# OpenFusion® CORBA Services

Version 5

# Trading Service

PRISMTECH

# OpenFusion
## CORBA Services

# Trading Service Guide

PrismTech

Part Number: OFCOR-TRDG-5

Doc Issue 15, 17 September 2012

## Copyright Notice

# Contents

# Table of Contents

# List of Figures

# List of Figures

# Preface

## About the Trading Service Guide

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

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

### Intended Audience

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

### Organisation

The *Trading Service Guide* is organised into two main sections. The first section describes OpenFusion Trading Service. This section provides

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

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

### Conventions

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

| | |
|---|---|
| ⚠ | Item of special significance or where caution needs to be taken. |
| *i* | Item contains helpful hint or special information. |
| WIN | Information applies to Windows (*e.g.* NT, 2000, XP) only. |

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

Hypertext links are shown as *blue italic underlined.*

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

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

`Courier` fonts indicate programming code and file names.

Extended code fragments are shown in shaded boxes:

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

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

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

**Arial Bold** indicates user related actions, *e.g.* **File | Save** from a menu.

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

## Contacts

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

| **USA Corporate Headquarters** | **European Head Office** |
|---|---|
| PrismTech Corporation<br>400 TradeCenter<br>Suite 5900<br>Woburn, MA<br>01801<br>USA | PrismTech Limited<br>PrismTech House<br>5th Avenue Business Park<br>Gateshead<br>NE11 0NG<br>UK |
| Tel: +1 781 569 5819 | Tel: +44 (0)191 497 9900<br>Fax: +44 (0)191 497 9901 |

| | | |
|---|---|---|
| Web: | *http://www.prismtech.com* | |
| Technical questions: | *crc@prismtech.com* | (Customer Response Center) |
| Sales enquiries: | *sales@prismtech.com* | |

# Introduction

The *OpenFusion Trading Service* is one of a range of services and interfaces included with the *OpenFusion CORBA Services* product.

The *OpenFusion Trading Service* can be used stand-alone or with other OpenFusion CORBA Services' interfaces and services.

*OpenFusion Trading Service* is standards based, i.e. fully compliant with recognised industry standards and specifications, and supports portability and interoperability.

# Trading Service

CHAPTER

# 1 Description

*The OpenFusion Trading Service provides a mechanism through which objects can advertise their capabilities, or match their needs against advertised capabilities.*

*Just as the Naming Service is analogous to an ordinary "white pages" telephone directory, where services are found under a business name, the Trading Service is analogous to a "yellow pages" directory where services are found under the type of business that provides the service. The Trading Service is sometimes used in conjunction with the Naming Service; together they are referred to as Directory Services.*

*The Trading Service can be used in many situations where different services or products may be required by a variety of clients or customers. It could be used to manage system resources, such as printers: printers advertise their capabilities (speed, colour, multiple paper sizes), and users and applications choose a printer suitable for a specific printing job. It could be used for routing telephone calls: long-distance carriers advertise destinations and costs, and the user's local system chooses the lowest-cost carrier for the specific destination of each call. It might be used as the basis for a company share-dealing service, where buyer's wants are matched against offers of shares for sale. The Trading Service could also be used to implement a video-on-demand service, where viewers (clients) choose from movies or other programmes offered by different TV companies.*

## 1.1 OMG Standard Features

The OMG Trading Service specification defines six kinds of trading service, which provide different levels of functionality depending on which interfaces they implement. The OpenFusion Trading Service is an implementation of the *full service* specification, which provides the greatest functionality.

The OpenFusion Trading Service is wholly compliant with the OMG specification. The main features of the OMG specification enable:

- server objects to advertise the availability of their capabilities (*exports*)
- client objects to find and use servers with specific capabilities (*imports*)
- server objects to be linked together so that all their individual capabilites are available to all of their clients (*federation*)

## 1.2 OpenFusion Enhancements

The OpenFusion implementation of the Trading Service includes several enhancements to the OMG-defined Trading Service. These enhancements do not affect the use of the standard service.

Enhancements include:

- multi-threaded implementation for performance and scalability
- database persistence
- service type definition by importing XML documents
- service offer export and import as XML documents
- improved offer searching algorithm
- instrumentation (monitoring functions)

The OpenFusion Trading Service includes a graphical tool, the Administration Manager, which eases the task of managing the service whilst it is running. The Administration Manager is fully described in the *System Guide*.

## 1.3 Concepts

The OMG specification defines six kinds of Trading Service. These range from simple query services (providing lookups only) to *full service* implementations, which provide additional facilities such as administration services. The OpenFusion Trading Service is a full-service implementation, so everything in this guide is described from that point of view: there is no indication of which facilities or features may or may not be available in the other kinds of Trading Service. (The different kinds of Trading Service and the functions they provide are listed in *Supplemental Information* on page 39).

### 1.3.1 Standard CORBA Concepts

An object that supports the Trading Service is called a *trader*. A trader can be a server, a client, or both.

The following typical sequence of events illustrates both the terminology of the Trading Service and the way it could be used in a simple share-dealing system.

- A server adds a description of a service to the trader as a *service type*. This description is added to the Service Type Repository, which holds the catalogue of available service types. In a share-dealing service, these service types would probably be the names of the companies whose shares are being bought and sold.
- The server *exports* an *offer* of a service of a particular service type to the trader, and the offer is *registered*. The offer has a reference to an object providing the service (it is not necessary for this object to be the same as the one advertising the service) plus some properties that describe that object. In a share-dealing service,

an offer would be for shares in a particular company, and the properties would be the kind of share (Ordinary or Preference, for example), the number of shares available, and the price expected. The properties of an offer can be *static* or *dynamic*: the value of a static property is determined when the offer is exported (the value is held in the trader), but the value of a dynamic property may change during the life of the offer, so the value is requested from the server when a query is received. In the share offer example, the kind of share would be static but the price and quantity available would both be dynamic.

- A client queries the trader for a service with certain characteristics. For example, the client might be looking for shares in a particular company, or it might be looking for any kind of share being offered within a price range. The trader returns offers which match the requirements, and the client *imports* the offers returned by the trader. These offers contain references to server objects with the needed properties. If an offer has dynamic properties, then they are evaluated when a query is received and the offer is only returned if they match the query criteria after the current value has been provided by the server.
- If more than one offer is returned, the client selects one (or more) according to its requirements. If several offers of shares in the same company are returned, then the client will probably select the one which has the lowest price.
- The client then interacts directly with the selected server to invoke objects which implement the required service (the client agrees to purchase a number of the offered shares at the price specified).
- Depending on the kind of service it is providing, once it starts interacting with a client the server may withdraw its offer from the trader. In the example share dealing system, the server would withdraw its offer when all of the shares offered had been sold.

Note that the Trading Service is no longer involved when the interaction between the server and the client starts; it only *locates* objects, it does not do *trading* in the sense of mediating transactions between servers and clients.

The same sequence would occur in the simplified share dealing system if the offers made by servers were offers to *buy* shares; in this case the client (with shares to sell) would select the offer with the *highest* bid (price).

There can be variations in detail in different situations, but this demonstrates the general principles illustrated in Figure 1, *Trading Service Interactions*.

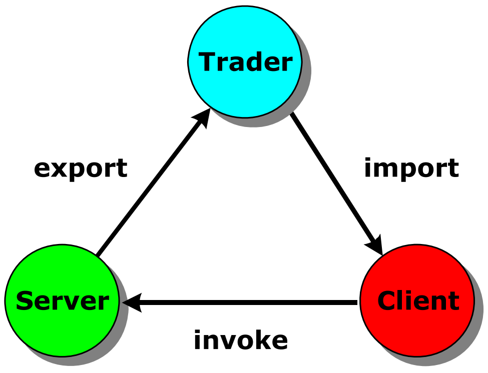


**Figure 1  Trading Service Interactions**

#### *1.3.1.1* Linking Traders

A trader can be explicitly linked to other traders. This allows clients to not only interact with a single trader, but also retrieve service offers from other traders linked to the first one. Since the linked traders may in turn be linked to more traders, a large number of traders may be reached from a given starting trader. Such a network of linked traders is known as a *federation of traders* or *trading graph*. Because each link is unidirectional, leading *from* the trader on which it is created, the trading graph is termed *directed*. There are no restrictions on the number of links that can be created from any one trader.

It is possible to specify *scoping* or *link-following policies*, to control the extent of a search for matching offers. This is often desirable in widely-distributed systems to help limit the length of time that a query takes to complete, or of the resources that are involved in its completion, but it can be for other reasons. For example, it is common to specify that a link should only be followed if no matching offers are found within the current trader. This policy may be appropriate in a printer management service, where a trader handles printers and users in a particular location: users do not want to have to go far to collect a printout, so the service should only look for a suitable printer farther afield if the print job cannot be handled by a local device.

Whatever the policies applied to individual traders and links, overall system policies can provide absolute limits: for example, although it might be possible in a large system to follow a very long daisychain of links from trader to trader, a low limit might be imposed on the number of steps (often referred to as *hops*) to follow along any one chain of links from any one trader. This *hop count* limit can also help to ensure that a search does not enter an infinite loop if it returns to the initiating trader via a link from another trader in the chain.

## 1.4 Architecture

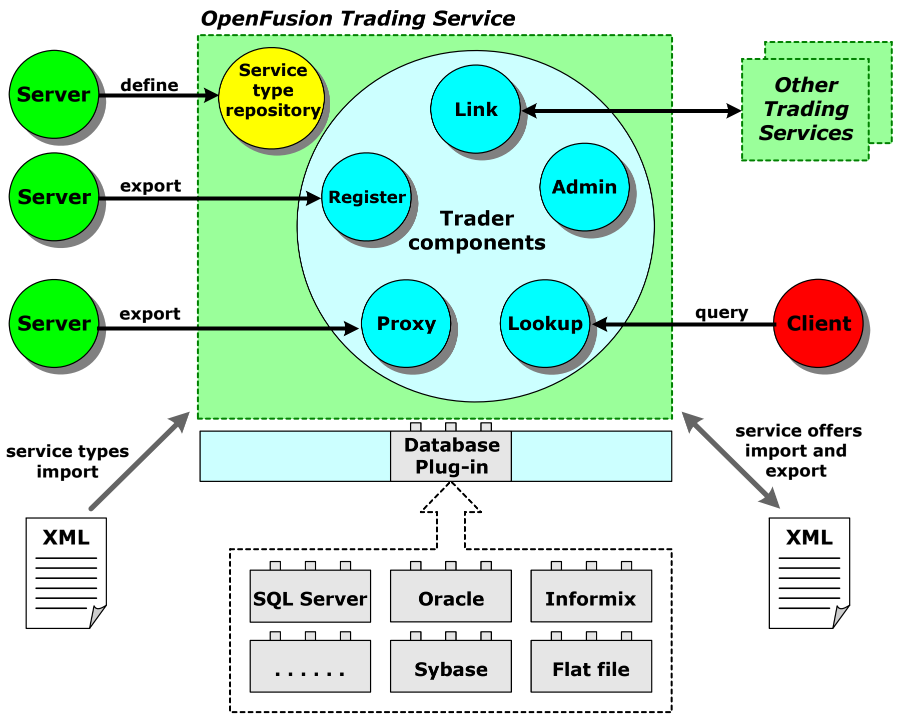


**Figure 2 Structure of the Trading Service**

The OpenFusion Trading Service consists of a number of component parts, which are illustrated in *Figure 2*. Each component of the service and of a trader is described below.

### 1.4.0.1 Service Type Repository

The Service Type Repository contains details of service types, as a kind of catalogue. Each service type definition has a unique name and zero or more properties. An appropriate service type must exist in the Service Type Repository before a server can export an offer.

### 1.4.0.2 Register

The `Register` is used by servers to export ("advertise") service offers.

To export an offer, a server must provide a description of the service and a reference to an interface where that service can be obtained.

### 1.4.0.3 Proxy

The `Proxy` supports the delayed evaluation of offers and can be used to encapsulate legacy systems.

A proxy offer is like a normal service offer in that it has a service type and named properties. However, it does not contain an object reference leading directly to the interface providing the service; it contains a reference to an object supporting the `Lookup` interface (the Trader performs a secondary lookup on this interface, transparently to the client).

### 1.4.0.4 Lookup

`Lookup` is used by the client to query the trader for service offers.

A query specifies criteria for the service type and properties which the client requires.

### 1.4.0.5 Link

The `Link` interface is used to federate traders.

Note that links are one-way only: a trader only has knowledge of links *from* itself to other traders. There is no limit to the number of links any one trader can have; however, the way those links are used can be controlled by a hierarchy of link-following policies. These policies determine when a link is used (always, sometimes, never) and how far along a chain of links any search can go.

### 1.4.0.6 Admin

The `Admin` interface is used to query and change the administrative properties of the trader. For example, link-following policies may be relaxed (within limits set for the Trading Service as a whole) or tightened, or support for proxy offers may be enabled or disabled.

### 1.4.0.7 Database Plug-in

Persistence in the Trading Service is normally implemented with a database. The OpenFusion Trading Service supports many different databases through the use of JDBC plug-ins. Please refer to the *System Guide* for information on using JDBC databases for persistence in the Trading Service.

