



Data, Information and Process integration  
with Semantic Web Services

**DIP**

*Data, Information and Process Integration with Semantic Web Services*

**FP6 – 507483**

Deliverable

**WP6: Interoperability and Architecture**

**D6.4**

**WSMO API**

Marin Dimitrov, Damyan Ognyanov, Atanas Kiryakov

4th February 2005



## SUMMARY

wsmo4j is an open-source project with two parts:

- WSMO API — application programming interfaces for Web Services Modelling Ontology (WSMO, v.1.0), which allow for basic manipulation of WSMO descriptions, e.g. creation, exploration, storage, retrieval, parsing, and serialization;
- wsmo4j — a reference implementation of the WSMO API, including a WSML parser and a file-system-based datastore.

One of the major design rationales behind the WSMO API is to ensure the compatibility and interoperability between the web service-related and the ontology management-related software infrastructure within DIP. wsmo4j fits in the basis of the WSMO Studio, which is a SWS browser and an integrated development environment (IDE) specified and developed within DIP workpackage WP4 (deliverables D4.4, D4b.5, D4b.11). The *Ontology* package of the API will also serve as a core of the ontology representation and data integration (ORDI) framework, which in itself is the interoperability basis of the ontology management and the reasoning infrastructure, which is developed in workpackages WP2 and WP1.

This document is derived from the "wsmo4j Programmer's Guide" and provides a short introduction, a reference-style documentation for the WSMO API, and a few examples. The reference implementation is not documented hereby.

The WSMO API is a directly exploitable software component, which is already used in WP2 and WP4. Most of the tool developers and the technical partners in the use cases should make use of the API.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.

## DOCUMENT INFORMATION

<b>IST Project Number</b>	FP6 – 507483	<b>Acronym</b>	DIP
<b>Full Title</b>	Data, Information, and Process Integration with Semantic Web Services		
<b>Project URL</b>	http://dip.semanticweb.org/		
<b>Document URL</b>			
<b>EU Project Officer</b>	Brian Macklin		

<b>Deliverable</b>	<b>Number</b>	6.4	<b>Title</b>	WSMO API
<b>Work Package</b>	<b>Number</b>	6	<b>Title</b>	Interoperability and Architecture








<b>Date of Delivery</b>	<b>Contractual</b>	M12	<b>Actual</b>	31-Dec-04
<b>Status</b>	version 0.2		final <input type="checkbox"/>	
<b>Nature</b>	prototype <input checked="" type="checkbox"/> report <input type="checkbox"/> dissemination <input type="checkbox"/>			
<b>Dissemination Level</b>	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			









<b>Authors (Partner)</b>	Marin Dimitrov, Damyan Ognyanov, Atanas Kiryakov (all from Sirma AI)			
<b>Resp. Author</b>	Atanas Kiryakov		<b>E-mail</b>	naso@sirma.bg
	<b>Partner</b>	Ontotext Lab, Sirma AI	<b>Phone</b>	+359 (2) 9768 303



<b>Abstract (for dissemination)</b>	wsmo4j is an API and a Reference Implementation for the Web Services Modelling Ontology (WSMO, v.1, 20.09.2004). wsmo4j is Java based and is distributed under a LGPL licence. This document is derived from the "wsmo4j Programmer's Guide" and provides documentation for the API and few examples; the reference implementation is not documented here.
<b>Keywords</b>	Semantic Web Services, API, WSMO

Version Log			
Issue Date	Rev No.	Author	Change
15-01-05	1	Atanas Kiryakov	Derived from "wsmo4j Programmer's Guide" (26.11.2004), based on wsmo4j version rc1 from 16.11.2004.
31-01-05	2	Atanas Kiryakov	Changes according to the reviewer's comments and an internal proofreading.

## PROJECT CONSORTIUM INFORMATION

Partner	Acronym	Contact
National University of Galway	NUIG 	Prof. Dr. Christoph Bussler Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland E-mail: chris.bussler@deri.ie Tel: +353 91 512460
Fundacion De La Innovacion.Bankinter	Bankinter 	Monica Martinez Montes Fundacion de la Innovation. BankInter, Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
Berlecon Research GmbH	Berelcon 	Dr. Thorsten Wichmann Berlecon Research GmbH, Oranienburger Str. 32, 10117 Berlin, Germany E-mail: tw@berlecon.de Tel: +49 30 2852960
British Telecommunications Plc.	BT 	Dr. John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL 	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland E-mail : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex 	Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. E-mail: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI 	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany E-mail: abecker@fzi.de Tel: +49 721 96540

Institut für Informatik, Leopold-Franzens Universität Innsbruck	<p>UIBK</p> 	<p>Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485</p>
ILOG SA	<p>ILOG</p>  <p>Changing the rules of business</p>	<p>Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981</p>
inubit AG	<p>inubit</p>  <p>the integration experts</p>	<p>Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany E-mail: ts@inubit.com Tel: +49 30726112 0</p>
Intelligent Software Components, S.A.	<p>iSOCO</p> 	<p>Dr. V. Richard Benjamins, Director R&amp;D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain E-mail: rbenjamins@isoco.com Tel. +34 913 349 797</p>
The Open University	<p>OU</p> 	<p>Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: j.b.domingue@open.ac.uk Tel.: +44 1908 655014</p>
SAP AG	<p>SAP</p> 	<p>Dr. Elmar Dörner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: elmar.dorner@sap.com Tel: +49 721 6902 31</p>
Sirma AI Ltd.	<p>Sirma</p>  <p>Ontotext Knowledge and Language Engineering Lab of Sirma</p>	<p>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chaussée, Sofia 1784, Bulgaria E-mail: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303</p>
Tiscali Österreich GmbH	<p>Tiscali</p> 	<p>Dr Dieter Haacker Tiscali Österreich GmbH. Diefenbachgasse 35, A-1150 Vienna, Austria E-mail: Dieter.Haacker@at.tiscali.com Tel: +43 1 899 33 160</p>

Unicorn Solution Ltd.	<p>Unicorn</p> 	<p>Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111</p>
Vrije Universiteit Brussel	<p>VUB</p> 	<p>Carlo Wouters, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium E-mail: carlo.wouters@vub.ac.be Tel.: +32 (0) 2 629 3719</p>

