how to set this in the Administration Manager.

The format of the output message is customised by entering a pattern string in the *Log Pattern* property for the service. See *Log Pattern* on page 55.

For example, to prefix the date and time to the log messages generated by the Notification Service use the following pattern:

```
%d{DATE} - %m%n
```

It is recommended that `%n` is always appended to the end of any log pattern. This forces a line break at the end of each message and makes the log file easier to read.

Conversion Characters

The characters that can be used in a logging pattern are shown in Table 16, *Conversion Characters*. The patterns are case-sensitive.

Table 16 Conversion Characters

Conversion Character	Effect
%c	The category of the logging event. The category conversion specifier can be optionally followed by a *precision specifier*, which is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right-most components of the category name will be printed. By default, the category name is printed in full. For example, for the category name "a.b.c" the pattern **`%c{2}`** will output "b.c".
%C	The fully-qualified class name of the caller issuing the logging request. This conversion specifier can be optionally followed by a *precision specifier*, which is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right-most components of the class name will be printed. By default the class name is output in fully-qualified form. For example, for the class name "org.apache.xyz.SomeClass" the pattern **`%C{1}`** will output "SomeClass". **WARNING:** Generating the caller class information is slow. Its use should be avoided unless execution speed is not an issue.
%F	The file name where the logging request was issued. **WARNING:** Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.

Table 16 Conversion Characters (Continued)

Conversion Character	Effect
%d	The date of the logging event. The date conversion specifier may be followed by a *date format specifier* enclosed between braces. For example, **%d{HH:mm:ss,SSS}** or **%d{dd MMM yyyy HH:mm:ss,SSS}**. If no date format specifier is given then ISO8601 format is assumed. The date format specifier uses the same syntax as the time pattern string of the SimpleDateFormat. Although part of the standard JDK, the performance of SimpleDateFormat is quite poor. For better results it is recommended to use the log4j date formatters. These can be specified using one of the strings ABSOLUTE, DATE and ISO8601 for specifying AbsoluteTimeDateFormat, DateTimeDateFormat, and ISO8601DateFormat respectively. For example, **%d{ISO8601}** or **%d{ABSOLUTE}**. These dedicated date formatters perform significantly better than SimpleDateFormat.
%l	Location information of the caller which generated the logging event. The location information depends on the JVM implementation but usually consists of the fully qualified name of the calling method followed by the caller's source file name and line number between parentheses. The location information can be very useful. However, its generation is *extremely* slow. Its use should be avoided unless execution speed is not an issue.
%L	The line number from where the logging request was issued. **WARNING:** Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.
%m	The application-supplied message associated with the event being logged.

Table 16 Conversion Characters (Continued)

Conversion Character	Effect
%M	The name of the method where the logging request was issued. **WARNING:** Generating caller location information is extremely slow. Its use should be avoided unless execution speed is not an issue.
%p	The priority of the logging event.
%r	The number of milliseconds elapsed since the start of the application and the time of creation of the logging event.
%t	The name of the thread that generated the logging event.
%x	The NDC (nested diagnostic context) associated with the thread that generated the logging event.
%n	A platform-dependent new line character (usually included at the end of each logging pattern, to force one message per line in the log file). This conversion character offers practically the same performance as using non-portable line separator strings such as “\n”, or "\r\n". Thus, it is the preferred way of specifying a line separator.
%%	A single percent sign (required to escape the percent sign).

Appendix C

Managing Java Objects

Overview

The OpenFusion Administration Manager can be used to manage and configure user-defined Java Objects. Java Objects can be added to the Administration Manager's Object Hierarchy as described in *Extending the Object Hierarchy* on page 19. To make a Java Object available for management in the Administration Manager, the Java Object must be set up as described in this Appendix.

The Administration Manager can be used to configure the Java Object's properties. To provide further management facilities, a custom GUI browser (which must extend `com.prismt.browser.BaseBrowser`) can be created for the Object. The browser can be launched from the Administration Manager (effectively acting as an Administration Manager plug-in) or started from a command line.

Creating the Java Object

If a Java Object is to use the ORB or any of the properties passed through from the Openfusion Service, then it must implement the `com.prismt.openfusion.plugin.JavaObject` interface. This interface is defined as follows:

```
public interface JavaObject
{
   public void init (org.omg.CORBA.ORB orb, ExtendedProperties props) throws Exception;
}
```

`ExtendedProperties` is found in the `com.prismt.util` package, which would have to be imported.

Management of the Java Object also requires a default constructor. When a Service containing the Java Object is started, the Administration Manager calls the Object's default constructor followed by the `init` method (if the JavaObject interface is not implemented then only the default constructor is called).