### 1.4.0.8 XML Import and Export

The OpenFusion Trading Service enables offers to be both exported and imported as XML documents. Service type definitions can also be imported into the service type repository as XML documents.

## 1.4.1 Service Types

A *service type* represents the information needed to describe a service. The service type has a name  This is usually meaningful within the the context of the business where the service will be used; in a video-on-demand system, for example, service types would probably have names such as `programme`, `movie` and `sports_event`.  A service type definition contains:

- an *interface name*
- zero or more *named property types*
- zero or more *super-types*

The service type model can be illustrated using this notation:

```
service <ServiceTypeName>[:<BaseServiceTypeName>[,<BaseServiceTypeName>]*]
(
   interface <InterfaceTypeName>;
   [[mandatory] [readonly] property <IDLType> <PropertyName>;]*
);
```

### 1.4.1.1 Interface Names

The interface name is the IDL repository identifier for the service; in a simple video-on-demand system, the interface name for a service type called `programme` might be:

```
  interface
IDL:prismt.com.cos.CosTrading.examples/Video/Programme:1.0;
```

### 1.4.1.2 Property Types

Property types describe or define aspects of the service's behaviour.

The property types of a service contain a *name*, *type* and *mode*.

The name is usually a meaningful identifier; in the video-on-demand system mentioned above, a service such as `movie` would probably have properties named `title`, `certificate` and `price`.

The type can be any arbitrarily complex IDL type. Note, however, that only *simple* types (boolean, short, unsigned short, long, unsigned long, float, double, char, Ichar, string and Istring) can be handled by the Trader Constraint Language which is used for evaluating and selecting offers for a query to return.

The mode determines whether the property is static (fixed at the time the offer is exported) or dynamic (capable of being updated after the offer has been exported). There are four possible modes:

- normal or default: if no mode is specified, then the property is optional and if a value is given it can be updated
- readonly: if the mode is `readonly`, then the property is optional but if a value is given it cannot be updated

- mandatory: if the mode is `mandatory`, then a value for the property must be given but it can be updated
- mandatory readonly: if the mode is `mandatory readonly`, then a value for the property must be given and cannot be updated

The mode of a property type has strength. The normal or default mode is the "weakest"; the `mandatory` and `readonly` modes are of equivalent strength, and the combined `mandatory readonly` mode is strongest. This is relevant when service type properties are inherited.

#### 1.4.1.3 Super-types and Inheritance

A new service type can be declared to be a *sub-type* of an existing service type (the *super-type*), so that it inherits the characteristics of the super-type. The use of super-types enables the creation of hierarchies of service types which reflect the inheritance of interface types and the aggregation of property types.

For example, in a video-on-demand system, `film` and `sports event` would be defined as sub-types of the super-type `programme`. They would both inhherit the property `title` but would have their own properties in addition: `certificate` for `film` and `sport` for `sports event`.

This is illustrated below:

```
service programme
{
   interface
IDL:prismt.com/cos/CosTrading/examples/Video/Programme:1.0;
   property string title;
};

service film : programme
{
   interface
IDL:prismt.com/cos/CosTrading/examples/Video/Film:1.0;
   property string certificate;
};

service sportsEvent : programme
{
   interface
IDL:prismt.com/cos/CosTrading/examples/Video/SportsEvent:1.0;
   property string sport;
};
```

The rules for inheritance are:

- the interface type of the sub-type must be the same as the interface type of the super-type, or derived from it
- all of the properties defined for the super-type are implicitly defined for the sub-type

- an inherited sub-type property has the same value type as the super-type property
- an inherited sub-type property must have a mode the same as or stronger than that of the super-type

#### 1.4.1.4 Incarnation Numbers

When a new service type is successfully created, it is given an *incarnation number*. This is incremented each time a new service type is added to the repository; it is a sequence number, *not* a version number.

An incarnation number can be used with the `list_types` operation to find all service types added to the repository since that number was assigned.

The incarnation number is currently defined as a `struct` consisting of two unsigned `longs`, but it is expected that this will change when all CORBA systems fully support 64-bit integers.

### 1.4.2 Offers

A *service offer* represents the information given by a server about the service it is advertising. This information is:

- the service type name
- an object reference to an interface providing the service
- zero or more property values for the service

A server must specify values for all mandatory properties of the service type. Properties in a service offer can be modified when their property mode is not `readonly`.

Service offers may also contain dynamic properties whose values are not held within the trader but are obtained on demand using the interface of a dynamic property evaluator nominated by the server. This is useful in systems such as the share dealing service mentioned earlier, where the price at which shares are offered for sale is likely to fluctuate during the lifetime of the offer.

It is also possible to specify offer properties which are additional to those specified in the service type definition in the service type repository. Property type checking is not performed for such offer properties.

Each offer has an ID which is used by the exporter to identify it to the trader; it only has meaning within the trader with which the offer is registered.

### 1.4.3 Persistence

The OpenFusion Trading Service can store its persistent data in memory or databases.

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

The persistence mechanism must be configured before the Trading Service is started; this is normally done with the Administration Manager. Please refer to the *System Guide* for information on using JDBC databases with the Trading Service.

### *1.4.4* Instrumentation

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

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

### *1.4.5* Fail-over

Fail-over is the ability of the OpenFusion Trading Service to continue to provide a service even though part of the system has failed; another part of the system takes over from the failed part. This is currently only available with the Visibroker `osagent` daemon, which performs the fail-over transparently to the Trading Service.

All OpenFusion Trading Service instances in a cluster offer the same service (data and response times) to the rest of the system. If a member of the cluster fails, other cluster members ensure that the levels of service are maintained. If automatic re-binding is supported by the underlying ORB, the clients of the failed cluster member are automatically and transparently reconnected to other cluster members. Note that caching must be disabled to keep the service instances in synchronization. Service offers can be exported and imported as XML DTDs to assist cluster set-up.

### *1.4.6* Replication

Replication is the duplication of data across two or more databases. The duplication and synchronisation is normally performed by the database itself, and is therefore transparent to the Trading Service. This enables two or more Trading Services to use the same data, but from physically distinct databases, thus often improving performance.

CHAPTER

# 2 Using Specific Features

*This section describes how, with illustrative examples, to use the Trading Service.*

*The Trading Service's features which are covered, in the order of presentation, are:*

*Import Statements* - requisite import statements for using the Trading Service

*Obtaining a Reference to the Trading Service* - describes how to resolve a reference to the Trading Service

*Obtaining References to Trading Service Interfaces* - describes how to resolve references to the various components of the Trading Service

*Managing Service Types* - adding, removing and listing service types

*Managing Service Offers* - exporting, listing, querying, detailing and removing service offers

*Federation of Traders* - linking traders together

*Changing Trader Attributes* - managing the behaviour of the trader

*A selection of the most useful interfaces and operations are listed in API Descriptions.*

*The exceptions raised by the OpenFusion Trading Service are listed in Supplemental Information.*

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

***Note***

CORBA system exceptions are ***not*** caught in any of the examples used in this section: the code to trap these exceptions has been omitted for clarity and brevity. These exceptions should be properly caught and handled in a working system.

## 2.1 Import Statements

The following libraries must be imported into any application using the OpenFusion Trading Service:

```
import org.omg.CosTrading.*;
import org.omg.CosTrading.AdminPackage.*;
import org.omg.CosTrading.LinkPackage.*;
import org.omg.CosTrading.LookupPackage.*;
import org.omg.CosTrading.ProxyPackage.*;
import org.omg.CosTrading.RegisterPackage.*;
import org.omg.CosTradingDynamic.*;
import org.omg.CosTradingRepos.*;
import org.omg.CosTradingRepos.ServiceTypeRepositoryPackage.*;
import com.prismt.orb.*;
```

## 2.2 Obtaining a Reference to the Trading Service

***Step 1:*** A reference to the Trading Service is obtained after configuring and starting the trader by initialising the `orb` and resolving the Trading Service using:

```
org.omg.CORBA.ORB orb = null;
org.omg.CORBA.Object object = null;

orb = ObjectAdapter.init (args);

try
{
   object = orb.resolve_initial_references (“TradingService”);
}
catch (org.omg.CORBA.ORBPackage.InvalidName ex)
{
   System.err.println (“Failed to resolve service: “ + ex);
   System.exit (1);
}
```

***Step 2:*** This reference is then cast to a CORBA object and narrowed to give a reference to one of the trader components. A reference to the `Lookup` component is obtained in this example. All traders implement the lookup interface:

```
org.omg.CORBA.Object obj = (org.omg.CORBA.Object) object;
Lookup lookup = LookupHelper.narrow (obj);
```

## 2.3 Obtaining References to Trading Service Interfaces

References are obtained to any of the other four trader components by using a reference to one of the trader components. Using a reference to the `Lookup` component is used to obtain references to all of the other components.

```
Register register = lookup.register_if ();
Admin admin = lookup.admin_if ();
Link link = lookup.link_if ();
Proxy proxy = lookup.proxy_if ();
```

A reference to the `ServiceTypeRepository` is obtained using the `type_repos` method of the `Lookup` interface and then narrowing the reference to the correct type.

```
org.omg.CORBA.Object obj2 = lookup.type_repos ();
ServiceTypeRepository repository
   = ServiceTypeRepositoryHelper.narrow (obj2);
```

## 2.4 Managing Service Types

This section provides examples of how to *add*, *list* and *remove* service types using the Service Type Repository component of the Trading Service.

### 2.4.1 Adding and Removing Service Types

The full Trading Service IDL definitions of the derived types used here are provided on the distribution CD. Adding a service type to the trader is done using the `add_type` operation of the `ServiceTypeRepository`. The `ServiceTypeRepository` is part of the `CosTradingRepos` interface.

```
IncarnationNumber add_type
(
   in CosTrading::ServiceTypeName name,
   in Identifier if_name,
   in PropStructSeq props,
   in ServiceTypeNameSeq super_types
)
raises
(
   CosTrading::IllegalServiceType,
   ServiceTypeExists,
   InterfaceTypeMismatch,
   CosTrading::IllegalPropertyName,
   CosTrading::DuplicatePropertyName,
   ValueTypeRedefinition,
   CosTrading::UnknownServiceType,
   DuplicateServiceTypeName
);
```

A service type named `programme`, representing television programmes, is added to the `ServiceTypeRepository` in order to show the use of the `add_type` operation. This service type implements the `IDL:prismt.com/cos/CosTrading/examples/Video/Programme:1.0` interface, has a single property called `title` of type `string`, and has no super types. Details of valid names for service types are given in *Valid Service Type Names* on page 47 .

```
serviceType = "programme";
interfaceType
   = "IDL:prismt.com/openfusion/examples/Video/Programme:1.0";

props = new PropStruct[1];
props[0] = new PropStruct ();
props[0].name = "title";
props[0].mode = PropertyMode.PROP_NORMAL;
props[0].value_type =
   orb.get_primitive_tc (org.omg.CORBA.TCKind.tk_string);

// this type has no supertypes
supers = new String[0];
```

The `add_type` operation is then used to add this service type to the `ServiceTypeRepository`.

```
repos.add_type (serviceType, interfaceType, props, supers);
```

This operation should be included in a try-catch block in order to check that the type is successfully added.

A subtype representing a film is added to the `ServiceTypeRepository`. A film is created as a subtype, named `film`, of the service type `programme` (previously placed in the `ServiceTypeRepository`) since it is a type of television programme.

```
serviceType = “film”;
interfaceType = “IDL:prismt.com/openfusion/examples/Video/Film:1.0”;
props = new PropStruct[1];
props[0] = new PropStruct ();
props[0].name = “certificate”;
props[0].mode = PropertyMode.PROP_NORMAL;
props[0].value_type =
   orb.get_primitive_tc (org.omg.CORBA.TCKind.tk_string);
supers = new String[1];
supers[0] = “programme”;
```

The new service type, `film`, inherits the `title` property from its supertype, `programme`. The service type, `film`, is given an additional property named `certificate` of type `string`.

```
repos.add_type (serviceType, interfaceType, props, supers);
```

The `remove_type` operation of the `ServiceTypeRepository` is used to delete a service type:

```
void remove_type
(
   in CosTrading::ServiceTypeName name
)
raises
(
   CosTrading::IllegalServiceType,
   CosTrading::UnknownServiceType,
   HasSubTypes
);
```

```
repository.remove_type (“film”);
```

Service types can also be added to the trader from text files using the Service Type Repository Manager, which is part of the Administration Manager.

### 2.4.2 Listing Service Types

Once service types have been defined, a complete list of all service types can be retrieved from the Service Type Repository using the `list_types` operation:

```
ServiceTypeNameSeq list_types (in SpecifiedServiceTypes
which_types);
```

The `SpecifiedServiceTypes` parameter gives two options enabling service types to return either a complete list of service types or all service types which were added to the trader since a given incarnation number.

```
union SpecifiedServiceTypes switch (ListOption)
{
   case since: IncarnationNumber incarnation;
};
```

The following example retrieves a complete list of all service type names:

```
SpecifiedServiceTypes whichTypes = new SpecifiedServiceTypes ();
whichTypes.__default (ListOption.all);
String [] types = repository.list_types (whichTypes);
```

## 2.5 Managing Service Offers

This section provides examples of managing service offers in the Trading Service. Both ordinary and proxy offers are exported and removed. Listing offers from the trader and describing offers are also covered, as is the use of dynamic properties within offers.