---

## LIST OF KEYWORDS/ABBREVIATIONS

<b>WSMO</b>	Web Service Modelling Ontology (see <a href="http://www.wsmo.org">http://www.wsmo.org</a> )
<b>WSML</b>	Web Service Modelling Language (see <a href="http://www.wsmo.org/wsml">http://www.wsmo.org/wsml</a> )
<b>WSMX</b>	Web Service Execution Environment (see <a href="http://www.wsmo.org/wsmx">http://www.wsmo.org/wsmx</a> )
<b>JavaDoc</b>	a tool for generating API documentation in HTML format from doc comments in Java source code (see <a href="http://java.sun.com/j2se/javadoc/">http://java.sun.com/j2se/javadoc/</a> )
<b>SourceForge</b>	portal providing environment for open-source software development, publication and maintenance. (see <a href="http://www.sourceforge.net">http://www.sourceforge.net</a> )
<b>API</b>	Application Programming Interface(s)
<b>NFP</b>	Non-Functional Properties, see page 6
<b>Xref</b>	a hyper-textual (HTML) Java source-code documentation, enhancing the standard JavaDoc by fully cross-referencing source code, see <a href="http://www.xref-tech.com/java2html/main.html">http://www.xref-tech.com/java2html/main.html</a>

---

# CONTENTS

1	INTRODUCTION	1
1.1	Versions and Links	2
1.2	Adjustment of Scope	2
2	STRUCTURE AND ROLE	3
3	INTERFACE DEFINITIONS	5
3.1	Global Issues	5
3.1.1	Naming Conventions	5
3.2	Common Interfaces	5
3.2.1	Entities	5
3.2.2	Identifiable	5
3.2.3	Non-functional Properties	6
3.2.4	Identifiers	6
3.3	Helper interfaces	8
3.3.1	Factories	8
3.3.2	Mediateable	10
3.3.3	Ontology Container	10
3.3.4	Mediator Container	10
3.3.5	Exceptions	11
3.4	Ontologies	11
3.4.1	LogicalExpression	12
3.4.2	LogicalExpressionHolder	12
3.4.3	Ontology	12
3.4.4	Concept	14
3.4.5	Instance	16
3.4.6	Relation	17
3.4.7	Function	18
3.4.8	RelationInstance	19
3.4.9	Value	20
3.4.10	Axiom	20
3.5	Goal	20
3.6	Mediators	21
3.6.1	OOMediator	22

---

3.6.2	GGMediator	22
3.6.3	WGMediator	22
3.6.4	WWMediator	23
3.7	Web Services	23
3.7.1	Web Service	23
3.7.2	Capability	24
3.7.3	Interface	25
3.8	Persistence, Import and Export	26
3.8.1	Persistence	26
3.8.2	Parsing	27
3.8.3	URI resolution	28
4	EXAMPLES	31
4.1	Ontology Creation	31
4.2	Web Service Example	34
4.3	DataStore Example	35
5	CONCLUSION	36
	REFERENCES	36

# 1 INTRODUCTION

Web Services (WS) allow for automated invocation of software programs over the WWW. The Semantic Web Services (SWS) use ontologies, and more generally Semantic Web technology, for annotation of web services, which enables their flexible and efficient discovery and composition and facilitates the mediation necessary between the various players. The Web Services Modelling Ontology (WSMO, <http://www.wsmo.org>) is a leading semantic web services description framework. The Web Services Modelling Language (WSML, [deBruijn *et al.* 04]) is a representation language for WSMO.

wsmo4j is an open-source project with two parts:

- WSMO API — application programming interfaces for WSMO, which allow for basic manipulation of WSMO descriptions, e.g. creation, exploration, storage, retrieval, parsing, and serialization;
- wsmo4j — a reference implementation of the WSMO API, including a WSML parser and a file-system-based datastore.

wsmo4j is based on Java (a JDK version higher than 1.4.2 required). The results of the project (both the API and the reference implementation) are available under the LGPL licence<sup>1</sup>. wsmo4j is a successor of the corresponding parts of the SWWS Studio<sup>2</sup>.

Although based on similar developments preceding it, the API is relatively young and can be expected to evolve further considering the dynamic development of WSMO, WSML, and the SWS and the WS areas in general. For these reasons, the strategy for development of documentation and other supporting materials is balanced around the following objectives:

- to allow for its sustainable development and maintenance;
- to make its understanding and adoption by software engineers possible on the basis of reasonable acquaintance efforts;
- to allow for short development cycles, which follow — with minimum delay — the corresponding developments of WSMO and WSML;
- to minimize the waste of effort related to development of complementary resources for software, which is likely to evolve.

This deliverable represents a reference-style documentation of the WSMO API. A considerable part of the document is occupied by source-code listings, integrated within the corresponding sections of its main body. Although a proper layout would require the listings to be provided as annexes, this is not appropriate in this case, as long as those represent an essential part of its content.

---

<sup>1</sup><http://www.opensource.org/licenses/lgpl-license.php>

<sup>2</sup><http://swws.ontotext.com>

The remainder of this chapter provides comments on the versions of the API and links to related resources. Chapter two introduces packages of the API and its role in several other systems. The third chapter represents the core content of the document — it is dedicated to the presentation of the concrete interfaces. Section four presents a few sample usage scenarios for the API and the reference implementation. The final chapter concludes the document and provides guidelines for future and related developments.

## 1.1 Versions and Links

The version of the API documented hereby is rc1 from 16.11.2004, which is compliant with the WSMO specification v1.0, [Roman *et al.* 04], from 20.9.2004. As of the date of editing of this document, the latest version is v. 0.2 from 11.1.2005. The JavaDoc documentation of the latest release can be found at <http://wsmo4j.sourceforge.net/apidocs/index.html>. The changes between versions rc1 and 0.2 are evolutionary, caused mostly by requests from users and bug fixes. A list of changes between the versions can be found at: <http://wsmo4j.sourceforge.net/changes-report.html>.

A notable structural change made in v.0.2 is that the *org.wsmo.io* package has been broken down into the following packages: *org.wsmo.io.parser*, *org.wsmo.io.datastore*, and *org.wsmo.io.locator* packages. As a result of this, some of the JavaDoc and Xref links to it do not work.

The web site of wsmo4j is <http://wsmo4j.sourceforge.net> ; it contains announces, download links, documentation, [mailing lists](#), source-code [xref](#) (an automatically generated hyper-textual representation), and a link to the [SourceForge project](#), with its various services, including: bug and feature request tracking, CVS repository, download statistics, etc.