Describing the Java Object in XML

An XML file must be created for each Java Object and placed in the `<INSTALL>/xml/templates/objects` directory (where `<INSTALL>` is the OpenFusion installation directory). This file should be given the name of the Java Object. For example:

`<INSTALL>/xml/templates/objects/MyObject.xml`

The presence of this file makes the Java Object available for adding to the Object Hierarchy.

See *XML Configuration Files* on page 153 for more of the XML files used by OpenFusion.

The Java Object definition file must conform to the DTD specified in `<INSTALL>/xml/schema/JavaObject.dtd`.

The following example illustrates the `MyObject.xml` file for the `MyObject` Java Object:

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE JavaObject SYSTEM "file://PrismTech/OpenFusion/xml/schema/JavaObject.dtd">
<JavaObject>
  <Name>
    My Object
  </Name>
  <ClassName>
    user.path.MyObjectImpl
  </ClassName>
  <Browser>
    <BrowserName>
      My Object Manager
    </BrowserName>
    <BrowserClassName>
      user.path.browser.MyObjectBrowser
    </BrowserClassName>
  </Browser>
</JavaObject>
```

Name - The name of the Java Object as it will appear in the menu of available objects in the Administration Manager (this is illustrated in Figure 5, *Adding a Java Object* on page 21).

ClassName - The name of the class which is actually executed.

BrowserName - The name of the GUI browser which will be used to manage and configure the Java Object. This is the name which will be displayed on screen in the Administration Manager (optional).

BrowserClassName - The class name of the GUI browser used to manage and configure the Java Object (optional).

Defining Properties for the Java Object

If the Java Object has properties which should be set through the Administration Manager, the properties must be described in an XML file in the `<INSTALL>/xml/templates/objects/properties` directory (where `<INSTALL>` is the OpenFusion installation directory). This file should be given a name of the form `<Java-Object>Properties.xml`. For example:

```
<INSTALL>/xml/templates/objects/properties/MyObjectProperties.xml
```

The properties XML file must conform to the DTD specified in `<INSTALL>/xml/schema/Properties.dtd`. See *Defining a Property in the XML File* on page 159 for details.

The Object Hierarchy

When an instance of the Java Object is added to a Service in the Administration Manager, an XML file is created for it. This file records the current value and locking status of each property belonging to the Java Object instance. The file is located as follows:

```
<INSTALL>/domains/<domain>/<node>/<service>/<javaobject>/<javaobject>.xml
```

where `<INSTALL>` is the OpenFusion installation directory, `<domain>` is the name of the domain, `<node>` is the name of the node, `<service>` is the name of the Service, and `<javaobject>` is the name of the Java Object.

For example, an instance of the *ChannelConfiguratorObject* Java Object in the Administration Manager's *Object Hierarchy* could be represented by the following XML file:

```
<INSTALL>/domains/OpenFusion/localhost/NotificationService/
     ChannelConfiguratorObject/ChannelConfiguratorObject.xml
```

Glossary

Glossary

Term	Meaning
Activate	Prepare an object to receive requests.
Active Object Map	A table of associations between Object IDs and Servants, which is maintained by a POA to allow it to dispatch incoming requests.
Administration Manager	Tool used to manage and configure the OpenFusion Services.
alias	An additional or alternative name for the same thing; an object containing the name of another object. Aliases enable one object to have more than one name. For example: LoadBalancer and LoadBalancerAlias refer to same object. This enables LoadBalancer to change dynamically even after being bound into the Naming Service.
AMI	See *Asynchronous Messaging Interface*.
AOM	See *Active Object Map*.
Asynchronous Messaging Interface	This is an extension of CORBA functionality into a complete messaging semantics (as opposed to request brokering). This includes various modes of communication between the originator and recipient and also various qualities of service.
bind	To bind is to associate a meaningful name with an object reference as a name-value pair. Binding is the process of associating a name with a remote object in a server application, so that a client application can resolve the name and obtain a reference to the (remote) object. A binding is an association between a name and a reference.
BOA	Basic Object Adapter – the standard within CORBA versions 2.2 and lower which specifies how objects invoke and obtain references to each other. An object adapter is the way in which a programming language object in the server is associated with a CORBA object. The BOA loosely describes an inheritance and delegation based approach. The BOA has been deprecated and is superseded by the POA.
CCM	CORBA Component Model.

Term	Meaning
CFA	Common Facilities Architecture.
composite namespace	See *Federated namespace*.
ConnectionFactory	A connection factory is an administered object that JMS clients use primarily for bootstrapping purposes. There are both topic and queue connection factories, which are typically resolved from the Java Naming and Directory Interface (JNDI). Clients use the factory object to create new connections.
Constraint	Selection criterion or search condition. See *TCL* (Trader Constraint Language).
context	See *Naming context*.
CORBA	Common Object Request Broker Architecture. An open standard for interoperable distributed object systems developed and maintained by the OMG. The standard only defines the architecture; it is up to individual companies how they produce actual implementations.
Corbaloc	The Corbaloc URL scheme provides URLs that are familiar to people and similar to ftp or http URLs. This URL format is independent of the Naming Service.
Corbaname	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.
Core Object Model	The fundamental object-oriented model in the OMA which defines the basic concepts on which CORBA is based.
COS	CORBA Object Service. This is a label for a broad set of add-on services that extend the core CORBA specification.
cyclic	A ‘backward’ reference from a naming context to a ‘parent’ or ‘grandparent’ context in the same naming graph.
DCE	Distributed Computing Environment. A distributed computing architecture developed by the OSF before CORBA.

Term	Meaning
DCOM	Distributed Component Object Model (COM). The architecture and implementation of the distributed request/response technology from Microsoft.
delegate	One object only presents an interface and makes a local call to another object which actually implements the functions offered by the first. Enables separation of application and control interfaces; used in OpenFusion Load Balancing (a Naming Service option). Also referred to as a 'Tie' in CORBA.
Directory Service	A service providing facilities for organising and finding objects. The service often includes operations for creating, adding, removing, and modifying the attributes associated with objects in a directory. The CORBA Naming and Trading Services are collectively referred to as directory services.
DTD	Document Type Definition. A file containing declarations that specify a format for XML files.
DTF	Domain Task Force; OMG working group.
EJB	Enterprise Java Beans. Part of the J2EE standard which defines the application and data component models.
ESIOP	Environment-Specific Inter-ORB Protocol. The implementation of GIOP for a non-TCP/IP environment, e.g., DCE.
etherealize	The action of destroying a Servant associated with an Object ID, so that the ID no longer identifies a CORBA object with respect to a particular POA.
fail-over	The use of two or more systems running in parallel so that if one fails another immediately takes over with no disruption apparent to users. Normally implemented as two identical synchronised systems, nominated "master" and "slave". The slave takes over if the master fails. Fail-over is usually performed at a low level and is therefore transparent to applications.
federated namespace	A single logical namespace comprised of multiple autonomous naming systems.

Term	Meaning