### 2.5.1 Exporting Service Offers

A server advertises its service offers using the export operation of the `Register` interface.

```
OfferId export
(
   in Object reference,
   in ServiceTypeName type,
   in PropertySeq properties
)
raises
(
   InvalidObjectRef,
   IllegalServiceType,
   UnknownServiceType,
   InterfaceTypeMismatch,
   IllegalPropertyName,
   PropertyTypeMismatch,
   ReadonlyDynamicProperty,
   MissingMandatoryProperty,
   DuplicatePropertyName
);
```

The ServiceTypeName and PropertySeq variables are set up when the variable reference already contains an object of type org.omg.CORBA.Object. An offer is set up here with type film (used in *Adding and Removing Service Types* on page 19); values are set for the two properties title and certificate. The any variable is of type org.omg.CORBA.Any.

```
String type = "film";

Property [] property = new Property[2];
property[0] = new Property ();
property[0].name = "title";
any = orb.create_any ();
any.insert_string ("Star Wars");
property[0].value = any;
property[1] = new Property ();
property[1].name = "certificate";
any = orb.create_any ();
any.insert_string ("U");
property[1].value = any;
```

The code to export the service offer is then simply:

```
offerId = register.export (reference, type, property);
```

The OfferIds given to offers in the OpenFusion Trading Service are strings representing UUIDs. These are fixed length strings.

```
com.prismt.util.UUID UUID = null;
try
{
   UUID = new com.prismt.util.UUID (offerId);
}
catch (com.prismt.util.UUIDArgumentException ex)
{
   System.err.println ("Invalid offer ID: " + ex);
}

boolean res = UUID.equals (otherUUID);
```

### 2.5.2 Using Dynamic Properties

Offers exported to the trader may contain properties that are name-value pairs where the name is a string and the value is of type CORBA any. The example above used a static property named title with its value set to Star Wars by inserting that string into an any. It is also possible to export offers with dynamic properties where the actual value of the property is held externally and can be obtained on demand.

The next code example shows how a property value can contain a DynamicProp structure rather than a static property value. The value of the date property will not be set until the property value is required during the execution of a query.

```
property[0] = new Property ();
property[0].name = "date";
any = orb.create_any();
MyProperty mp = new MyProperty (orb);
DynamicProp myProp = new DynamicProp
```

```
(
   DynamicPropEvalHelper.narrow (ObjectAdapter.getObject (mp)),
   orb.get_primitive_tc (org.omg.CORBA.TCKind.tk_string),
   orb.create_any ()
);
DynamicPropHelper.insert (any, myProp);
property[0].value = any;
```

The class `MyProperty`, which provides the `DynamicPropEval` interface, shows how a properly typed property value is obtained during the evaluation of a query. The property value is set to the current date represented as a string when the `evalDP` operation is called on behalf of a property with a `string` typecode. However, the property value is set to the current date represented as the number of milliseconds since January 1st 1970 when the `evalDP` operation is called on behalf of a property with a `long` typecode. The third parameter of the `DynamicProp`, `extra_info`, is not used by the Trading Service but is passed onto the `evalDP` operation when the dynamic property value is evaluated.

```
public class MyProperty
   implements org.omg.CosTradingDynamic.DynamicPropEvalOperations
{
   private org.omg.CORBA.ORB orb = null;

   public MyProperty (org.omg.CORBA.ORB orb)
   {
      ObjectAdapter.createTransient (this);
      this.orb = orb;
   }

   public org.omg.CORBA.Any evalDP
   (
      String name,
      org.omg.CORBA.TypeCode tc,
      org.omg.CORBA.Any extraInfo
   )
      throws org.omg.CosTradingDynamic.DPEvalFailure
   {
      org.omg.CORBA.Any value = orb.create_any ();
      org.omg.CORBA.TypeCode tc2 =
         orb.get_primitive_tc (org.omg.CORBA.TCKind.tk_string);

      if (tc.equal (orb.get_primitive_tc
(org.omg.CORBA.TCKind.tk_string)))
      {
         value.insert_string ((new java.util.Date ()).toString ());
      }
      else
      if (tc.equal (orb.get_primitive_tc
(org.omg.CORBA.TCKind.tk_longlong)))
      {
         value.insert_longlong ((new java.util.Date ()).getTime ());
      }
      else
      {
         throw new org.omg.CosTradingDynamic.DPEvalFailure ();
      }
```

```
        return value;
    }
}
```

`Readonly` properties may not have dynamic values. No offers containing dynamic properties are matched when the trader does not support dynamic properties, either because the `supports_dynamic_properties` attribute is set to `false`, or a query requests that dynamic properties not be used.

### 2.5.3 Exporting Proxy Offers

In *Exporting Service Offers* above, the interface that provides the service is an integral part of the service offer. As illustrated in Figure 3, *Proxy Offer Indirection* below, the trading service also supports a second kind of service offer, a *proxy offer*, where the interface that provides the offer is determined at run-time. The object reference is obtained by invoking another query operation on the target `Lookup` interface held in the proxy offer when a proxy offer matches a query.

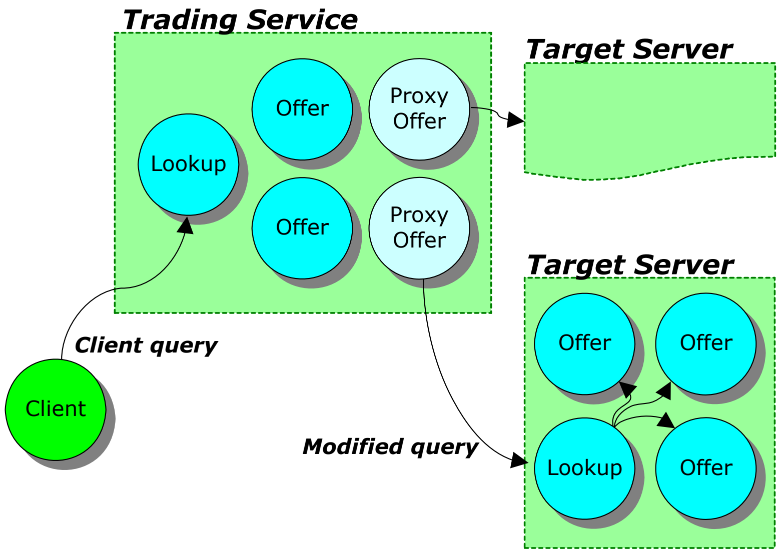


**Figure 3 Proxy Offer Indirection**

Proxy offers are useful, for example, in systems where objects are created dynamically on request as part of a query. This can be a convenient way to encapsulate legacy objects; the query to the proxy trader permits the dynamic creation of a service instance that encapsulates the legacy object.

The target interface of a proxy offer must syntactically support the `Lookup` interface of the trader, but all that is required of the target interface is that the query operation return zero or one offer using the `offers` parameter.

```
public void query
(
    String type,
    String constr,
    String pref,
    org.omg.CosTrading.Policy [] policies,
    SpecifiedProps desired_props,
    int how_many,
    OfferSeqHolder offers,
    OfferIteratorHolder offer_iter,
    PolicyNameSeqHolder limits_applied
)
   throws IllegalServiceType,
          UnknownServiceType,
          IllegalConstraint,
          IllegalPreference,
          IllegalPolicyName,
          PolicyTypeMismatch,
          InvalidPolicyValue,
          IllegalPropertyName,
          DuplicatePropertyName,
          DuplicatePolicyName
{
   int index1 = -1;
   int index2 = -1;
   System.out.println ("Programme Factory passed constraint: " +
constr);
   index1 = constr.indexOf ("title==");
   index1 = index1 + 7; // move to the first quote mark
   if (index1 >= 0)
   {
      // "title==" appears in constraint
     index2 = constr.indexOf ("'", index1 + 7); // find second quote
mark
      title = constr.substring (index1 + 1, index2);

      offers.value = new Offer[1];
      offers.value[0] = new Offer ();
      offers.value[0].properties = new Property[0];

      ProgrammeOperations pops = new ProgrammeOperations ()
      {
         public String title ()
         {
            return title;
         }

         public String description ()
         {
            return "Unknown";
         }

         public void show ()
         {
            System.out.println
            ("Showing from the programme factory: " + title);
         }
      };
      offers.value[0].reference = ObjectAdapter.createTransient
(pops);
```

```
      limits_applied.value = new String[0];
   }
   else
   {
      // we can’t create the programme
      offers.value = new Offer[0];
      limits_applied.value = new String[0];
      System.out.println
         (“Programme Factory unable to create a film using the
constraint: ”
          + constr);
   }

}
```

In addition to the query operation, the target interface must support all other operations of the `Lookup` interface but none of these will be used by the trading service.

Proxy offers are exported to the trading service using the `export_proxy` operation of the Proxy interface.

```
OfferId export_proxy
(
   in Lookup target,
   in ServiceTypeName type,
   in PropertySeq properties,
   in boolean if_match_all,
   in ConstraintRecipe recipe,
   in PolicySeq policies_to_pass_on
)
raises
(
   IllegalServiceType,
   UnknownServiceType,
   InvalidLookupRef,
   IllegalPropertyName,
   PropertyTypeMismatch,
   ReadonlyDynamicProperty,
   MissingMandatoryProperty,
   IllegalRecipe,
   DuplicatePropertyName,
   DuplicatePolicyName
);
```

A proxy offer contains a service type and properties in the same way as an ordinary service offer. The proxy offer is considered to be a match to a query on service type conformance alone when the `if_match_all` property is set to `true`. This means that the query’s constraint expression is not evaluated against the properties contained in the proxy offer.

The `recipe` parameter of a proxy offer is an instruction to the trader describing how to construct the constraint expression for the secondary query to the target interface from the primary constraint expression and the proxy offer's properties. Details of the recipe language are provided in *Supplemental Information*.

The `policies_to_pass_on` parameter contains any policies that should be appended to those given in the primary query when passing the query onto the target interface. These policy values are only seen by the target interface so duplicated values will not necessarily throw `DuplicatePolicyName` exceptions. Instead, they could be used, for example, to override the values given by the primary query operation.

### 2.5.4 Listing All Service Offers

A complete list of all service offers can be retrieved from the trader. This is done using the `list_offers` operation of the `CosTrading` module which allows the specification of the number of offers to be retrieved in the first instance and the return of an iterator to obtain further service offers.

```
void list_offers
(
   in unsigned long how_many,
   out OfferIdSeq ids,
   out OfferIdIterator id_itr
)
raises
(
   NotImplemented
);
```

The retrieval of all service offers is demonstrated by first retrieving ten of them, and then using the iterator to retrieve further sets of ten until all offers have been retrieved. First, initialise the `Holder` variables:

```
OfferIdSeqHolder ids = new OfferIdSeqHolder ();
OfferIdIteratorHolder iter = new OfferIdIteratorHolder ();
```

The `list_offers` operation is used to retrieve the first ten offers:

```
admin.list_offers (10, ids, iter);
```

These ten offers are stored in a vector. The correct number of offers is stored in the vector when the trader contains less than ten offers.

```
java.util.Vector v = new java.util.Vector ();
for (int i = 0; i < ids.value.length; i++)
{
   v.addElement (ids.value[i]);
}
```

The iterator returned by the `list_offers` operation is now used to retrieve the rest of the offers from the trader in sets of ten until there are no more offers returned by the iterator. All these offers are added to the storage vector.

```
if (iter.value != null)
{
   OfferIdSeqHolder seq = new OfferIdSeqHolder ();
   boolean more;
   do
   {
      more = iter.value.next_n (10, seq);
      for (int i = 0; i < seq.value.length; i++)
      {
         v.addElement (seq.value[i]);
      }
   }
   while (more);

   iter.value.destroy ();
}
```

The vector `v` containing all of the offers that have been found in the trader can now be transformed into a string array containing the offers.

```
String [] offers = new String [ v.size()];
for (int i = 0; i < v.size(); i++)
{
   offers[i] = (String) v.elementAt(i);
}
```

### 2.5.5 Querying Service Offers

The user can query the trader to find service offers that have certain characteristics or properties. This is done using the query operation of the `CosTrading` module:

```
void query
(
   in ServiceTypeName type,
   in Constraint constr,
   in Preference pref,
   in PolicySeq policies,
   in SpecifiedProps desired_props,
   in unsigned long how_many,
   out OfferSeq offers,
   out OfferIterator offer_iter,
   out PolicyNameSeq limits_applied
)
raises
(
   IllegalServiceType,
   UnknownServiceType,
   IllegalConstraint,
   IllegalPreference,
   IllegalPolicyName,
   PolicyTypeMismatch,
   InvalidPolicyValue,
   IllegalPropertyName,
   DuplicatePropertyName,
   DuplicatePolicyName
);
```

A simple example of querying the trader is now worked through. An offer for the service type `film` is looked for, although any constraints for the offers returned are not specified, but requested to be returned in random order.

```
String stype = "film";
String constraints = "";
String preferences = "random";
```

The `policies` parameter is used to override the trader's default behaviour. One of the trader's policies is changed to ensure that no proxy offers are considered by setting the policy `supports_proxy_offers` to `false`. The variable `any` has already been defined to have the type `org.omg.CORBA.Any`.

```
Policy [] policies = new Policy[1];

policies [0] = new Policy();
policies[0].name = "supports_proxy_offers";
any = orb.create_any();
any.insert_boolean(false);
policies[0].value = any;
```

All the properties that should be returned, along with the offers using the `SpecifiedProps` parameter are specified. In this case, three offers are to be returned in the first instance.

```
SpecifiedProps properties = new SpecifiedProps ();
properties.__default (HowManyProps.all);

int how_many = 3;
```

The `Holder` objects are set up for the returned parameters. Any matching offers are returned in the `OfferSeqHolder` object. The `OfferIteratorHolder` contains the iterator that enables any other remaining offers to be retrieved.

```
OfferIteratorHolder offer_itr = new OfferIteratorHolder ();
OfferSeqHolder offerseq = new OfferSeqHolder ();
PolicyNameSeqHolder limits = new PolicyNameSeqHolder ();
```

The call to the query operation is then:

```
lookup.query
(
   stype,
   constraints,
   preferences,
   policies,
   properties,
   how_many,
   offerseq,
   offer_itr,
   limits
);
```

### 2.5.6 Details of Service Offers

References to services offers are obtained using either the query or list-offers methods above. Each offer is returned as an `OfferId` which is a String when all service offers are listed. Each offer is returned as an `Offer` object which contains a reference to the object and a list of the properties associated with the offer when the trader is queried.