## 1.2 Adjustment of Scope

The original title of the deliverable was "Description of the mediation function interface to be used by the case studies". However, its title and scope had to be redefined. On one hand, the original definition appeared not to be relevant at the current state of development of the project — no detailed mediation interfaces can be defined at this stage. On the other hand, the WSMO API, as presented hereby, appeared to be of a key significance to all the technical infrastructure developed in DIP.

## 2 STRUCTURE AND ROLE

The API comprises the following packages:

- [org.wsmo.common](#) containing a common functionality that is not specific to any other package (non-functional properties, identifiers, containers, exceptions)
- [org.omwg.ontology](#) containing ontology specific interfaces (ontologies, concepts, instances, relations, axioms, logical expressions, etc.)
- [org.wsmo.goal](#) containing interfaces related to WSMO goal descriptions
- [org.wsmo.mediator](#) containing mediator specific interfaces
- [org.wsmo.service](#) containing web service specific interfaces (web services, capabilities, interfaces, etc.)
- [org.wsmo.io](#) containing interfaces related to parsing and persistence (e.g. export/import of WSMO elements)

One of the major design rationales behind the WSMO API is to ensure the compatibility and interoperability between the software infrastructure related to web services (WS) and the ontology management (OM) within DIP. Its relations to the WS and OM infrastructure are commented in the following paragraphs and visualized on figure 2.1.

The Ontology package is meant to serve as the core of the implementation of the ontology representation and data integration (ORDI<sup>1</sup>) framework, which in itself is the interoperability basis of the ontology management suite developed within workpackage WP2 and the Ontology Management Working Group (OMWG, <http://www.omwg.org>). To reflect this, the package name prefix is *org.omwg.* in contrast to *org.wsmo.*, which is used for the other packages. Further, the Ontology package is not dependent on the WS-specific packages, while the latter depend on it.

wsmo4j also fits in the basis of the WSMO Studio, which is a SWS browser and an integrated development environment (IDE) specified and developed within DIP workpackage WP4 (deliverables D4.4, D4b.5, D4b.11). As already mentioned, WSMO Studio is a successor of the SWWS Studio. It will provide editing functionality and serve as an extensible platform allowing for the independent development of plug-ins for specific tasks (e.g. composition and monitoring).

Figure 2.1 represents some of the dependencies between the packages, components, and systems, indicated with arrows pointing to the dependent component. The stacking of the boxes in most of the cases also implies a dependency of the upper ones on the lower ones. A list of the major dependencies follows:

- package Common does not depend on anything;
- package IO depends on Common;

---

<sup>1</sup>ORDI is specified in D2.2, [Kiryakov *et al.* 04], and will be implemented in deliverable D2.3 of DIP.

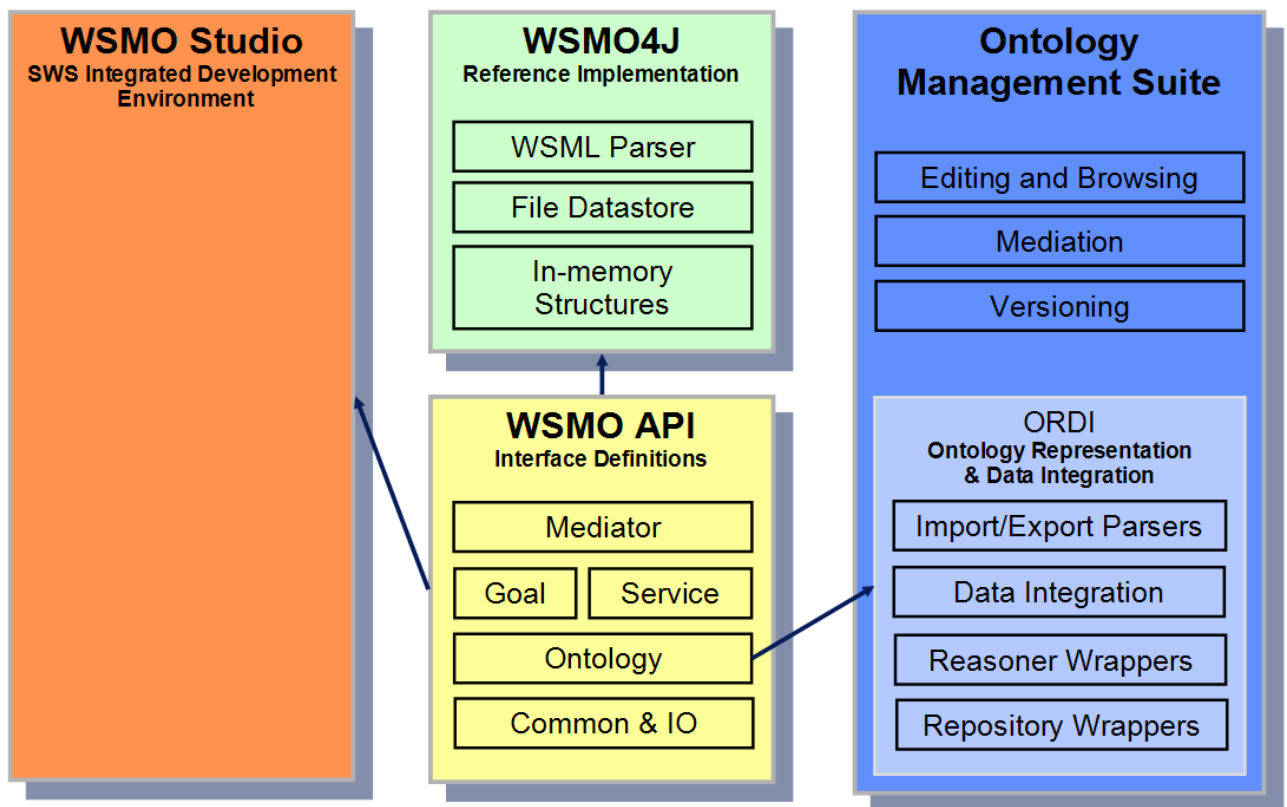


Figure 2.1: wsmo4j role and dependencies

- package Ontology depends on Common and IO;
- packages Goal, Service, and Mediator depend on Ontology, Common, and IO, as well as on one another;
- the wsmo4j reference implementation and WSMO Studio depend on the whole WSMO API;
- ORDI, and the whole Ontology Management Suite, depend on the Ontology package and therefore on the Common and IO packages as well.

## 3 INTERFACE DEFINITIONS

This chapter provides an overview and comments on all the major interfaces defined within the WSMO API.

### 3.1 Global Issues

The introduction to the design principles, conventions, and general interfaces, which are relevant to the whole API, follows in the next sub-sections.