federation	A grouping of autonomous systems linked in such a way as to appear to be or to work as a single system. A component within one system should be able to communicate or interact with a component of a different system as though it were communicating or interacting with another component in the same system, even though the systems may be implemented on different platforms using different languages and/or protocols.
GIOP	General Inter-ORB Protocol. The high level specification of wire protocol introduced in CORBA 2.0. All CORBA 2.0-compliant ORBs use this common wire protocol specification, which allows clients and servers using different ORBs to interoperate. GIOP is implemented using a network protocol. See also *IIOP* and *ESIOP*.
HTTP	Hypertext Transfer Protocol. The standard Internet transport protocol for HTML documents.
IDL	Interface Definition Language. A high-level declarative language for defining the interfaces of distributed objects. It is used for the definition of CORBA services and objects because it is platform-independent.
IDL compiler	An application which converts an IDL specification into programming-language-specific stub and skeleton files which are used to implement distributed objects.
IFR	Interface Repository.
IIOP	Internet Inter-ORB protocol. A TCP/IP-based protocol developed by the OMG. The IIOP enables multiple ORBs to interoperate to provide requests to objects. The implementation of GIOP for TCP/IP.
incarnate	The action of providing a running Servant to serve requests associated with a particular Object ID. A POA will keep this association in its Active Object Map.
initial context	The starting point for the resolution of names for naming and directory operations. Also known as a *root* context.

Term	Meaning
INS	Interoperable Naming Service. One of the CORBA services. The Interoperable Naming Service functions just like the Naming Service, holding bindings between meaningful names and Object References (IORs). The INS provides additional “under the covers” support for interoperability between different ORBs.
instrumentation	Functions which return information about the current status of a system or items within it. Used for monitoring performance and detecting problems. For example, resetable counters can report the number of events occurring in a specified time interval (see also *Quality of Service*).
Interface Repository (IFR or IR)	The CORBA service (server or component) that stores meta-data about IDL interfaces. A CORBA component which stores type information and makes it available through standard interfaces at run time. It contains all the registered component interface definitions, including the methods they support and the parameters they require. Programs may use the IFR APIs to access and update this information.
IOP	Inter-Operable Protocol.
IOR	Interoperable Object Reference. An Object Reference (OR) is the way a CORBA Object is named. The IOR is the CORBA 2.x-compliant format for a standard representation of an OR for all ORBs.
Istring	An IDL data type, the “internationalized string”, which is not implemented. In the original CORBA specifications, the Istring was “a placeholder for an internationalized string data type”; it is now only retained for compatibility reasons, and is always mapped to the string data type with `typedef string Istring` in IDL.
JDMK	Java Dynamic Management Kit.
JDO	Java Data Objects.
Jini	A distributed system based on the idea of federating groups of users and the resources required by those users.

Term	Meaning
JMS	Java Message Service. Part of the J2EE standard specifying how applications can send asynchronous messages to each other.
JMX	Java Management eXtensions.
JNDI	Java Naming and Directory Interface. JNDI is a standard extension to the Java platform which provides Java-enabled applications with a unified interface to multiple naming and directory services.
JTS	Java Transaction Service; an API defined as a part of the J2EE specification for transactional capabilities.
LDAP	Lightweight Directory Access Protocol; an API defined to provide a common interface to data stores, regardless of their underlying nature and location.
load balancing	Optimisation of 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. Load balancing involves the distribution of requests for a particular service amongst multiple servers which provide that service. The methods used to allocate requests are known as *policies*.
marshalling	Conversion of data into a programming-language- and architecture-independent format ready for transmission.
MessageConsumer	A message consumer is an object that receives JMS messages. A consumer in the point-to-point model is referred to as a *queue receiver*, while the publish/subscribe model uses the term *topic subscriber*. Message consumers are created by sessions and may use either a push model to receive messages asynchronously, or a pull model to receive messages synchronously. JMS supports both transient and durable message consumers.
MessageProducer	A message producer is an object that sends JMS messages. A producer in the point-to-point model is referred to as a *queue sender*, while the publish/subscribe model uses the term *topic publisher*. Message producers are created by sessions.

Term	Meaning
meta-data	Data which describes the format of the representation or storage of other data.