It is possible to obtain the information contained within an `Offer` object when all service offers have been listed, and there is an `OfferId` object, by using the describe operation of the `Register` interface.

```
OfferInfo describe
(
   in OfferId id
)
raises
(
   IllegalOfferId,
   UnknownOfferId,
   ProxyOfferId
);
```

Therefore, using either method above, the reference and properties of the offer are obtained. The service type will be either specified in the query that returned the offer, or returned by the `describe` operation.

The reference is in the form of an `IOR`. The following Java code prints out the IOR of the reference when the variable `reference` is of type `org.omg.CORBA.Object`.

```
System.out.println (orb.object_to_string (reference));
```

The details of all the properties it contains are printed when the variable props is of type `org.omg.CosTrading.Property[]` using the following code:

```
for (int i = 0; i < props.length; i++)
{
   System.out.println ("Property Name: " + props[i].name);
   System.out.println ("Property Value: " + props[i].value);
}
```

This code does not deal with dynamic properties. However, the following code could be used in conjunction with that above to deal with dynamic properties. It first checks that the property has the typecode of a dynamic property, and then extracts the property details, providing that it is a dynamic property.

```
org.omg.CORBA.TypeCode tc = props[i].value.type ();

if (tc.equal (DynamicPropHelper.type () ) )
{
   DynamicProp dp = DynamicPropHelper.extract(props[i].value);

   System.out.println
```

PrismTech

```
        ("eval_if = " + orb.object_to_string (dp.eval_if) );
    System.out.println
        ("returned_type = " + dp.returned_type );
    System.out.println
        ("extra_info = " + dp.extra_info);
}
```

### 2.5.7 Removing Service Offers

Service offers remain in the trader until they are explicitly withdrawn. The offer is useless to a client when the object reference contained in an offer is no longer valid. It is therefore good practice for a server to withdraw each of its service offers when they become invalid. This is done using the `withdraw` operation of the `Register` interface:

```
void withdraw
(
    in OfferId id
)
raises
(
    IllegalOfferId,
    UnknownOfferId,
    ProxyOfferId
);
```

This operation is very simple to use: when `offerid` is an object of type `OfferId` which is a `String`, the Java code for withdrawing the offer is:

```
register.withdraw (offerid);
```

## 2.6 Federation of Traders

The operations of the `Link` interface allow traders to be linked together so that a trader can make the offer spaces of other trading services available to its own clients. A *link* describes the knowledge that the trader has of a target trader. In order to link to a target trader, a reference to the `Lookup` interface of the target trader is needed, and a name for the target trader is given. Policies determining when the link will be followed also need to be defined. The first policy determines the default behaviour for following the link and is used when a client does not specify a particular mode of link-following behaviour. The second policy determines the most permissive link-following behaviour that a client may request.

The three possible modes for link-following behaviour are:

- *`FollowOption.local_only`*: links are not followed in any circumstances.
- *`FollowOption.if_no_local`*: links are followed when, and only when, a trader cannot find any offers in the local trading service.
- *`FollowOption.always`*: links are always followed.

The add_link operation of the link interface is used to add a link to the trader. Here, a link to a second trader is set up with the name Link from 1 to 2. A reference to the Lookup interface of this trader is stored in the variable lookup2. The default policy for following links is to follow them when no offers can be found locally. However, the client may request any behaviour for following links since the limiting policy value is to always follow links.

```
link.add_link
(
   “Link from 1 to 2”,
   lookup2,
   FollowOption.if_no_local,
   FollowOption.always
);
```

Information about a link is obtained using the name it has been given. The LinkInfo object returned contains details of the link-following policies of the link and a reference to the Lookup interface of the target trader; it also contains a reference to the Register interface of the target trader when the target trader implements this interface .

```
LinkInfo linkInfo = link.describe_link (“Link from 1 to 2”);
```

It is possible to modify the link following policies of a link. This is done using the modify_link operation. Here, the link is modified so that it is never followed by default, but is set by the client so that it is followed when no offers are found locally.

```
link.modify_link
(
   “Link from 1 to 2”,
   FollowOption.local_only,
   FollowOption.if_no_local
);
```

The list_links operation returns a list containing the names of all the links within the trader. This example shows how to obtain the list and then output the name of each link.

```
String [] allLinks = link.list_links ();
for (int i = 0; i < allLinks.length; i++)
{
   System.out.println (“Link “ + i + “ is called “ + allLinks[i]);
}
```

The only remaining operation in the Link interface is used to remove a link from a trader using its name.

```
link.remove_link (“Link from 1 to 2”);
```

## 2.7 Changing Trader Attributes

The following examples show how to modify the default behaviour of a trading service. The attributes that determine the behaviour of the trader are contained in the `SupportAttributes`, `ImportAttributes` and `LinkAttributes` interfaces. The operations to modify these attributes are contained in the `Admin` interface. The `Admin` interface is used to both determine the values of attributes and to change them since it also extends all three of the `Attributes` interfaces.

This example demonstrates how to obtain the attribute values, assuming the variable `admin` is a reference to the `Admin` interface.

```
System.out.println
(
   "Maximum number of offers to search in a query: "
   + admin.max_search_card ()
);
System.out.println
(
   "Trader supports modifiable properties? "
   + admin.supports_modifiable_properties ()
);
System.out.println
(
   "Most permissive link following policy: "
   + admin.max_link_follow_policy ()
);
```

Each of the attributes used above is given a new value in the next example:

```
admin.set_max_search_card (10000);
admin.set_supports_modifiable_properties (false);
admin.set_max_link_follow_policy (FollowOption.if_no_local);
```

*CHAPTER*

# 3 *API Descriptions*

*The OpenFusion Trading Service provides most of its functionality through the interfaces of the trader components: Register, Lookup, Admin, Link and Proxy. The interfaces listed here are these primary interfaces. The complete IDL API is provided elsewhere as part of the product distribution.*

## 3.1 TraderComponents Interface

The `CosTrading::TraderComponents` interface has references to the component services that make up a trader. Not all services may be present, depending on the type of trader.

For an example of the use of this interface, please refer to *Obtaining References to Trading Service Interfaces* on page 18.

This is an abstract interface, inherited by each of the components.

| Operation | Description |
|---|---|
| `admin_if` | Provides a reference to the Admin interface |
| `register_if` | Provides a reference to the Register interface |
| `lookup_if` | Provides a reference to the Lookup interface |
| `link_if` | Provides a reference to the Link interface |
| `proxy_if` | Provides a reference to the Proxy interface |

## 3.2 Admin Interface

The `CosTrading::Admin` interface is used to manage the various policies and constraints applied by the trader.

| Operation | Description |
|---|---|
| `list_offers` | Returns a list of offers made by the trader. Offer identifiers not included in the returned sequence can be accessed through the returned iterator |
| `list_proxies` | Returns a list of proxy offers made by the trader. Offer identifiers not included in the returned sequence can be accessed through the returned iterator |
| `set_def_follow_policy` | Specifies the default link-following behaviour of the trader |
| `set_def_hop_count` | Specifies the default maximum number of steps to take when following any single chain of links when executing federated queries; overridden by `max_hop_count` |
| `set_def_match_card` | Specifies the default maximum number of offers to be selected; overridden by `max_match_card` |
| `set_def_return_card` | Specifies the default maximum number of offers to be returned; overridden by `max_return_card` |
| `set_def_search_card` | Specifies the default maximum number of offers to be searched; overridden by `max_search_card` |
| `set_max_follow_policy` | Specifies the link-following policy for all links from the current trader |
| `set_max_hop_count` | Specifies the maximum number of steps to take when following any single chain of links that can be followed when executing federated queries in the current trader |
| `set_max_link_follow_ policy` | Specifies the maximum follow options that can applied to links when they are created or modified |

| Operation | Description |
|---|---|
| `set_max_list` | Specifies the maximum number of objects that can be returned in any list (including iterators) in the current trader |
| `set_max_match_card` | Specifies the maximum number of offers that can be selected by a query in the current trader |
| `set_max_return_card` | Specifies the maximum number of offers that can be returned by a query in the current trader |
| `set_max_search_card` | Specifies the maximum number of offers that can be searched by a query in the current trader |
| `set_request_id_stem` | This operation (intended to help avoid problems with cyclical references when resolving federated queries) is not supported by the OpenFusion Trading Service, as the use of UUIDs renders it superfluous |
| `set_supports_dynamic_properties` | Determines whether dynamic properties in offers are supported in the current trader |
| `set_supports_modifiable_properties` | Determines whether modification of properties in offers is supported in the current trader |
| `set_supports_proxy_offers` | Determines whether proxy offers are supported in the current trader |
| `set_type_repos` | Specifies the reference to the Service Type Repository to be used by the current trader |

## 3.3 Register Interface

The `CosTrading::Register` interface provides operations to export offers to the trader, and to manage the exported offers.

| Operation | Description |
|---|---|
| `describe` | Describes an offer of service from the trader |
| `export` | Exports an offer of service to the trader with a set of properties it supports |
| `modify` | Modifies the properties supported by a traded offer of service. Properties that are read-only or mandatory may not be deleted and properties that are read-only may not be modified |

| Operation | Description |
|---|---|
| `resolve` | Resolves the Register interface of a linked trader |
| `withdraw` | Withdraws an offer of service from the trader |
| `withdraw_using_constraint` | Withdraws any offers matching the specified constraints |

## 3.4 Lookup Interface

The `CosTrading::Lookup` interface provides one operation for finding offers which match specified requirements.

| Operation | Description |
|---|---|
| `query` | Queries the trader for offers. The various call parameters determine how the trader finds offers to satisfy the query |

## 3.5 Link Interface

The `CosTrading::Link` interface is used to federate traders and manage the links between them.

| Operation | Description |
|---|---|
| `add_link` | Links a federated trader |
| `describe_link` | Describes a link to a federated trader, returning a structure containing the link description. |
| `list_links` | Returns a sequence of links. |
| `modify_link` | Modifies a link to a federated trader, changing the follow options that are applied to the link. |
| `remove_link` | Removes a link to a federated trader. |

## 3.6 Proxy Interface

The `CosTrading::Proxy` interface provides operations to support proxy offers.

| Operation | Description |
|---|---|
| `describe_proxy` | Describes a proxy offer of service from the trader |
| `export_proxy` | Exports a proxy offer of service from the trader with a set of properties it supports |
| `withdraw_proxy` | Withdraws a proxy offer of service from the trader |

CHAPTER

# 4 Supplemental Information

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

*First there is a description of the Trader Constraint Language and examples of how to define Preferences and Constraint Recipes. Valid Service Type names are described next, and then the importing and exporting of XML files containing service types or offers, together with descriptions of the XML files themselves. This is followed by a summary of the different kinds of Trading Service with a list of their components. Finally the exceptions supported by the Trading Service are listed and described.*

## 4.1 Trader Constraint Language

Constraint languages are used to define or limit the scope of an operation by specifying one or more conditions which must be satisfied before the operation proceeds. The most common use for these conditions is in the definition of *filters* for queries or searches. These selection criteria are sometimes referred to as *policies*.

The OMG specifies a standard constraint language which ensures interoperability between traders. The OpenFusion Trader Constraint Language (TCL) implements this standard.