#### 3.1.1 Naming Conventions

wsmo4j follows the standard naming conventions for method signatures, e.g.:

- *set\*()* and *get\*()* for accessing and modifying properties of a class, which are not collections;
- *list\*()*, *find\*()*, *add\*()* and *remove\*()* for accessing a collection, lookup for a specific member of a collection and adding/removing elements from/to a collection.

### 3.2 Common Interfaces

Core interfaces such as *Entity*, *Identifiable*, *Identifier* and the interfaces related to *non-functional properties* are located in the [org.wsmo.common](http://org.wsmo.common) package.

#### 3.2.1 Entities

*Entity* is the top level interface in wsmo4j, which handles any sort of WSMO related entities. It roughly corresponds to the notion of WSMO element, used in [Roman *et al.* 04].

*Additional information:*

- [JavaDoc](#)
- [source code](#)

#### 3.2.2 Identifiable

*Identifiable* represents entities that have an identifier. Note that this interface introduces a slight deviation from the WSMO specification, which does not distinguish identifiable entities - the only way to identify an entity in WSMO is through the non-functional properties such as "*identifier*" from the Dublin Core [dub03].

The *Identifiable* interface has a single method *getIdentifier()* returning an *Identifier*.

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.2.3 Non-functional Properties

Non-functional properties can be associated with some entities. In `wsmo4j` non-functional properties can be attached to all entities with the exception of *Identifier*, *LogicalExpression* and *Value*. The entities that can hold non-functional properties extend the *NFPHolder* interface.

[Listing 3.1](#) presents the *NFPHolder* interface. Note that more than one value can be associated with the same property key.

Listing 3.1: NFPHolder interface

```
public interface NFPHolder {  
  
    Set listNFPValues (URI key)  
        throws SynchronisationException;  
  
    Map listNFPValues ()  
        throws SynchronisationException;  
  
    void addNFPValue (URI key, String value)  
        throws SynchronisationException, InvalidModelException;  
  
    void removeNFPValue (URI key, String value)  
        throws SynchronisationException, InvalidModelException;  
}
```

A set of non-functional keys based on the Dublin Core [[dub03](#)] is available in the *NFP* class.

*Additional information:*

- [JavaDoc for NFPHolder](#)
- [source code for NFPHolder](#)
- [JavaDoc for NFP](#)
- [source code for NFP](#)
- [WSMO definition](#)

### 3.2.4 Identifiers

WSMO defines 4 types of identifiers<sup>1</sup>:

- URI references;

---

<sup>1</sup>See also the [WSMO definition](#)

- literals;
- anonymous identifiers;
- variable names.

In `wsmo4j` identifiers are represented respectively by the [URIRef](#), [Literal](#), [AnonymousID](#) and [VariableName](#) interfaces, which are derived from the generic [Identifier](#) interface.

## URI References

The [URIRef](#) interface presents a URI reference. See the [WSMO definition](#) of identifiers. *Additional information:*

- [JavaDoc](#)
- [source code](#)

## Literals

The [Literal](#) interface presents a WSMO literal. See the [WSMO definition](#) of identifiers. *Additional information:*

- [JavaDoc](#)
- [source code](#)

## Anonymous IDs

The [AnonymousID](#) interface presents a anonymous identifier. See the [WSMO definition](#) of identifiers.

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## Variable Names

The [VariableName](#) interface presents a WSMO variable. See the [WSMO definition](#) of identifiers.

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## 3.3 Helper interfaces

There are three groups of helper interfaces and classes in `wsmo4j` that do not have an exact equivalent in the WSMO domain model, because they represent software design patterns:

- factories;
- containers;
- exceptions.

Those are addressed in the subsequent sections. Note that there are interfaces, such as *NSContainer*, which are not documented here, because they were not present in version `rc1` of `wsmo4j`.

### 3.3.1 Factories

The *WSMOFactory* interface presents a Factory pattern<sup>2</sup> responsible for creating instances of all WSMO entities. All *create\*()* calls accept an *Identifier* parameter for the entity identifier (which in most cases is a *URIRef* but some entities can have anonymous identifiers as well).

[Listing 3.2](#) presents the *WSMOFactory* interfaces.

Listing 3.2: WSMOFactory interface

```
public interface WSMOFactory {  
  
    /* goals */  
    Goal createGoal(URIRef goalID);  
  
    /* web services */  
    WebService createWebService(URIRef wsID);  
  
    Capability createCapability(URIRef capID);  
  
    Interface createInterface(URIRef ifaceID);  
  
    Choreography createChoreography(URIRef chorID);  
  
    Orchestration createOrchestration(URIRef orchID);  
  
    /* ontologies */  
    Ontology createOntology(URIRef ontID);  
  
    Axiom createAxiom(URIRef axiomID);  
}
```

<sup>2</sup><http://c2.com/cgi/wiki?AbstractFactoryPattern>

```
Concept createConcept(Ontology owner, URIRef conceptID)
    throws InvalidModelException;

Instance createInstance(Ontology owner,
                        Identifier instID,
                        Concept concept)
    throws InvalidModelException;

Relation createRelation(Ontology owner, URIRef relID)
    throws InvalidModelException;

Function createFunction(Ontology owner, URIRef functionID)
    throws InvalidModelException;

RelationInstance createRelationInstance(Ontology owner,
                                        Identifier instID,
                                        Relation rel)
    throws InvalidModelException;

Value createValue(Identifier value);

LogicalExpression createLogicalExpression(String value);

/* mediators */
GGMediator createGGMediator(URIRef medID);

WGMediator createWGMediator(URIRef medID);

WWMediator createWWMediator(URIRef medID);

OOMediator createOOMediator(URIRef medID);

/* identifiers */
URIRef createURIRef(String src);

Literal createLiteral(String value, URIRef dataType);

AnonymousID createAnonymousID();
}
```

The *WSMOFactory* is created by the *Factory* class (which is sort of a meta-factory). Note that multiple different implementations of the *WSMOFactory* interface may be created by the meta-factory if desired.

*Additional information:*

- [JavaDoc for WSMOFactory](#)
- [source code for WSMOFactory](#)
- [JavaDoc for Factory](#)

- [source code](#) for *Factory*

### 3.3.2 Mediateable

*Mediateable* is a helper interface that represents an entity that can be either a source or a target component of a *Mediator*. The interface does not expose any methods. Goals, web services, ontologies and mediators are mediateable. In contrast to *Identifiable*, it is not "classified" as a common interface, because there is no such notion in the WSMO specification — it is concrete design decision taken within wsmo4j.