MOF	Meta-Object Facility.
MSMQ	Microsoft Message Queue; a messaging product from Microsoft.
MTS	Microsoft Transaction Server; a transaction processor product from Microsoft.
name binding	In the Naming Service name bindings are contained in naming contexts. A binding can refer to either an object or another naming context. The process of associating a Name with a remote object in a server application, so that a client application can resolve the Name and obtain a reference to the remote object.
name resolution	The process of *resolving* a name to obtain a reference to the object to which it is bound.
name space	The set of all names in a naming system.
naming context	An object containing name bindings which refer to other objects, which may be naming contexts. A set of naming contexts which can be traversed by following (resolving) the bindings it such bindings is a naming graph.
naming graph	An hierarchy of naming contexts and objects in a Naming Service. Name bindings are contained in naming contexts. A binding can refer to either an object or another naming context. A set of naming contexts which can be traversed by following (resolving) such bindings is a Naming graph.
naming system	A connected set of naming contexts. The naming contexts are all of the same type, have the same naming convention, and provide the same set of operations with identical semantics.
NTP	Network Time Protocol.
object adapter	The ORB component which provides object reference, activation, and state related services to an object implementation. See also *BOA* and *POA*.

Term	Meaning
OMA	Object Management Architecture. The overall architecture and roadmap of the OMG, of which CORBA forms a part.
OMG	Object Management Group. A cross-industry consortium which develops and promotes the CORBA open-systems standards.
OR	Object Reference. The way CORBA objects are identified.
ORB	Object Request Broker.
OSF	Open Software Foundation.
OTM	Object Transaction Monitor. A set of CORBA Services for developing Enterprise systems.
PIDL	Pseudo Interface Definition Language (Pseudo-IDL). This is identical to IDL, however it is not used for describing a remotely accessed CORBA Object but rather an object in the CORBA infrastructure that is implicitly local.
POA	Portable Object Adapter. An object adapter is the way in which a programming language object in the server is associated with a CORBA object. The POA describes a full set of models and policies for managing object life-cycles. (Introduced in CORBA 2.3, superseding the BOA.)
point-to-point	Method of event delivery which ensures that an event generated by a supplier is received by a subscriber once (and only once). Sometimes referred to as “exactly once” delivery. Contrast with normal “at least once” delivery (as in the publish-subscribe model).
POS	Persistence Object Service. A deprecated CORBA Service for storing the state of implementation objects into a database. The PSS supersedes the POS.

Term	Meaning

provider resource file	An optional properties file named [*prefix*/]jndiprovider.properties, where *prefix* is the package name of the service provider class with each period character converted to a forward slash character (“/”). This file is used by the JNDI when determining the values of the following JNDI-defined properties:
	`java.naming.factory.object`
	`java.naming.factory.state`
	`java.naming.factory.control`
	`java.naming.factory.url.pkgs`

PSS	Persistence State Service. A CORBA Service for storing the state of implementation objects into a database. The PSS supersedes the POS.
QoS	Quality of Service.
reference	Information needed for accessing an object. It contains one or more addresses for referring to or communicating with an object. See also *Object reference*.
resolve (resolution)	The process of obtaining an object reference from a name binding. (See also *name binding*.)
rollback	To undo the successful steps of a sequence of operations when one step fails. Part of a method of ensuring database integrity whereby if any one of a group of related operations or updates fails, then all the other operations in the group are undone and the database is restored to the state it was in before the operations were attempted.
RPC	Remote Procedure Call. A strategy which allows procedures to be called from outside the currently running program's memory. RPC allows two or more different programs to interoperate with one another.
RUP	Rational Unified Process.
SASL	Simple Authentication and Security Layer.

Term	Meaning
servant	An implementation object that provides the run-time semantics of one or more CORBA objects. An instance of an object implementation for an IDL interface. The servant object is registered with the ORB so that the ORB knows where to send invocations. It is the servant that performs the services requested when a CORBA object's method is invoked.
session	A session is the context used by clients for sending and receiving messages. Sessions are created by connections and are factories for creating message suppliers and consumers as well as message objects. Each session retains messages received by all its consumers until they have been acknowledged.
SID	Service ID, Server Persistent ID, or Server persistence UUID scope.