Other constraint languages may be supported by particular trader implementations. Statements expressed in such languages must be identified with an escape prefix of the form `<<name major.minor>>` , where `name` is the name of the language and `major.minor` is the version. Any constraint with such a prefix which is not recognised by a particular trader is simply ignored and the query operation continues as though no constraint was given.

The absence of a prefix is treated as equivalent to the presence of the `<<OMG 1.0>>` prefix, which identifies a standard TCL statement. Note that throughout this section this identity prefix is omitted for brevity.

The constraint string `TRUE` is assumed when no constraint string is given.

All keywords in TCL are case-sensitive.

Please refer to the OMG Trading Service specification for the definition of the Trader Constraint Language.

This section describes the way TCL is used to represent constraints in the `Lookup::query` operation.

### *4.1.1* Common Types of Constraints

Some examples of common types of constraint strings are:

- `name == 'Prism'`
- `numberUsers > 42`
- `'java' in topics`
- `quality > 10 and price < 2000`
- `exist name`
- `memsize * 3.6 + filesize / 10.7 > 1234`
- `not 'Prism' in names`

### *4.1.2* Property Value Types

Service types can be defined with properties having arbitrarily complex IDL value types. However, only a limited set of property value types can be manipulated using the constraint language. The simple types that can be used are:

- `boolean`
- `short`
- `unsigned short`
- `long`
- `unsigned long`
- `float`
- `double`
- `char`
- `string`

Sequences of these types can also be manipulated.

The `exist` operator can be applied to any property even when the property does not have one of these types.

### *4.1.3* Query Elements

This section details all the possible items that can be used in a query string.

#### 4.1.3.0.1 Comparative Functions

*Table 1* summarises the comparison operators that can feature in a query string.

**Table 1 Comparative Functions**

| Symbol | Description | Restrictions |
|---|---|---|
| **==** | equality | Can be applied only when both the left and right operands are of the same simple type. |
| **!=** | inequality | Can be applied only when both the left and right operands are of the same simple type. |
| **>=** | greater than or equal to | Can be applied only when both the left and right operands are of the same simple type. |
| **<** | less than | Can be applied only when both the left and right operands are of the same simple type. |
| **<=** | less than or equal to | Can be applied only when both the left and right operands are of the same simple type. |
| **~** | substring match | Can be applied only when both the left and right operands are strings. |
| **in** | element in sequence | Can be applied only when the left operand is of a simple type and the right operand is a sequence of the same simple type. |

The result of applying any of these comparative functions is a boolean value.

In an OpenFusion trader, the operations `==, !=, >, >=, <, and <=` can be used to compare any numeric type to any other numeric type. For example, `1.0 == 1` and `2e2 > 199` evaluate to `TRUE`. Comparisons between operands of different types return `FALSE`.

`<, <=, >, >=` can be used on boolean values where `TRUE` is considered to be greater than `FALSE`.

`name =='Prism'` means that the query will only return offers where the value of the property `name` is `'Prism'`.

`numberUsers > 42` means that the query will only return offers where the value of the property `numberUsers` is greater than `42`.

`'java' in topics` means that the query will only return offers where the value of the `topics` property, which consists of a set of strings, contains the string `'java'`.

`'Open' ~ product` means that the query will only return offers where the value of the `product` property contains the substring `'Open'`.

#### *4.1.3.0.2* Boolean Connectives

The following boolean connectives can be used in a query string:

- `and`
- `or`
- `not`

`not ('Prism' in names)` implies that the query should consider matching offers only when the value of the `names` property, which consists of a set of strings, does not contain the string `'Prism'`.

`quality > 10 and price < 2000` implies that the query should consider matching offers only when the value of the `quality` property is greater than `10` and the value of the `price` property is less than `2000`.

#### *4.1.3.0.3* Property Existence

The `exist` operator tests for the presence of a particular property name without having to test its value.

**Table 2 Property Existence**

| Description | Restrictions |
|---|---|
| **`exist`** | Can be applied to any property regardless of its type. |

`exist name` implies that the query should consider matching only offers which contain a property named `name`.

#### *4.1.3.0.4* Property Names

Property names feature in almost all constraints. At least one property name has been used in each expression in the above examples.

| Expression | Property Name |
|---|---|
| **`numberUsers > 42`** | numberUsers |
| **`'java' in topics`** | topics |
| **`not 'Prism' in names`** | names |
| **`quality > 10 and price < 2000`** | quality, price |
| **`exist name`** | name |

PRISMTECH

#### 4.1.3.0.5 Numeric and String Constants

Integers should be written as sequences of digits with a possible leading + or -.

Examples: `5, 27165, +45, -9`

Floats are sequences of digits with a decimal point. Exponential notation is optional.

Examples: `1.0, .6, 34., 123.45, 12E3, 1234e-05, 1.3E+03`

`char` is of the form `'<char>'` and `string` is of the form `'<char><char>+'`. To embed an apostrophe or a backslash in a string, place a backslash \ in front of it.

Examples: `'i\'m here'` evaluates to the string *i'm here*, `'c:\\temp\\'` evaluates to the string *c:\temp\*, `'c'` evaluates to the character *c*.

`TRUE` and `FALSE` represent the boolean constants.

In `quality > 10 and price < 2000` both `10` and `2000` are numeric constants. In `'java' in topics` the word *`java`* is a string constant.

#### 4.1.3.0.6 Mathematical Operators

*Table 3* summarises the mathematical operators that may be used in a query string.

**Table 3 Mathematical Operators**

| Symbol | Description | Restrictions |
|---|---|---|
| `+` | addition | Can be applied only to simple numeric operands. |
| `-` | subtraction | Can be applied only to simple numeric operands. |
| `*` | multiplication | Can be applied only to simple numeric operands. |
| `/` | division | Can be applied only to simple numeric operands. |

Negation is indicated with a prefixed hyphen `-` (minus sign).

`memsize * 3.6 + filesize / 10.7 > 1234` implies that the query should return matching offers only when the arithmetic function, in terms of the values of the `memsize` and `filesize` properties, is greater than `1234`.

#### 4.1.3.0.7 Grouping Operator

The comma `,` may be used as a grouping operator in query strings. It is treated as a synonym for `AND`; thus the expression `(2 > 1) , (3 > 4)` evaluates to `FALSE`.

Note that this interpretation of the comma operator is specific to the OpenFusion implementation of TCL; the purpose of the comma is ambiguous in the OMG Trading Service specification.

⚠ Because there is a possibility of alternative incompatible interpretations of the comma operator (it may not be implemented at all), its use in constraints used on heterogeneous systems is not recommended. It can of course be used safely on pure OpenFusion systems.

### 4.1.4 Precedence

The operators have the following precedence order in the absence of parentheses:

**Table 4 Operator Precedence**

| Operator | Order of Precedence |
|---|---|
| `( )   exist   unary-minus` | highest precedence |
| `not` | |
| `* /` | |
| `+ -` | |
| `~` | |
| `in` | |
| `== != < <= > >=` | |
| `and` | |
| `or` | lowest precedence |

Operators are evaluated from left to right when they have the same precedence. The order of evaluation can be controlled by grouping values and operators with parentheses; note, however, that implicit multiplication of bracketed expressions is not valid, so expressions such as `( unitcost + margin ) * ( taxrate + ifactor )` must always include the `*` operator.

## 4.2 Trader Preference Language

When a set of offers matching the service type, constraint and policies has been obtained with a `Lookup::query` operation, it is returned in an order determined by a *preference string*.

As with TCL, preference strings may be expressed in other languages provided they are identified with an escape prefix of the form `<<name major.minor>>`. The absence of a prefix is treated as equivalent to the use of the `<<OMG 1.0>>` prefix, which identifies a preference string which complies with the standard.

The OpenFusion Trading Service only supports the standard language.

Any preference string with such a prefix which is not recognised by a particular trader is simply ignored and the operation continues as though no preference string was present.

The preference string `first` is assumed when no preference string is present.

There are five different types of preference string, each beginning with one of five case-sensitive keywords:

- *`first`*: This is the preference that is used when no preference string is given. Matched offers are returned in the order in which they are discovered.
- *`random`*: The offers are returned in a random order.
- *`max expression`*: The `max` expression is numeric. Matched offers are returned in descending order of the expression. For example, `max numberUsers` first returns the offer with the highest value of the property `numberUsers`, and, finally, the lowest value of the property `numberUsers`. Any offers for which the expression cannot be evaluated are appended to the set of offers returned after the offer with the lowest value of the expression.
- *`min expression`*: The `min` expression is numeric. Matched offers are returned in ascending order of the expression. For example, `min numberUsers` first returns the offer with the lowest value of the property `numberUsers`, and, finally, the highest value of the property `numberUsers`. Any offers for which the expression cannot be evaluated are appended to the set of offers returned after the offer with the highest value of the expression.
- *`with expression`*: The `with` expression is a constraint expression. Matched offers are returned such that those that are `TRUE` precede those that are `FALSE`. For example, `with numberUsers > 42` returns offers where the `numberUsers` property has a value greater than `42` before those where the `numberUsers` property has a value less than or equal to `42`. Any offers for which the expression cannot be evaluated are appended to the set of offers returned after all the offers where the expression is `FALSE`.

## 4.3 Trader Constraint Recipe Language

The Trader Constraint Recipe Language is used to construct secondary constraint expressions when resolving proxy offers.

As with TCL itself, such constraint recipes may be expressed in other languages provided they are identified with an escape prefix of the form `<<name major.minor>>`. The absence of a prefix is treated as equivalent to the use of the `<<OMG 1.0>>` prefix, which identifies a recipe string which complies with the standard. The OpenFusion Trading Service only supports the standard language.

Any constraint recipe string with such a prefix which is not recognised by a particular trader is simply ignored and the operation continues as though no recipe string was present.

An empty constraint is returned when no recipe string is given.

The secondary constraint expression is constructed from the primary constraint expression, the properties associated with the proxy offer and the recipe string. The algorithm for constructing the secondary constraint expression is:

```
while not end of recipe
   fetch the next character from the recipe
   if not a ’$’ character
      append the character to the secondary constraint
   else
      fetch next character from the recipe
      if a ’*’ character
         append the entire primary constraint to the secondary
         constraint
      else if not a ’(’ character
         append the character to the secondary constraint
      else
         collect characters up to a ’)’ character, discarding
         ’)’
         lookup property with that name
         append formatted value of that property to secondary
         constraint
```

For example:

A proxy offer has been exported to the trader with the following properties:

| | |
|---|---|
| `numberUsers` | `42` |
| `machineName` | `’ithuriel’` |
| `serialNumber` | `123456` |
| `cost` | `1099` |

and the primary constraint string for the query is `’.co.uk’ in domainName`.

The following recipes will generate the given secondary constraint which is used in the call to the trader behind the proxy.

| Recipe | Secondary Constraint |
|---|---|
| **`numberUsers == $(numberUsers)`** | **`numberUsers == 42`** |
| **`machineName == $(machineName)`** | **`machineName == ’ithuriel’`** |

| Recipe | Secondary Constraint |
|---|---|
| `serial number is $#$(serialNumber)` | `serial number is #123456` |
| `$* and machineName == $(machineName)` | `'.co.uk' in domainName and machineName == 'ithuriel'` |
| `price == $$$(cost)` | `price == $42` |

## 4.4 Valid Service Type Names

Service Type Names can either be valid *IDL interface repository names* or *identifiers* (as defined by the OMG) or *scoped names* of the form:

```
::Scope1::Scope2::Name
```

The initial `::` is not mandatory; the name need not contain any scoping. The alphanumeric identifiers must not begin with a digit but can contain the underscore character in addition to upper and lower case letters and digits.

Some examples of valid service type names are:

```
test
_test
::scope_1::_test
Xscope::test_X
```

Some examples of service type names which do not conform to the OMG specification are shown in *Table 5*.

**Table 5 Service Type Name Errors**

| Invalid Name | Reason(s) |
|---|---|
| `1Scope:test` | Numeric first character not allowed; single colon not a valid scope symbol |
| `2test` | Numeric first character not allowed |
| `::scope#1::test` | Non-alphanumeric character # in identifier |
| `A Scope::the test` | Embedded spaces not allowed |
| `scope : test` | Embedded spaces not allowed; single colon not a valid scope symbol |
| `test::` | Incomplete (scope symbol should be followed by an alphanumeric identifier) |

*i* Note that checking of service type names can be disabled. The property `CheckServiceTypeNameFormat` can be switched on and off whilst the service is running. This can be done through the Administration Manager or remotely through SNMP.

## 4.5 XML Export and Import

The OpenFusion Trading Service can both export and import XML files containing details of service types and offers. This is performed at the command line on a per-trader basis: service types or offers for a single trader are handled with a single command.

The formats of the XML files are described later.