*Additional information:*

- [JavaDoc](#)

### 3.3.3 Ontology Container

The *OntologyContainer* interface presents WSMO elements that import ontologies directly (that is, without the use of OO mediators). Such entities are: goals, web services, capabilities, interfaces, ontologies and mediators.

[Listing 3.3](#) presents the *OntologyContainer* interface. Note that the container holds *references* (e.g. *URIRefs*) to ontologies and not the actual *Ontology* objects.

Listing 3.3: OntologyContainer interface

```
public interface OntologyContainer {  
  
    Set listImportedOntologies()  
        throws SynchronisationException;  
  
    void addImportedOntology(URIRef ontRef)  
        throws SynchronisationException, InvalidModelException;  
  
    void removeImportedOntology(URIRef ontRef)  
        throws SynchronisationException, InvalidModelException;  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.3.4 Mediator Container

The *MediatorContainer* interface presents WSMO elements that make use of mediators. Such entities are: goals, web services, capabilities, interfaces, ontologies and mediators. The used mediators are either *OOMediators* or *GGMediators*

[Listing 3.4](#) presents the *MediatorContainer* interface. Note that the container holds *references* (e.g. *URIRefs*) to mediators and not the actual *OOMediator/GGMediator* objects.

Listing 3.4: MediatorContainer interface

```
public interface MediatorContainer {  
  
    Set listUsedMediators()  
        throws SynchronisationException;  
  
    void addUsedMediator(URIRef medRef)  
        throws SynchronisationException, InvalidModelException;  
  
    void removeUsedMediator(URIRef medRef)  
        throws SynchronisationException, InvalidModelException;  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.3.5 Exceptions

There are two exceptions defined in `wsmo4j`:

- *InvalidModelException* – a **checked** exception indicating an error in the programming logic. All *set\*()*, *add\*()* and *remove\*()* calls may raise such exception. For example removing a concept from an ontology or adding a new postcondition to a goal may result in a contradiction or inconsistency at the logical level.
- *SynchronisationException* – an **unchecked** exception indicating a runtime error. Most methods may raise such an exception because of some runtime error condition that is not caused by the programming logic (f.e. not being able to retrieve an element from a remote repository)

*Additional information:*

- [JavaDoc for InvalidModelException](#)
- [source code for InvalidModelException](#)
- [JavaDoc for SynchronisationException](#)
- [source code for SynchronisationException](#)

## 3.4 Ontologies

The `org.omwg.ontology` package contains ontology specific interfaces (ontologies, concepts, instances, relations, axioms, logical expressions, etc.)

### 3.4.1 LogicalExpression

*LogicalExpression* is the basic interface that presents a WSMO logical expression. The interface has a single *toString()* method that returns the textual representation of the expression. Initialisation of the *LogicalExpression* with the actual content (e.g. logical formula) is implementation specific<sup>3</sup>.

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.4.2 LogicalExpressionHolder

*LogicalExpressionHolder* is a helper interface for entities defined by a *LogicalExpression* (e.g. entities, such as *Concepts*, *Relations*, *Axioms*, etc., that have a *definedBy* property).

[Listing 3.5](#) presents the *LogicalExpressionHolder* interface.

Listing 3.5: LogicalExpressionHolder interface

```
public interface LogicalExpressionHolder {  
  
    LogicalExpression getDefinedBy()  
        throws SynchronisationException;  
  
    void setDefinedBy(LogicalExpression logExpr)  
        throws SynchronisationException, InvalidModelException;  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.4.3 Ontology

The *Ontology* interface represents WSMO ontologies. The ontology may contain concepts, instances, relation instances, etc. An ontology may import other ontologies or use mediators.

[Listing 3.6](#) presents the *Ontology* interface.

<sup>3</sup>see <http://wsmo4j.sourceforge.net/xref/com/ontotext/wsmo4j/ontology/LogicalExpressionImpl.html> for an example

Listing 3.6: Ontology interface

```
public interface Ontology
    extends Entity, Identifiable, NFPHolder, OntologyContainer,
    MediatorContainer, Mediateable, NSContainer {

    /* concepts */
    Set listConcepts()
        throws SynchronisationException;

    Concept findConcept(Identifier id)
        throws SynchronisationException;

    void addConcept(Concept concept)
        throws SynchronisationException, InvalidModelException;

    void removeConcept(Concept concept)
        throws SynchronisationException, InvalidModelException;

    /* relations */
    Set listRelations()
        throws SynchronisationException;

    Relation findRelation(Identifier id)
        throws SynchronisationException;

    void addRelation(Relation relation)
        throws SynchronisationException, InvalidModelException;

    void removeRelation(Relation relation)
        throws SynchronisationException, InvalidModelException;

    /* functions */
    Set listFunctions()
        throws SynchronisationException;

    Function findFunction(Identifier id)
        throws SynchronisationException;

    void addFunction(Function function)
        throws SynchronisationException, InvalidModelException;

    void removeFunction(Function function)
        throws SynchronisationException, InvalidModelException;

    /* instances */
    Set listInstances()
        throws SynchronisationException;

    Instance findInstance(Identifier id)
        throws SynchronisationException;
```

```

void addInstance(Instance instance)
    throws SynchronisationException , InvalidModelException ;

void removeInstance(Instance instance)
    throws SynchronisationException , InvalidModelException ;

/* relation instances */
RelationInstance findRelationInstance(Identifier id)
    throws SynchronisationException ;

void addRelationInstance(RelationInstance instance)
    throws SynchronisationException , InvalidModelException ;

void removeRelationInstance(RelationInstance instance)
    throws SynchronisationException , InvalidModelException ;

/* axioms */
Set listAxioms()
    throws SynchronisationException ;

Axiom findAxiom(Identifier id)
    throws SynchronisationException ;

void addAxiom(Axiom axiom)
    throws SynchronisationException , InvalidModelException ;

void removeAxiom(Axiom axiom)
    throws SynchronisationException , InvalidModelException ;

Set listRelationInstances()
    throws SynchronisationException ;
}

```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.4.4 Concept

The *Concept* interface represents WSMO concepts. The *Concept* is itself a factory for *Attributes*.

[Listing 3.7](#) presents the *Concept* interface.

Listing 3.7: Concept interface