SII	Static Invocation Interface (or Stub Invocation Interface). This is the client-side API for generating network messages that is based on the stubs that are code-generated by the IDL compiler for a given IDL interface. See also *DII*.
SNMP	Simple Network Management Protocol. A widely used standard for specifying systems management interfaces and operations.
SOAP	Simple Object Access Protocol. A protocol for sending RPC calls over the Internet encoded as XML and using the HTTP protocol. This is intended to avoid the firewall problems that face the use of protocols such as IIOP.
SSL	Secure Socket Layer.
stringification	Conversion of an object reference to a character string. Used when an object reference needs to be saved in a text file or stored in a database (persistence) or sent to a client program.
TCL	Trader Constraint Language. A simple language used for constructing constraints (also referred to as search conditions or selection criteria) used in queries to retrieve offers from servers.

Term	Meaning
TCP/IP	Transport Control Protocol/Internet Protocol. A network protocol used on the Internet and many internal networks. TCP is for establishing connections between hosts and guaranteeing delivery of data packets in the correct order; IP determines the structure of the packets themselves.
TOG	The Open Group.
TP	Transaction Processor. Typically used as TP Monitor. A kind of middleware that manages connections and transactions to databases.
trader	An object which supports the Trading Service. A trader can be a server, a client, or both. The components (Register, Proxy, Lookup, Admin, Link) provide the functions for handling offers of service and queries.
transaction server	A server which supports transactional semantics, (for example, commit or rollback).
UML	Unified Modelling Language. A standard developed and maintained by the OMG to facilitate object analysis and design representation. A method of modelling any process using simple diagrams; used for communications and analysis/design.
UUID	Universally Unique IDentifier. A 128-bit identifier generated by an algorithm which will never produce the same value twice and hence can uniquely identify entities in a distributed system.
XMI	XML Metadata Interchange.
XML	eXtensible Markup Language. A standard for representing data in a language- and database-neutral format. XML separates a document's definition, content, and presentation (style).
XSL	eXtensible Stylesheet Language. A standard for defining the formatting of an XML document.
XSLT	XSL Transformation. A standard for describing transformations between XML documents.

Index

Index

A

Access Control List 124
Access Entry Details. 145
Accessibility 27
ACL 124
ACL Groups 129
acls (property). 138
Add new principals. 145, 147
Adding
  Java Objects 20
  Nodes 19
  Singletons 20
Administration Manager. 13
  Properties 71
Administration Manager Tool 117
adminMgrTool 117
Appenders. 163
Assign a Principal to a Class or Method 147
Assign Principals Globally. 149
Assign Value
  Globally 30
  to Peers 29
  to Properties 27
Authentication 30, 124
Authorised Credentials. 131

B

BAD_PARAM Exception Count (property) . . . 60
BaseBrowser class 167
Browser
  CORBA Object. 34
  Framework 11
  Log 33
  Save Configuration. 33
BrowserClassName element 168
BrowserName element. 168

C

C++ Support 106
CategoryName 160
Central Configuration
  Set up Host 36
Central Configuration Host (property) . . 40, 41, 73
Central Host 36
Changing the Order of Services and Singletons. 22
ClassName element 168
CLASSPATH
  Jar files 103
  ORB jar files. 7
clients
  user defined. 102
clientSideLogin (property) 138
Command Line Switches 12
  -noorb 12
  -port. 12
  -remote 12
  -start 9
  -status 9
  -stop 9
Command Line Tools. 117
Conditional Properties 27, 160
ConfigFile object (property). 138
configGen. 119
Configuration 133
  Distributed Installation 36
  Files 153
  from the Command Line. 161
  Save 33
Configuration Generator 119
Configuration Manager
  configuring persistent storage
    JDBC Data Source 107

Configure from Remote Host (property). . . 40, 41
Configuring 78, 161
Secure Client 136
Secure Service 133
ConfigViaWebServer (property). 74
Conversion Characters 164
CORBA Object
Browser 34
CORBA Object Activity Timeout (property) . . 61
CORBA.BadParamExceptions (property) 60
CORBA.Calls (property) 62
CORBA.InitializeExceptions (property). 60
CORBA.InternalExceptions (property) 60
Creating
ACL Groups 129
Principal Mappings 130
Credentials 131

D

Daemons 7
DB.JBDC.Driver (property) 53
DB.JDBC.AutoCreate (property) 50
DB.JDBC.Handler (property) 51
DB.JDBC.Logging (property). 53
DB.JDBC.Password (property). 54
DB.JDBC.Type (property) 51
DB.JDBC.URL (property) 53
DB.JDBC.User (property). 54
DB.WriteBatch (property). 50
DB.WriteInterval (property) 49
Default Trap Community (property). 80
Default Trap Port (property) 80
DefaultTrapCommunity (property). 80
DefaultTrapPort (property) 80
Delete
Nodes. 21
selected browser 33
Delete Access Entries Globally. 148
Dependencies 160
Directory Tree. 155, 158
Distributed Installation 36
Domain 16, 155
Domain Configuration Parameters 13
Domain.dtd 160
DynAnyFactory class 99
creation operations 99

E

Enable Dynamic Portable Interceptors (property) 62
Enable Traps (property) 79