To export service type or offer definitions to XML files, use this command:

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

where the `<options>` are:

| Option | Description |
|---|---|
| `-s serviceTypeFile` | name of file to export service types into |
| `-o offerFile` | name of file to export offers into |
| `resolveName` | the trader's resolve name |
| `-f traderIORFile` | name of file containing the trader's IOR |
| `-i traderIOR` | the trader's IOR |

To import service type or offer definitions from XML files, use this command:

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

where the `<options>` are:

| Option | Description |
|---|---|
| `-s serviceTypeFile` | name of file to import service types from |
| `-o offerFile` | name of file to import offers from |
| `resolveName` | the trader's resolve name |
| `-f traderIORFile` | name of file containing the trader's IOR |
| `-i traderIOR` | the trader's IOR |

The options can occur in any order.

The `-s` option specifies the filename for service types. The `-o` option specifies the filename for offers. At least one of `-s` or `-o` must be present.

The trader to use is specified in one of four ways. The trader's resolve name can be given, its IOR can be given with the -i option, or a file containing the IOR can be specified with the `-f` option. If none of these options are given, then a resolve name of "TradingService" will be used.

## 4.6 Service Type XML File Format

Below is an example of an XML file describing service types that can be imported using the Service Type Repository Manager. The full DTD is supplied as part of the Trading Service distribution.

Each service type must have one `ServiceTypeName` element and one `InterfaceTypeName` element. One or more `BaseServiceTypeName` elements can be provided for each servicetype. One or more `Property` elements can be provided for each `servicetype`. Each property can be given a mode attribute, the possible values being:

- `normal` (the default)
- `readonly`
- `mandatory`
- `mandatory readonly`

Each property must have a name and an IDLType. The IDLType can either be a base IDL type or a defined IDL type. The Java `CLASSPATH` used to run the Service Type Repository Browser must contain the generated Java stubs for the IDL types when defined IDL types are to be used in import definitions.

```
<?xml version="1.0" encoding="UTF-8"?>

<ServiceTypes>

  <ServiceType>

    <ServiceTypeName>typeA</ServiceTypeName>

    <InterfaceTypeName>IDL:prismt.com/tests/TypeA:1.0</InterfaceTypeName>

    <Property mode="normal">

      <PropertyName>propA</PropertyName>

      <IDLType>string</IDLType>

    </Property>

  </ServiceType>
  <ServiceType>
    <ServiceTypeName>typeB</ServiceTypeName>
    <BaseServiceTypeName>typeA</BaseServiceTypeName>
    <InterfaceTypeName>IDL:prismt.com/tests/TypeB:1.0</InterfaceTypeName>
    <Property mode="readonly">
      <PropertyName>propB1</PropertyName>
      <IDLType>short</IDLType>
    </Property>
    <Property mode="mandatory">
      <PropertyName>propB2</PropertyName>
      <IDLType>long</IDLType>
    </Property>
  </ServiceType>
  <ServiceType>
    <ServiceTypeName>typeC</ServiceTypeName>
    <BaseServiceTypeName>typeA</BaseServiceTypeName>
    <BaseServiceTypeName>typeB</BaseServiceTypeName>
    <InterfaceTypeName>IDL:prismt.com/tests/TypeC:1.0</InterfaceTypeName>
    <Property mode="normal">
      <PropertyName>propC</PropertyName>
      <IDLType sequence="true">string</IDLType>
    </Property>
  </ServiceType>
</ServiceTypes>
```

## 4.7 Offer XML File Format

Below is an example of an XML file describing offers that have been exported by the trader. Please refer to the description of the Trading Service Manager for more information about the use of XML for exporting offers. The full DTD is supplied as part of the Trading Service distribution.

*i* Note that this XML description cannot be written by hand because property values can only be exported in a serialised form.

Each offer has one `Reference` element, which contains an IOR string, and one `ServiceTypeName` element. Offers can contain many `Property` elements. Each `Property` element has one `PropertyName` and one `PropertyValue` which is listed in string format. Offers cannot be migrated from a trader running one ORB to a trader running a different ORB because of the differences in the serialised form of the property values between the ORBs.

```
<?xml version="1.0" encoding="UTF-8"?>

<Offers>

 <OfferInfo>

  <Reference>{IOR}</Reference>

  <ServiceTypeName>film</ServiceTypeName>

  <Property>

   <PropertyName>title</PropertyName>

   <PropertyValue>{Serialised Value}</PropertyValue>

  </Property>

  <Property>

   <PropertyName>certificate</PropertyName>

   <PropertyValue>{Serialised Value}</PropertyValue>

  </Property>

 </OfferInfo>

</Offers>
```

## *4.8* Trading Service Types

The OMG Trading Service specification defines six kinds of trading service, which provide different levels of functionality depending on which interfaces they implement. This naming scheme is summarised in *Table 6*.

**Table 6 Kinds of Trading Service**

|  | Trading Service Components | | | | |
|---|---|---|---|---|---|
| **Kind of Trading Service** | **Lookup** | **Register** | **Admin** | **Link** | **Proxy** |
| **Query** | • | | | | |
| **Simple** | • | • | | | |
| **Stand-Alone** | • | • | • | | |
| **Proxy** | • | • | • | | • |
| **Linked** | • | • | • | • | |
| **Full-Service** | • | • | • | • | • |

There are two further Trading Service iterator interfaces in addition to those described in Table 7, *Trading Service Components*: `OfferIterator` and `OfferIdIterator`. These interfaces are used by operations to return lists of `Offer` or `OfferId`. They do this by returning a list of a given length and enabling the remainder of the list to be extracted by successive operations on the iterator interface.

The following abstract interfaces are defined by the Trading Service to enable the construction of traders which have differing support for the individual trader components.

- The `TraderComponents` interface provides a method of obtaining an object reference to any component when given a reference to one of the components. For example, the interface may be used to obtain a reference to the `Admin` component given a reference to the `Register` component
- The `SupportAttributes` interface gives a trading service implementation the ability to choose whether to support
  - modifiable properties
  - dynamic properties
  - proxy offers

- The `ImportAttributes` interface is used to configure the trading service with default and maximum values for constraints that apply to queries
- The `LinkAttributes` interface is used to configure the trading service as to the most permissive behaviour a new or modified link will be allowed

The `ServiceTypeRepository` forms a major part of the trading service and handles all processing of service types. This component is used to *add*, *remove*, *describe* and *mask* service types.

Dynamic properties in the trading service use the `DynamicPropEval` interface which implements properties whose values can be evaluated at runtime.

The OpenFusion Trading Service is a full service trader: it includes all of the required components (listed in Table 7, *Trading Service Components*), and implements all of the interfaces detailed above.

## 4.9 Trading Service Components

The Trading Service functionality is divided into five components. Each component has an associated interface. Individual implementations of the Trading Service may choose to implement some or all of the components; the OpenFusion Trading Service is a full service implementation, and so includes all components.

The five interfaces are separable and are summarised in *Table 7*.

**Table 7 Trading Service Components**

| Component | Description |
|---|---|
| **`Lookup`** | Queries the trading service to find offers matching given requirements. |
| **`Register`** | Manages service offers. Offers can first be exported, then later modified or withdrawn. |
| **`Admin`** | Configures the trading service settings. |
| **`Link`** | Supports the federation of trading services. |
| **`Proxy`** | Supports the delayed evaluation of offers. |

## 4.10 Exceptions

The Trading Service supports many exceptions which are summarised in the following tables.

| Exception | Description |
|---|---|
| **DuplicatePolicyName** | The policy name is specified more than once. |
| **DuplicatePropertyName** | The property name is specified more than once. |
| **IllegalConstraint** | The constraint given has invalid syntax. |
| **IllegalOfferId** | The `OfferId` has invalid syntax. |
| **IllegalPropertyName** | The property name has invalid syntax. |
| **IllegalServiceType** | The service type has invalid syntax. |
| **InvalidLookupRef** | An invalid lookup target has been given. |
| **MissingMandatoryProperty** | The property that was declared mandatory by the service type has not been given a value. |
| **NotImplemented** | The functionality has not been implemented to carry out this operation. |
| **PropertyTypeMismatch** | The property value type is not the same type as declared in the service type. |
| **ReadonlyDynamicProperty** | The `readonly` property cannot have a dynamic value assigned to it. |
| **UnknownMaxLeft** | The `OfferIterator` cannot determine the number of offers left. |
| **UnknownOfferId** | The `OfferId` is not recognised by the trader. |
| **UnknownServiceType** | The service type is not recognised. |

| Exception | Description |
|---|---|
| **IllegalPolicyName** | The policy name specified has invalid syntax. |
| **IllegalPreference** | The preference has invalid syntax. |
| **InvalidPolicyValue** | The policy value is invalid. |
| **PolicyTypeMismatch** | The policy value type is not the specified type. |

| Exception | Description |
|---|---|
| **`IllegalTraderName`** | The Trader name has invalid syntax. |
| **`InterfaceTypeMismatch`** | The interface of the object reference is not a subtype of the service type interface. |
| **`InvalidObjectRef`** | The object reference cannot be exported. |
| **`MandatoryProperty`** | The property has been declared as mandatory when that is not possible. |
| **`NoMatchingOffers`** | The constraint does not match any offers. |
| **`ProxyOfferId`** | The `OfferId` belongs to a proxy offer. |
| **`ReadonlyProperty`** | An attempt was made to modify a `readonly` property. |
| **`RegisterNotSupported`** | The trader does not support the register component. |
| **`UnknownPropertyName`** | The property name does not exist in the offer. |
| **`UnknownTraderName`** | The trader name was not found, or the trader does not support links. |

| Exception | Description |
|---|---|
| **`IllegalRecipe`** | The given constraint recipe is not well formed. |
| **`NotProxyOfferId`** | The `OfferId` does not belong to a proxy offer. |

| Exception | Description |
|---|---|
| **`IllegalLinkName`** | The link name has invalid syntax. |
| **`UnknownLinkName`** | The link name is not known to this trader. |
| **`DefaultFollowTooPermissive`** | The default mode for following links specified by the query is more permissive than the link is willing to tolerate. |
| **`LimitingFollowTooPermissive`** | The most permissive link follow behaviour given to this link is more permissive than the value allowed by trader's attribute `max_link_follow_policy` at the time of the links creation. |

| Exception | Description |
|---|---|
| **`DPEvalFailure`** | The property value cannot be evaluated. |

| Exception | Description |
|---|---|
| **`AlreadyMasked`** | An attempt was made to mask a service type that is already masked. |
| **`DuplicateServiceTypeName`** | The service type name has been specified more than once. |
| **`HasSubTypes`** | An attempt was made to remove a service type with extant subtypes. |
| **`InterfaceTypeMismatch`** | The `if_name` parameter is not a subtype of the interface associated with a service type from which this service type is derived. |
| **`NotMasked`** | An attempt was made to unmask a service type that is not masked. |
| **`ServiceTypeExists`** | An attempt was made to redefine an existing service type. |
| **`ValueTypeRedefinition`** | Two supertypes are declaring different value types for the same property name. |

# Configuration and Management

CHAPTER

# 5 Trading Service Configuration

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

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

*Also, the configuration settings enable the Quality of Service and administration properties to be customised when needed.*

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

## 5.1 Common Properties

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

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

## 5.2 TradingSingleton Configuration

### 5.2.1 CORBA Properties

#### IOR Name Service Entry

The Naming Service entry for the Singleton.

| | |
|---|---|
| **Property Name** | `Object.Name` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### IOR URL

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

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

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

| | |
|---|---|
| **Property Name** | `IOR.URL` |
| **Property Type** | `FIXED` |
| **Data Type** | `URL` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### IOR File Name

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

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

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

| | |
|---|---|
| **Property Name** | `IOR.File` |
| **Property Type** | `FIXED` |
| **Data Type** | `FILE` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### Resolve Name

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

| | |
|---|---|
| **Property Name** | `ResolveName` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

#### IOR Name Service

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

| | |
|---|---|
| **Property Name** | `IOR.Server` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

### *5.2.2* Persistence Properties

#### Cache Metadata

If selected, metadata (Support Attributes, Import Attributes, and Link Attributes) will be cached for speed.

| | |
|---|---|
| **Property Name** | `DB.CacheMetadata` |
| **Property Type** | `FIXED` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

#### Query Cache Size

The number of SQL queries allowed in the database query cache.

| **Property Name** | `DB.QueryCacheSize` |
|---|---|
| **Property Type** | `FIXED` |
| **Data Type** | `INTEGER` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### 5.2.3 General Properties

#### Create Persistent Iterators

This flag determines whether offer and offer id iterators are created as persistent or transient objects. By default, transient iterators are created; these are invalidated when the services shut down.

| **Property Name** | `PersistentIterators` |
|---|---|
| **Property Type** | `DYNAMIC` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

#### Evaluation Timeout (secs)

The peroid, in seconds, which the trader will wait for a dynamic property to be evaluated. The default value for this property is `zero`.

| **Property Name** | `DynamicProperty.Timeout` |
|---|---|
| **Property Type** | `DYNAMIC` |
| **Data Type** | `INTEGER` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

#### Purge Interval

Invalid offers (object references which are not active and not persistent) will be purged from the system periodically.

This property sets the interval, in minutes, between invalid offer purges.