```

public interface Concept
    extends Entity , Identifiable , NFPHolder ,

```

```

LogicalExpressionHolder {

    Set listSuperConcepts()
        throws SynchronisationException;

    void addSuperConcept(Concept superConcept)
        throws SynchronisationException, InvalidModelException;

    void removeSuperConcept(Concept superConcept)
        throws SynchronisationException, InvalidModelException;

    Set listAttributes()
        throws SynchronisationException;

    Attribute findAttribute(Identifier id)
        throws SynchronisationException;

    Attribute createAttribute(Identifier attributeUri)
        throws SynchronisationException, InvalidModelException;

    void removeAttribute(Attribute attribute)
        throws SynchronisationException, InvalidModelException;

    URIRef getOwner()
        throws SynchronisationException;

    void setOwner(URIRef ontology)
        throws SynchronisationException, InvalidModelException;
}

```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

## Attribute

WSMO attributes are associated with concepts. The *Concept* interface is an attribute factory, e.g. it is responsible for creating attributes (via its *CreateAttribute()* method). Note that attributes may be sets and in this case multiple attribute values may be associated with the same attribute.

[Listing 3.8](#) presents the *Attribute* interface.

Listing 3.8: Attribute interface

```

public interface Attribute
    extends Entity, Identifiable, NFPHolder {

```

```
Concept getRange()  
    throws SynchronisationException;  
  
void setRange(Concept aConcept, boolean ofTypeSet)  
    throws SynchronisationException, InvalidModelException;  
  
boolean isOfTypeSet()  
    throws SynchronisationException;  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.4.5 Instance

The *Instance* interface represents instances of concepts. An instance is associated with a single concept and is a part of an ontology.

[Listing 3.9](#) presents the *Instance* interface.

Listing 3.9: Instance interface

```
public interface Instance  
    extends Entity, Identifiable, NFPHolder {  
  
    void setMemberOf(Concept concept)  
        throws SynchronisationException, InvalidModelException;  
  
    Concept getMemberOf()  
        throws SynchronisationException;  
  
    Set listAttributeValues(Attribute attribute)  
        throws SynchronisationException;  
  
    Map listAttributeValues()  
        throws SynchronisationException;  
  
    void addAttributeValue(Attribute key, Value value)  
        throws SynchronisationException, InvalidModelException;  
  
    void removeAttributeValue(Attribute attribute, Value attrVal)  
        throws SynchronisationException, InvalidModelException;  
  
    void removeAttributeValues(Attribute attribute)  
        throws SynchronisationException, InvalidModelException;  
  
    URIRef getOwner()  
        throws SynchronisationException;  
}
```

```
void setOwner(URIRef ontology)
    throws SynchronisationException , InvalidModelException ;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.4.6 Relation

The *Relation* interface represents WSMO relations.

[Listing 3.10](#) presents the *Relation* interface.

Listing 3.10: Relation interface

```
public interface Relation
    extends Entity , Identifiable , NFPHolder ,
    LogicalExpressionHolder {

    Set listSuperRelations ()
        throws SynchronisationException ;

    void addSuperRelation (Relation relation)
        throws SynchronisationException , InvalidModelException ;

    void removeSuperRelation (Relation relation)
        throws SynchronisationException , InvalidModelException ;

    Set listParameters ()
        throws SynchronisationException ;

    Parameter findParameter (Identifier id)
        throws SynchronisationException ;

    Parameter createParameter (Identifier paramURI)
        throws SynchronisationException , InvalidModelException ;

    void removeParameter (Parameter param)
        throws SynchronisationException , InvalidModelException ;

    URIRef getOwner ()
        throws SynchronisationException ;

    void setOwner (URIRef ontology)
        throws SynchronisationException , InvalidModelException ;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

## Parameter

WSMO parameters are associated with relations. The *Relation* interface is a parameter factory, e.g. it is responsible for creating parameters (via its *CreateParameter()* method).

[Listing 3.11](#) presents the *Parameter* interface.

Listing 3.11: Parameter interface

```
public interface Parameter
    extends Entity, Identifiable, NFPHolder {

    Concept getDomain()
        throws SynchronisationException;

    void setDomain(Concept concept)
        throws SynchronisationException, InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.4.7 Function

Functions represent functional relations.

[Listing 3.12](#) presents the *Function* interface.

Listing 3.12: Function interface

```
public interface Function
    extends Relation {

    Concept getRange()
        throws SynchronisationException;

    void setRange(Concept concept)
        throws SynchronisationException, InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.4.8 RelationInstance

The *RelationInstance* interface represents instances of relations. A relation instance is associated with a single relation and is a part of an ontology.

[Listing 3.13](#) presents the *RelationInstance* interface.

Listing 3.13: RelationInstance interface

```
public interface RelationInstance
    extends Entity, Identifiable, NFPHolder {

    void setInstanceOf(Relation concept)
        throws SynchronisationException, InvalidModelException;

    Relation getInstanceOf()
        throws SynchronisationException;

    public Map listParameterValues();

    Value getParameterValue(Parameter parameter)
        throws SynchronisationException;

    void setParameterValue(Parameter key, Value value)
        throws SynchronisationException, InvalidModelException;

    void removeParameterValue(Parameter parameter)
        throws SynchronisationException, InvalidModelException;

    URIRef getOwner()
        throws SynchronisationException;

    void setOwner(URIRef ontology)
        throws SynchronisationException, InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.4.9 Value

The *Value* interface is a placeholder for *Attribute* and *Parameter* values. Values are created from the *WSMOFactory*

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.4.10 Axiom

The *Axiom* interface represents WSMO axioms. The *Axiom* interface extends *Logical-ExpressionHolder*, *Identifiable*, and *NFPHolder* interfaces without adding new methods.

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

## 3.5 Goal

The *Goal* interface represents a WSMO goal (definitions of problems that should be solved by web services).

[Listing 3.14](#) presents the *Goal* interface.

Listing 3.14: Goal interface

```
public interface Goal
    extends Entity, Identifiable, NFPHolder, OntologyContainer,
        MediatorContainer, Mediateable, NSContainer {

    Set listPostConditions()
        throws SynchronisationException;

    void addPostCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    void removePostCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    Set listEffects()
        throws SynchronisationException;

    void addEffect(Axiom axiom)
        throws SynchronisationException, InvalidModelException;
```

```
void removeEffect(Axiom axiom)
    throws SynchronisationException , InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

## 3.6 Mediators

The *Mediator* interface is the top level interface from which all types of mediators are derived (*OOMediator*, *GGMediator*, *WGMediator* and *WWMediator*).