Enable Write Access (property) 81
EnableDynamicInterceptors (property). 62
EnableTraps (property) 79
EnableWriteAccess (property) 81
Enter user identity 33
Event Log option. 58

F

File
Browser 33
option. 58
File Append (property) 57
File Backup Number (property) 57
File Maximum Size (property) 57
fileLocations (property group) 137

G

Generic Security Service Username and Password 126

Group Persistence File
Example 130
GroupName 160
Groups 125, 129
groups (property) 138
GSSUP 126, 131
gssupCredential (property) 137
gssupUsers (property) 138

H

hsqldb 107, 112
client/server 113
HSQLDBObject 113
Name 113
NoSystemExit 115
Port 115
Silent 115
Timeout 114
Trace 114

I

Identifiable interface 123
IDL
compiling 104
Implementation Name 65
Implementation Repository 38
Incoming Call Count (property) 62
Informix 107, 111
INITIALIZE Exception Count (property) 60
Instrumentation 77
Interfaces 149
INTERNAL Exception Count (property) 60
IOR Decoder 117
IOR File Name (property) 71
IOR Name Service (property) 69
IOR Name Service Entry (property) 70
IOR URL (property) 70
IOR.File (property) 71
IOR.Name (property) 70
IOR.Server (property) 69
IOR.URL (property) 70
iorDecoder 117

J

JAAS 123, 135
JAAS Configuration File (property) 135
jaasLoginConfig (property) 138
jaasLoginConfigName (property) 138
JacORB 103, 105
jacorb.properties file 38
Jar Files 103
Java
IDL Compilation 104
Properties 66
Java Authentication and Authorisation Service 123
Java Object
Properties 168
Java Objects 16, 157, 167
Adding 20
Managing 167
java.security.auth.login.config (system property) . 135, ... 138
JavaObject interface 167
JavaObject.dtd 160, 168
JDBC Auto-create tables (property) 50
JDBC Data Source 107
hsqldb 112
Informix 111
Oracle 109
SQL Server 112
Sybase 110
JDBC Database Type (property) 51, 107
JDBC Driver 53, 107
JDBC Handler (property) 50, 107
JDBC Logging (property) 53, 107
JDBC Password (property) 54, 107

JDBC URL (property) 52, 107
JDBC User (property) 54, 107
JVM Flags (property) 66
JVM Free Memory (property) 67
JVM Information (property) 66
JVM Total Memory (property) 67
JVM.Flags (property) 66
JVM.FreeMemory (property) 67
JVM.Info (property) 66
JVM.TotalMemory (property) 67
JVM.XBoot (property) 67

L

Launch
  file browser 33
Layout Managers 163
License File 16
Load CORBA Singletons on Startup (property) 62
LoadOnStart (property) 62
localhost 14, 16
Locking
  Nodes 23
  Properties 22
Log
  Layout 163
  Messages 163
Log File (property) 57
Log Layout (property) 55
Log Level (property) 59
Log Pattern (property) 55
Log Plugin (property) 58
Log Service
  Plugin option 58
log4j 163
log4j.appender.Default (property) 59
log4j.appender.Default.Append (property) 57
log4j.appender.Default.Facility (property) 56
log4j.appender.Default.File (property) 58
log4j.appender.Default.layout (property) 55
log4j.appender.Default.layout.ConversionPattern
  (property) 55
log4j.appender.Default.LogID (property) 59
log4j.appender.Default.MaxBackupIndex
  (property) 57
log4j.appender.Default.MaxFileSize (property) 57
log4j.appender.Default.SyslogHost (property) . 56
log4j.rootLogger (property) 59
Logging Plugin
  Event Log 58
  File 58
  Log Service 58
  Rolling File 58
  Syslog 58
LoginModule 126, 131

M

Manageable Resources 77
manager (script) 8, 12
manager.bat 7, 12
Mandatory
  Properties 26
Mapping Principals 126
Max Active Clients (property) 79
Max Packet Size (property) 78
MaxActiveClients (property) 79
MaxPacketSize (property) 79
Memory Profiler 31
Multiple Object Identity 95

N

Name (property) 113
New UUID 30
No System Exit (property) 115
Node.dtd 160
Nodes 16, 155
  Adding 19

Deleting 21
-noorb (Command Line Switch) 12
Notify Log ID (property) 59
Number of active CORBA objects (property) . . 61
Number of purged CORBA objects (property) . 61

O

object
creation 92
creation flags 94
deactivation 95
destruction 95
existence 96
identity 94
implementations 96
information 91
persistent state 97
reactvation 96
references 96
stringification 91
Object Browser 34
Object Cache Maximum Size (property) 63
Object Cache Minimum size (property) 64
Object Cache Purge Interval (property) 64
Object Hierarchy 14, 36, 153
Extending 19
Icons 15
Object Purging (property) 62
ObjectAdapter class 92
implementing an interface 100
initialization 92
multiple object identity 95
object creation 92
object deactivation 95
object destruction 95
object existence 96
object identity 94
object implementation 96
object persistent state 97
object reactivation 96
object references 96
ObjectRegistry.Interval (property) 64
ObjectRegistry.MaxSize (property) 63
ObjectRegistry.MinSize (property) 64
ObjectRegistry.Objects (property) 61
ObjectRegistry.Purge (property) 63
ObjectRegistry.Purges (property) 61
OF.Security.Password (system property) 137
OF.Security.UserName (system property) 137
OF_Admin_URL (property) 39
OF_DOMAIN_URL 13
OF_DOMAINS_URL 13
OF_DOMAINS_URL (property) 39
OF_NODE_URL 13
OpenFusion Graphical Tools 11
Starting 12
OpenFusion Install URL (property) 40, 41, 73
OpenFusion Java IDL Compilation 104
OpenFusion.Manager 72
OpenFusionInstallURL (property) 73
Oracle 107, 109
ORB
Daemons 7
initialization 90
shutdown 90
ORB Initialization Arguments (property) 64
Orb.Name (property) 64
ORBAdapter class 90
object information 91
object stringification 91