A value of `zero` (the default value) means that there is no purging of invalid offers.

| | |
|---|---|
| **Property Name** | `Clean.Interval` |
| **Property Type** | `DYNAMIC` |
| **Data Type** | `INTEGER` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### Purge at Startup

All invalid offers (object references which are not active and not persistent) are removed when the **Purge at Startup** option is selected.

| | |
|---|---|
| **Property Name** | `Clean.Startup` |
| **Property Type** | `FIXED` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### Service Type Repository Name

Specifies the name of the Service Type Repository to be used by this Singleton. Multiple Trading Service Singletons can share a Service Type Repository.

Setting this property avoids the need to perform sharing at runtime with the CORBA API.

| | |
|---|---|
| **Property Name** | `ServiceTypeRepositoryName` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### Offer Count

Contains the count of offers exported since the Trading Service started or since the counter was last reset to zero.

| | |
|---|---|
| **Property Name** | `OfferCount` |
| **Property Type** | `DYNAMIC` |
| **Data Type** | `COUNTER` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

## 5.3 ServiceTypeRepositorySingleton Configuration

### 5.3.1 CORBA Properties

#### IOR Name Service Entry

The Naming Service entry for the Singleton.

| | |
|---|---|
| **Property Name** | `Object.Name` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### IOR URL

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

```
file:/usr/users/openfusion/ServiceTypeRepositorySingleton.ior
http://www.prismtechnologies.com/openfusion/ServiceTypeRepositorySin
gleton.ior
```

| | |
|---|---|
| **Property Name** | `IOR.URL` |
| **Property Type** | `FIXED` |
| **Data Type** | `URL` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### IOR File Name

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

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

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

| | |
|---|---|
| **Property Name** | `IOR.File` |
| **Property Type** | `FIXED` |

| | |
|---|---|
| **Data Type** | `FILE` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

### Resolve Name

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

| | |
|---|---|
| **Property Name** | `ResolveName` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### IOR Name Service

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

| | |
|---|---|
| **Property Name** | `IOR.Server` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

## 5.3.2 Persistence Properties

### Cache Metadata

This property controls whether or not metadata (Service Types) will be cached.

| | |
|---|---|
| **Property Name** | `DB.CacheMetadata` |
| **Property Type** | `FIXED` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

### 5.3.3 General Properties

#### Check Name Format

If this is selected, the `CheckNameFormat` property of the `ServiceTypeRepositorySingleton` will check the interface name as well as the service type name for validity, to ensure strict standards compliance.

| | |
|---|---|
| **Property Name** | `CheckServiceTypeNameFormat` |
| **Property Type** | `DYNAMIC` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

#### Service Type Count

The number of new Service Types registered since the Trading Service started or was last reset.

| | |
|---|---|
| **Property Name** | `ServiceTypeCount` |
| **Property Type** | `DYNAMIC` |
| **Data Type** | `BOOLEAN` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `YES` |

## 5.4 ProcessSingleton Configuration

#### IOR Name Service Entry

The Naming Service entry for the Singleton.

| | |
|---|---|
| **Property Name** | `Object.Name` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

#### IOR URL

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

```
file:/usr/users/openfusion/ProcessSingleton.ior
http://www.prismtechnologies.com/openfusion/ProcessSingleton.ior
```

| | |
|---|---|
| **Property Name** | `IOR.URL` |
| **Property Type** | `FIXED` |
| **Data Type** | `URL` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

## IOR File Name

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

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

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

| | |
|---|---|
| **Property Name** | `IOR.File` |
| **Property Type** | `FIXED` |
| **Data Type** | `FILE` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

## IOR Name Service

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

| | |
|---|---|
| **Property Name** | `IOR.Server` |
| **Property Type** | `FIXED` |
| **Data Type** | `STRING` |
| **Accessibility** | `READ/WRITE` |
| **Mandatory** | `NO` |

# CHAPTER 6 Trading Service Manager

*The Trading Service Manager, supplied as part of the OpenFusion implementation of the CORBA Trading Service, is a tool for the administration, management, and use of the Trading Service. It can be used to manage any CORBA-compliant trading service, even when the service is not a full-service trader implementation.*

*The following tasks can be carried out from the Trading Service Manager:*

- Perform queries to find offers that match given criteria, constraints, and preferences.
- View all offers.
- View all proxy offers.
- View all linked (federated) Trading Services.

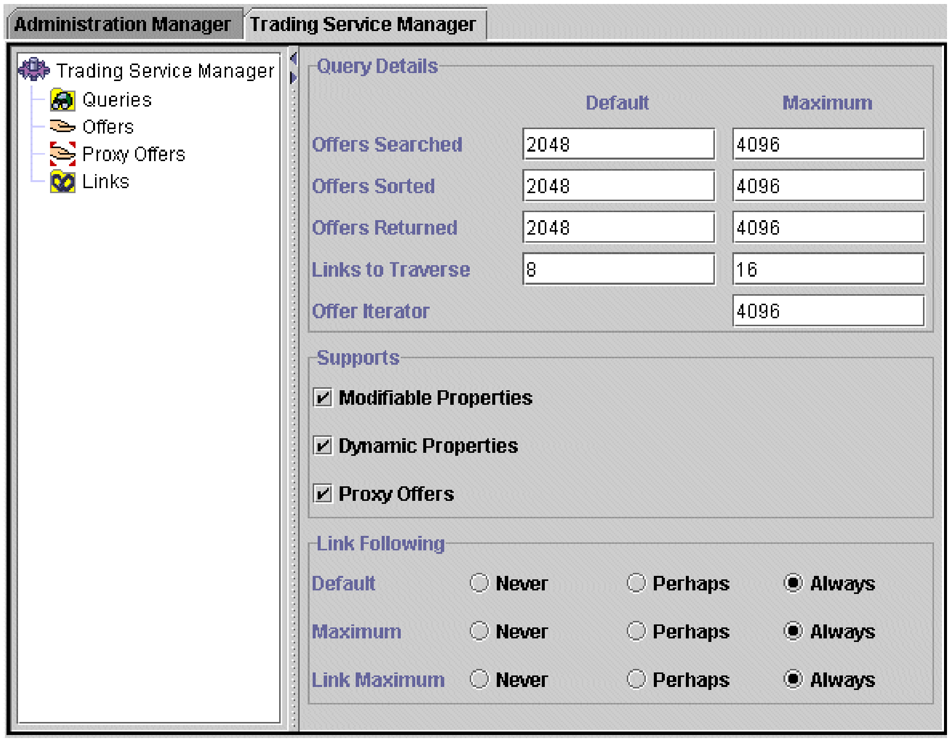
## 6.1 Using the Trading Service Manager

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

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

```
%  run
   com.prismt.cos.treebrowser.trader.TraderServicesBrowser
   -name TradingService
```

The Trading Service Manager is illustrated in *Figure 4*.



Query Details

| | Default | Maximum |
|---|---|---|
| Offers Searched | 2048 | 4096 |
| Offers Sorted | 2048 | 4096 |
| Offers Returned | 2048 | 4096 |
| Links to Traverse | 8 | 16 |
| Offer Iterator | | 4096 |

Supports

- ✓ Modifiable Properties
- ✓ Dynamic Properties
- ✓ Proxy Offers

Link Following

| | | | |
|---|---|---|---|
| Default | Never | Perhaps | [✓ Always] |
| Maximum | Never | Perhaps | [✓ Always] |
| Link Maximum | Never | Perhaps | [✓ Always] |

**Figure 4 Trading Service Manager**

Different nodes in the Trading Service Manager are identified by different icons. These icons are shown in *Table 8*.

**Table 8 Trading Service Object Icons**

| Icon | Node |
|---|---|
| | **Trading Service** The Trading Service root node represents the current instance of the Trading Service. |
| | **Queries** The parent node for queries. |

Table 8 Trading Service Object Icons (Continued)

| Icon | Node |
|---|---|
|  | **Offers**<br>The parent node for offers. |
|  | **Proxy Offers**<br>The parent node for proxy offers. |
|  | **Links**<br>The parent node for linked (federated) trading services. |

## 6.1.1 Trading Service Properties

If the **Trading Service Manager** root node is selected, properties relating to the current instance of the Trading Service are displayed in the right-hand panel. These properties can be changed and will take effect while the Trading Service is running, but if the Trading Service is stopped and restarted the changes will be lost.

### 6.1.1.1 Query Details

The top section of the panel, **Query Details**, contains the properties that affect the behaviour of queries to the trader. The first four of these properties have **Default** and **Maximum** values.

When a client of the Trading Service does not specify a value for a query property, the **Default** value is used. When a client specifies a value that is greater than the **Maximum** value, the **Maximum** value is used instead of the specified value.

##### 6.1.1.1.1 Offers Searched

The **Offers Searched** property specifies the upper bound of offers to be searched during a query operation.

##### 6.1.1.1.2 Offers Sorted

The **Offers Sorted** property specifies the upper bound of matched offers to be sorted according to the preference criteria.

##### 6.1.1.1.3 Offers Returned

The **Offers Returned** property specifies the upper bound of sorted offers to be returned by a query operation.

##### *6.1.1.1.4* Links to Traverse

The **Links to Traverse** property specifies the upper bound of links to traverse in carrying out a query.

##### *6.1.1.1.5* Offer Iterator

The **Offer Iterator** property specifies the upper bound on the size of any list returned by the trader. This property does not have a default value.

#### *6.1.1.2* Supports

The **Supports** section of the **Admin** panel contains check box options that are used to modify query behaviour. These options cannot be selected unless supported by the Trading Service.

##### *6.1.1.2.1* Modifiable Properties

Offers containing modifiable properties may be returned as a result of the query when this setting is checked.

##### *6.1.1.2.2* Dynamic Properties

Offers containing dynamic properties may be returned as a result of the query when this setting is checked.

##### *6.1.1.2.3* Proxy Offers

Proxy offers may be returned as a result of the query when this setting is checked.

#### *6.1.1.3* Link Following

The properties in the **Link Following** section specify the behaviour of the trader in following links to other federated traders during a query operation.

The possible property values are:

- *Never*: for never following links.
- *Perhaps*: for following links only when the trader cannot provide any matching offers locally.
- *Always*: for always following links.

##### *6.1.1.3.1* Default

The **Default** property specifies the link following behaviour for queries carried out on the current trader when the client does not specify a policy as part of the query.

##### *6.1.1.3.2* Maximum

The **Maximum** property is the most permissive link following behaviour for queries carried out on the current trader, no matter what behaviour is requested.

#### *6.1.1.3.3* Link Maximum

The **Link Maximum** property is the most permissive link following behaviour that can be given to new links created from this trader, no matter what behaviour is requested.

### *6.1.2* Tool Bar Buttons

The Trading Service Manager adds buttons to the tool bar. These buttons are only available when the Trading Service Manager is active. The new tool bar buttons are shown in *Table 9*.

**Table 9 Trading Service Tool Bar**

| Button | Function |
|---|---|
| 📂 | Load Offers From XML<br><br>Prompts for an XML file and will load offers identified by that XML file into the Trading Service. |
| ▬ | Withdraw Selected Offers<br><br>Removes the selected offers from the Trading Service. |

## *6.2* Queries

Right-click on the **Query** node of the Trading Service Manager and select **Create Query** from the pop-up menu. Name the query and click **OK** to create the query as a child node of the **Query** node. Each query must have a unique name.

A query can only be created if there are Service Types in the Trading Service.

### *6.2.1* Query Constraints

Constraint properties can be set to narrow the query results. Select a specific query node to display the property pane for that query.

#### *6.2.1.1* Properties

Click the **Properties** tab to display the properties panel for the query.

##### *6.2.1.1.1* Offers to Return

The **Offers to Return** property specifies the maximum number of offers returned by the query.

##### *6.2.1.1.2* Service Type

The **Service Type** property specifies the service type returned by the query.

The service type is either chosen from the drop-down list or typed into the field directly. The-drop down list contains only those service types that are currently active in the trader. However, a query can be based on other service types as offers may be found on linked traders or with masked or deleted service types.

##### *6.2.1.1.3* Query Constraints

Type a query expression into the **Query Constraints** field. The expression is written using the *Trader Constraint Language*. For details of using the Trader Constraint Language, see the *OpenFusion Trading Service Guide*.

The last five query expressions are saved and can be selected from a drop-down list.

##### *6.2.1.1.4* Properties to Return

The **Properties to Return** property specifies the offer properties returned by the query.

The offers returned by the trader can be specified to contain:

- **All**: a complete list of their properties.
- **Some**: a partial list of their properties. Choose the properties using the **AddProperty** button.
- **None**: no properties.

#### *6.2.1.2* Policies

Click the **Policies** tab to display the policies panel.

All policies default to the values given to the various policies on the **Admin** panel, with the exception of the **Service Type is Exact Match Only** policy.

The query considers only those offers that have the exact service type given, rather than offers of both the service type given and any of its sub-types when **Service Type is Exact Match Only** is set.

#### *6.2.1.3* Preferences

Click the **Preferences** tab to display the preferences panel. Preferences specify how the results of the query should be ordered:

The offers can be ordered as follows:

- Randomly.
- In order as found.
- Sorted in descending order by the numerical expression typed into the **Numeric Expression** field, which refers to the properties associated with the matching offers.