[Listing 3.15](#) presents the *Mediator* interface.

Listing 3.15: Mediator interface

```
public interface Mediator
    extends Entity , Identifiable , NFPHolder , OntologyContainer ,
    Mediateable , NSContainer {

    Set listSourceComponents()
        throws SynchronisationException;

    void addSourceComponent(Mediateable src)
        throws SynchronisationException , InvalidModelException;

    void removeSourceComponent(Mediateable src)
        throws SynchronisationException , InvalidModelException;

    Mediateable getTargetComponent()
        throws SynchronisationException;

    void setTargetComponent(Mediateable target)
        throws SynchronisationException , InvalidModelException;

    Identifier getMediationServiceID()
        throws SynchronisationException;

    void setMediationServiceID(Identifier newServiceID)
        throws SynchronisationException , InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)

- [source code](#)
- [WSMO definition](#)

### 3.6.1 OOMediator

This interface represents a WSMO ooMediator (mediators that import ontologies and resolve possible representation mismatches between ontologies). Note that the WSMO definition restricts the *source* of the mediator to either an ontology or another ooMediator but this restriction cannot be enforced by the java interface, so the respective implementations are responsible for enforcing it (by making the proper verifications)

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.6.2 GGMediator

This interface represents a WSMO ggMediator (mediators that link two goals, e.g. refinement of the source goal into the target goal). Note that the WSMO definition restricts the *source* of the mediator to either a goal or another ggMediator but this restriction cannot be enforced by the java interface, so the respective implementations are responsible for enforcing it (by making the proper verifications).

This mediator may use other mediators, e.g. it is a [MediatorContainer](#)

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.6.3 WGMediator

This interface represents a WSMO wgMediator (mediators that link web service to goals). Note that the WSMO definition restricts the *source* of the mediator to either a web service or another wgMediator but this restriction cannot be enforced by the java interface, so the respective implementations are responsible for enforcing it (by making the proper verifications)

This mediator may use other mediators, e.g. it is a [MediatorContainer](#)

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.6.4 WWMediator

This interface represents a WSMO `wwMediator` (mediators linking two Web Services). Note that the WSMO definition restricts the *source* of the mediator to either a web service or another `wwMediator` but this restriction cannot be enforced by the java interface, so the respective implementations are responsible for enforcing it (by making the proper verifications)

This mediator may use other mediators, e.g. it is a *MediatorContainer*

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## 3.7 Web Services

### 3.7.1 Web Service

The *WebService* interface presents a WSMO web service description. Each *WebService* is associated with exactly one *Capability* and one or more *Interfaces*. A *WebService* may use ontologies and/or mediators (e.g. it is an *OntologyContainer* and a *MediatorContainer*). A *WebService* may be a source/target component of a *Mediator* (e.g. it is *Mediateable*)

[Listing 3.16](#) presents the *WebService* interface.

Listing 3.16: WebService interface

```
public interface WebService
    extends Entity, Identifiable, NFPHolder, OntologyContainer,
           MediatorContainer, Mediateable, NSContainer {

    Capability getCapability()
        throws SynchronisationException;

    void setCapability(Capability cap)
        throws SynchronisationException, InvalidModelException;

    Set listInterfaces()
        throws SynchronisationException;

    void addInterface(Interface iface)
        throws SynchronisationException, InvalidModelException;

    void removeInterface(Interface iface)
        throws SynchronisationException, InvalidModelException;
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.7.2 Capability

The *Capability* interface presents a WSMO capability (definition of a web service functionality). Each *Capability* is associated with a set of *Axioms* (Pre-conditions, Post-conditions, Assumptions and Effects). A *Capability* may use ontologies and/or mediators (e.g. it is an *OntologyContainer* and a *MediatorContainer*).

[Listing 3.17](#) presents the *Capability* interface.

Listing 3.17: Capability interface

```
public interface Capability
    extends Entity, Identifiable, NFPHolder, OntologyContainer,
    MediatorContainer {

    Set listPostConditions()
        throws SynchronisationException;

    void addPostCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    void removePostCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    Set listPreConditions()
        throws SynchronisationException;

    void addPreCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    void removePreCondition(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    Set listEffects()
        throws SynchronisationException;

    void addEffect(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    void removeEffect(Axiom axiom)
        throws SynchronisationException, InvalidModelException;

    Set listAssumptions()
        throws SynchronisationException;

    void addAssumption(Axiom axiom)
```

```

    throws SynchronisationException , InvalidModelException ;

void removeAssumption(Axiom axiom)
    throws SynchronisationException , InvalidModelException ;
}

```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

### 3.7.3 Interface

The *Interface* interface presents a WSMO interface (description of the web service orchestration and choreography). Each *Interface* is associated with a single *Choreography* and a single *Orchestration*. An *Interface* may use ontologies and/or ooMediators (e.g. it is an *OntologyContainer* and a *MediatorContainer*). Note that the ooMediator restriction cannot be enforced on the java interface level and this is the responsibility of the respective implementations.

[Listing 3.18](#) presents the *Interface* interface.

Listing 3.18: Interface definition

```

public interface Interface
    extends Entity , Identifiable , NFPHolder , OntologyContainer ,
    MediatorContainer {

void setChoreography(Choreography chor)
    throws SynchronisationException , InvalidModelException ;

Choreography getChoreography ()
    throws SynchronisationException ;

Orchestration getOrchestration ()
    throws SynchronisationException ;

void setOrchestration(Orchestration orch)
    throws SynchronisationException , InvalidModelException ;
}

```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- [WSMO definition](#)

## Orchestration

This interface represents a web service orchestration (e.g. how a service makes use of other web service or goals in order to achieve its capability)

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## Choreography

This interface represents a web service choreography (e.g. it describes how the service works and how to access the service from the user's perspective)

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## 3.8 Persistence, Import and Export

The *org.wsmo.io* package contains interfaces related to:

- parsing — import and export from/to specific formats (WSML, OWL, etc.);
- persistence — storing and loading WSMO definitions to/from a data store (filesystem, RDBMS, triple store, etc.);
- URI resolution — mapping logical Resource Identifiers into physical locators.

Note that in v.0.2 the *org.wsmo.io* package have been broken down into three packages: [org.wsmo.io.parser](#), [org.wsmo.io.datastore](#), and [org.wsmo.io.locator](#). For this reason many of the JavaDoc and Xref links in this section are broken. Please, refer to the JavaDoc links for the new packages, hyper-linked to the previous sentence.