ORB initialization 90
ORB shutdown 90
Recommendations 98
restrictions 98
ORBInitRef.ImplementationRepository (property) 38

P

PAM 123, 131
Pattern Layout 163

Persistence Properties 49
persistent servers 101
Persistent Storage 107
PID (property) 65
Pluggable Authentication Modules . 123, 126, 131
POA Name (property) 64, 72
POA.Name (property) 65, 72
-port (Command Line Switch) 12
Port (property) 65, 73, 78, 115
Portability classes 89
Pre-load Properties (property) 74
PreLoadProperties (property) 74
Principal 124, 145
  Mapping 126, 130
Principal Persistence File
  Example 131
principalMappings (property) 137
Principals Panel 145
ProcessSingleton Configuration 69
Properties 25
  Administration Manager 71
  Assign Value Globally 30
  Assign Value to Peers 29
  Assigning Values 27
  Conditional 27, 160
  Java Properties 66
  Locking 22
  Locking Nodes 23
  Mandatory 26
  New UUID 30
  Persistence 49
  Refresh 29
  Reset Counter 29
  Security 66
  Set 29
  System 68
  Type 25
  XML File 159, 168
Properties.dtd 161

R

Read-only Community (property) 81
ReadOnlyCommunity (property) 81
Read-write Community (property) 82
ReadWriteCommunity (property) 82
Refresh 29
  current node 33
  selected browser 33
Reload Security Configuration (signal) 128
-remote (Command Line Switch) 12
Remote OpenFusion Install URL (property) 40, 41, 74
Remote Singletons 39
Remove a Principal from a Class or Method . 147
Remove All Principals from a Class or Method . . 148
Reset Counter 29
Resolve Name (property) 71
ResolveName (property) 71
Restoring Services and Singletons 24
Rolling File option 58
Root node 16
run script
  command line format 104
  using 103
Running OpenFusion 7
RunOpenFusionInstallURL (property) 74
RunViaWebServer (property) 73

S

Save
  Configuration 33
Save Changes to Security Access Entries 145
Securable Objects 123
Secure Client 136
Securing an Interface or Method 127
Security
  Properties 66
  User Identity 30
Security Access Entries 144

Security Configuration 133
Security Configuration File 136
Example 139
Security Configuration File (property) 136
Security Credentials File
Example 132
Security Credentials File (property) 135
Security Enabled (property) 133
Security Hierarchy Options 144
Security Manager 141
Security Object Icons 144
Security Principals 145
security.ConfigFile (property) 136
security.Enabled (property) 134
security.UserDataFile (property) 135
security.UserDataFile (system property) 138
security.XMLACLPersistenceFile (property) . 135
security.XMLACLPersistenceFile (system property) 138
security.XMLGroupPersistenceFile (property) 134
security.XMLGroupPersistenceFile (system property) 138
security.XMLPrincipalPersistenceFile 137
security.XMLPrincipalPersistenceFile (property) . 134
securityEnabled (property) 137
server
running 98
running user defined 102
server (script) 8, 9
Server Persistent ID (property) 54
Server Port (property) 65, 73
Server Process ID (property) 65
Servers
Persistent 101
serverSideLogin (property) 138
Service 16, 156
Changing the Order 22
Log 30
service
resolving 102
Service Portability 89
Service Resolution 91
Service Timeout (property) 75
Service.dtd 161
Service.Timeout (property) 75
Services
Starting 18
Set 29
Shared File System 38
SID (property) 54
Signals 30
Silent (property) 115
Singleton.dtd 161
Singletons 16, 157
Adding 20
Changing the Order 22
Remote 39
SNMP Agent 78
SNMPAgentObject 78
SQL Scripts 108
SQL Server 107, 112
Starting
Services 8, 18
Status 17
Status.Timeout (property) 75
StatusTimeout (property) 75
Stop current action 33
Storage Write Batch Size (property) 50, 107
Storage Write Interval (property) 49, 107
Subject 124
Supplying Authorised Credentials 131
Sybase 107, 110
Syslog Facility (property) 56
Syslog Host (property) 56
Syslog option 58
System Name (property) 68
System Properties 68
System Type (property) 68
System.Name (property) 68
System.Type (property) 68

T

Timeout (property) 61, 114
Tomcat Home Directory (property) 43

Tomcat Port (property) 45
Tomcat Security Policy File (property). 45
Tomcat WAR directory (property) 44
Tomcat WAR Files (property) 44
Tomcat Web Server. 39, 42
Tomcat Work Directory (property). 43
Tomcat.Archives (property) 44
Tomcat.Context (property) 44
Tomcat.Home (property). 43
Tomcat.PolicyFile (property) 45
Tomcat.Port (property) 45
Tomcat.WorkDir (property) 44
Tool Bar. 32
CORBA Object Browser 36
Tool Bar Button
Delete selected browser. 33
Launch the file browser. 33
Refresh selected browser. 33
Refresh the current node 33
Save Configuration 33
Stop current action. 33
View the browser log 33
Tool Bar Buttons. 145
Tool Tips. 15
Trace (property) 114
Trap Hosts File (property). 80
Trap On Authentication Failure (property). . . . 81
TrapHostsFile (property). 80
TrapOnAuthenticationFailure (property) 81
Type 25

U

Unlicenced
Java Object. 16
Singleton 16
Use Xbootclasspath (property) 67
User Identity 30
User Name (property) 68
User.Name (property) 68

V

View
browser log 33

W

Web Archives 42
Web server 39

X

Xbootclasspath 67
XML
Configuration Files 117, 153
Schema. 160
Templates. 158
XML ACL Persistence (property). 135
XML Group Persistence (property) 134
XML Group Persistence File
Example. 130
XML Principal Persistence (property) 134
XML Principal Persistence File
Example. 131