- Sorted in ascending order by the numerical expression typed into the **Numeric Expression** field, which refers to the properties associated with the matching offers.
- Such that the offers for which the constraint expression typed into the **Constraint Expression** field evaluates to **TRUE** are placed before those offers for which the constraint expression evaluates to **FALSE**.

In the last three cases, if the expression cannot be evaluated for a particular offer, that offer is placed at the end of the list (after the other offers).

### *6.2.2* Execute a Query

Right-click on a query and select **Execute Query** from the pop-up menu.

All offers that match the query criteria are found and returned as child nodes of the query.

A query will only show offers that existed at the time the query was executed. To update the query to include new offers, right-click on the query and select **Refresh Query** from the pop-up menu.

To remove a query, right-click on the query and select **Delete Query** from the pop-up menu.

#### *6.2.2.1* Offers

Each offer returned by a query is displayed as a child node of that query. Each offer has icons to represent its active status and persistent state. These icons are shown in Table 10, *Offer Icons*, on page 76.

The offers listed under a query are a sub-set of the offers listed under the **Offers** node. That is, the query returns a sub-set of the offers available in the Trading Service. See *Offers* on page 76 for details.

Click on an offer to see full details of the offer in the right-hand pane.

### *6.2.3* Export Offers

The offers returned by a query can be saved in an XML file. The offers can later be reloaded into the Trading Service, or into a new Trading Service instance, by using the **Load Offers From XML** tool bar button.

The **Export Query** command is only enabled for the query if the **Show All Properties** option was selected in the query properties.

To save all offers for a query, right-click on the query and select **Export Query** from the pop-up menu. Choose a directory and enter a file name for the XML file.

## 6.3 Offers

Expand the **Offers** node to see the offers available in the Trading Service. The first 100 offers available in the Trading Service are listed. To see the next 100 offers, right-click on the parent node and select **Get Next 100 Offers** from the pop-up menu. To return to the first 100 offers, select **Get First 100 Offers** from the pop-up menu.

Each offer has icons to represent its active status and persistent state. These icons are shown in *Table 10*.

**Table 10 Offer Icons**

| Icon | Status and Persistence |
|---|---|
| 🔴🟥 | **Inactive** and **Non-persistent** (both red) |
| 🔴🟩 | **Inactive** and **Persistent** (red and green) |
| 🟢🟥 | **Active** and **Non-persistent** (green and red) |
| 🟢🟩 | **Active** and **Persistent** (both green) |
| ❓🟥 | **Undetermined** status and **Non-persistent** (grey and red) |
| ❓🟩 | **Undetermined** status and **Persistent** (grey and green) |

The status of an offer is also displayed as a tool tip for the offer.

Select an offer to display its properties in the right-hand pane.

### 6.3.1 Properties

All offers have the following default properties:

- Offer Service Type.
- Decoded Type (only if the offer status is active).
- Decoded Host (only if the offer status is active).
- Decoded Port (only if the offer status is active).

Offers will have other properties specific to the offer.

### 6.3.2 Withdraw an Offer

To withdraw an offer from the Trading Service, right-click on the offer and select **Withdraw Offer** from the pop-up menu.

To withdraw multiple offers, select the offers (use Ctrl-click to select multiple offers or Shift-click to select offers in a contiguous block) and click the **Withdraw Offer** button on the tool bar. All selected offers will be withdrawn from the Trading Service and will be removed from the Trading Service Manager.

## 6.4 Proxy Offers

Expand the **Proxy Offers** node to see all proxy offers available in the Trading Service.

Expand the **Proxy Offers** node to see the proxy offers available in the Trading Service. The first 100 offers available in the Trading Service are listed. To see the next 100 offers, right-click on the parent node and select **Get Next 100 Offers** from the pop-up menu. To return to the first 100 offers, select **Get First 100 Offers** from the pop-up menu.

Each proxy offer has icons to represent its active status and persistent state. These icons are shown in Table 10, *Offer Icons*, on page 76.

Proxy offers have the same functionality as other offers. See *Offers* on page 76 for details.

## 6.5 Links

The **Links** node lists all federated instances of the Trading Service. Expand the node to show every Trading Service that the current Trading Service is linked to. Expanding any of those nodes will show all Trading Services that they are linked to in turn. A complete graph of the federated Trading Services can be produced in this way.

Right-click on any federated Trading Service node to get a pop-up menu with the following options:

- CORBA Object browser
- Service Type Repository Manager
- Trading Service Manager

Selecting any of these options will open a new browser of the specified type. For example, select **Trading Service Manager** to open the federated Trading Service in a new instance of the Trading Service Manager.

CHAPTER

# 7 Service Type Repository Manager

*The Service Type Repository Manager is an administration tool for the OpenFusion Trading Service which allows the contents of the Trader Service Type Repository to be managed.*

Use the Service Type Repository Manager to:

- View service types in the repository.
- Add service types.
- Remove service types.
- Import service type descriptions from XML files.
- Mask or unmask service types.

## 7.1 Using the Service Type Repository Manager

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

The Service Type Repository Manager is shown in *Figure 5*.

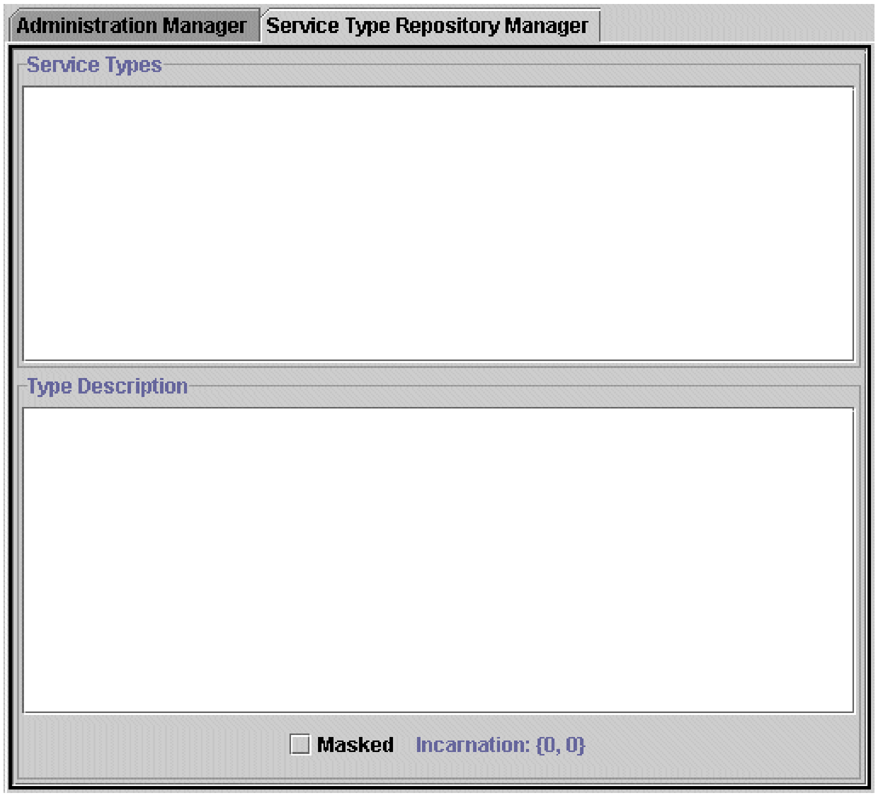


**Figure 5 Service Type Repository Manager**

The **Service Types** panel contains a list of all service types that are currently active in the trader. The **Type Description** panel shows the description of the service type selected in the **Service Types** panel.

### *7.1.1* Tool Bar Buttons

The Service Type Repository Manager adds new buttons to the tool bar. These buttons are only available when the Service Type Repository Manager is active. The new tool bar buttons are shown in Table 11, *Service Type Repository Manager Tool Bar*.

**Table 11 Service Type Repository Manager Tool Bar**

| Button | Function |
|---|---|
| 📂 | Load Service Types<br>Load service types from a plain text file. |
| 💾 | Save Service Types<br>Save service types in a plain text file. |
| ➕ | Add Service Types<br>Add a new service type to the repository. |
| ➖ | Remove Service Types<br>Delete selected service types from the repository. |
| XML 📂 | Load Service Types XML<br>Load service type descriptions from an XML file. |
| 💾 XML | Save Service Types XML<br>Save service types in an XML file. |

### *7.1.2* Removing Service Types

Select one or more service types in the Service Type panel and click the **Remove Service Types** tool bar button to remove the selected service types from the repository.

Use Ctrl-click to select multiple service types or Shift-click to select service types in a contiguous block.

### 7.1.3 Adding Service Types

Click the **Add Service Types** tool bar button to add a service type to the repository. The **Add Service Types** dialog box (*Figure 6*) is shown.

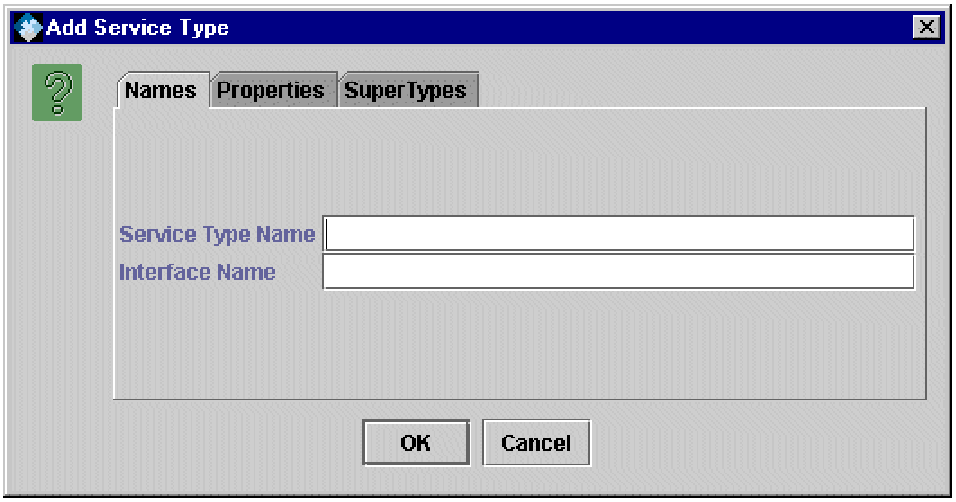


**Figure 6 Add Service Types Dialog Box**

#### 7.1.3.1 Names

##### 7.1.3.1.1 Service Type Name

The name of the service type.

##### 7.1.3.1.2 Interface Name

The name of the service type interface.

#### 7.1.3.2 Properties

The **Properties** tab of the **Add Service Type** dialog lists properties assigned to the service type. To add a new property, click the **Add Property** button. The **Add Property** dialog box (*Figure 7*) is shown.

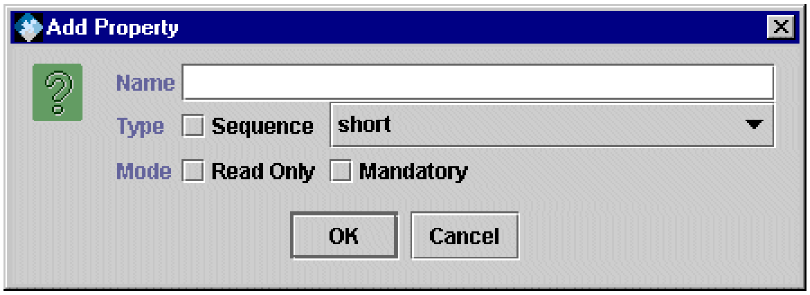


**Figure 7 Add Property Dialog Box**

##### 7.1.3.2.1 Name

The name of the property.

##### 7.1.3.2.2 Sequence

Check the **Sequence** box if the property is designated as a sequence.

##### 7.1.3.2.3 Type

The property can be one of the following types:

- short
- long
- unsigned short
- unsigned long
- float
- double
- boolean
- char
- string

##### 7.1.3.2.4 Mandatory

Check the **Mandatory** box if the property is designated as a mandatory property.

##### 7.1.3.2.5 Read Only

Check the **Read Only** box if the property is designated as a read-only property.

### 7.1.4 Load Service Types

The **Load Service Types XML** tool bar button allows a batch import of service types from an XML file into the repository.

Use the File Browser dialog to select the file containing the service type descriptions.

Alternatively, use the **Load Service Types** tool bar button to import service types from a plain text file.

The format of service types XML files and plain text files is documented in the *OpenFusion Trading Service Guide*.

### 7.1.5 Save Service Types

Select one or more service types in the Service Type panel and click the **Save Service Types XML** tool bar button to save the selected types to an XML file.

Alternatively, use the **Save Service Types** tool bar button to save service types to a plain text file.

Use Ctrl-click to select multiple service types or Shift-click to select service types in a contiguous block.

The format of service types XML files and plain text files is documented in the *OpenFusion Trading Service Guide*.

### 7.1.6 Mask Service Types

Offers which have a masked service type cannot be advertised.

To mask a service type, select the service type and check the **Masked** checkbox.

To unmask a masked service type, select the service type and clear the **Masked** checkbox.

# Index

# Index

PrismTech