### 3.8.1 Persistence

#### DataStore

The *DataStore* interface provides an abstraction of a persistent storage that can be used to store and load WSMO descriptions. The actual implementation of the datastore may be based on a filesystem, RDBMS, XML storage, triple store, etc. *DataStores* are initialised by the [DataStoreFactory](#)

[Listing 3.19](#) presents the *DataStore* interface.

Listing 3.19: DataStore definition

```
public interface DataStore {  
    void save(Identifiable object);  
    Identifiable load(URIRef objectID);  
    boolean contains(URIRef objectID);  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## DataStoreFactory

The *DataStoreFactory* is a factory class that initialises *DataStores*. The factory exposes a single method *getDataStore()* that accepts a *java.util.Map* with initialisation parameters for the datastore. At least one parameter should be specified — the class name of the datastore implementation<sup>4</sup>

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## 3.8.2 Parsing

### Parser

The *Parser* interface presents an abstraction of a parser that is able to import and export WSMO descriptions from/to a specific language format (f.e. WSML, OWL, etc.). *Parsers* are initialised by the *ParserFactory*. Note that the stream/buffer used as an input for the parser may contain more than one WSMO definition (f.e. descriptions of multiple web services, ontologies, etc. in the same file).

[Listing 3.20](#) presents the *Parser* interface.

Listing 3.20: Parser definition

```
public interface Parser {  
    Identifiable [] parse(Reader reader)  
        throws IOException;  
    Identifiable [] parse(StringBuffer source);  
}
```

<sup>4</sup>See [http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/datastore/DataStoreFactory.html#DS\\_PROVIDER\\_CLASS](http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/datastore/DataStoreFactory.html#DS_PROVIDER_CLASS) for details

```
void serialize(Identifiable item [], Writer writer)
    throws IOException;

void serialize(Identifiable item [], StringBuffer tagret);
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)
- *the typo in the second parameter ("tagret") of the serialize method is noticed and will be fixed. It is not changed here, in order to preserve the principle of presenting the "authentic" listings.*

## ParserFactory

The *ParserFactory* is a factory class that initialises *Parsers*. The factory exposes a single method *createParser()* that accepts a *java.util.Map* with initialisation parameters for the parser. At least one parameter should be specified — the class name of the parser implementation<sup>5</sup>

*Additional information:*

- [JavaDoc](#)
- [source code](#)

### 3.8.3 URI resolution

The [org.wsmo.io.locator](#) package contains interfaces and classes related to the mapping of logical resource identifiers to physical locators. Entities in WSMO (for example imported ontologies or used mediators) are referred by their logical ID (URI) but this logical identifier does not necessarily present a locator as well (e.g. a way to access the actual resource). This package contains interfaces that allow physical locators to be mapped to logical identifiers in a flexible manner.

A similar mapping problem is also discussed in the definition of the Resource Manager component of the DIP architecture, see [Hauswirth *et al.* 04] and [Kirov & Kiryakov 04]. This part of the WSMO API should be synchronized with the Resource Manager policy, when it gets defined in greater detail.

<sup>5</sup>See [http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/ParserFactory.html#PARSER\\_PROVIDER\\_CLASS](http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/ParserFactory.html#PARSER_PROVIDER_CLASS) for details

## Locator

The *Locator* interface presents a container for identifier-to-locator mappings. In other words, each locator knows how to resolve a set of URIs into physical resources. The *addMapping()* method maps a [URIRef](#) into a java *Object* that holds implementation specific details that are sufficient to access the resource identified by this *URIRef*. The *lookup()* method<sup>6</sup> will retrieve the resource identified by the supplied *URIRef* (the actual way of retrieval, e.g. reading a file, accessing a resource from a URL, reading a resource from a [DataStore](#), etc., is implementation specific).

[Listing 3.21](#) presents the *Locator* interface.

Listing 3.21: Locator definition

```
public interface Locator {  
  
    Identifiable lookup(org.wsmo.common.URIRef uri);  
  
    void addMapping(org.wsmo.common.URIRef uri, Object target);  
  
    void removeMapping(org.wsmo.common.URIRef key);  
}
```

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## LocatorFactory

The *LocatorFactory* is a factory class that initialises *Locators*. The factory exposes a single method *createLocator()* that accepts a *java.util.Map* with initialisation parameters for the locator. At least one parameter should be specified - the class name of the locator implementation<sup>7</sup>

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## LocatorManager

The *LocatorManager* class keeps track of all registered *Locators*. When a URI has to be resolved to a physical resource the *resolveURI()* method of the *LocatorManager*

<sup>6</sup>Note that the *lookup()* method of the *Locator* should not be called directly. Instead, the *LocatorManager.resolveURI()* method should be called and it will delegate to the *lookup()* methods of all registered *Locators*

<sup>7</sup>See [http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/locator/LocatorFactory.html#LOC\\_PROVIDER\\_CLASS](http://wsmo4j.sourceforge.net/apidocs/org/wsmo/io/locator/LocatorFactory.html#LOC_PROVIDER_CLASS) for details

should be called and it will call in turn the *lookup()* methods of all registered *Locators*. *Locators* can be registered/unregistered from the *LocatorManager* (e.g. activated and deactivated) at any time.

*Additional information:*

- [JavaDoc](#)
- [source code](#)

## 4 EXAMPLES

This chapter provides a set of sample programs, which cover basic use cases for the WSMO API.

### 4.1 Ontology Creation

This example shows the creation of a simple ontology, based on the *Locations* ontology from [Stollberg *et al.* 04]

The first step is the **wsmo4j** initialisation. The *Factory* class (which is a meta-factory) is responsible for instantiating a specific implementation of the *WSMOFactory*. In this case the **wsmo4j** implementation will be used (as specified by the *Factory.WSMO\_FACTORY\_PROVIDER\_CLASS* parameter) but other implementations of the WSMO API may be used as well.

```

1 //1. initialise the factory with the wsmo4j provider
2   HashMap factoryParams = new HashMap();
3   factoryParams.put(Factory.WSMO_FACTORY_PROVIDER_CLASS,
4     "com.ontotext.wsmo4j.common.WSMOFactoryImpl");

```

After **wsmo4j** was initialised, the next step is to obtain a *WSMOFactory* that can create WSMO elements. The *WSMOFactory* instance is created from the *Factory* class:

```

5 //2.get a reference to the WSMO Factory
6   WSMOFactory wsmoFactory = Factory.getFactory(factoryParams);

```

Now the *WSMOFactory* can create a new ontology:

```

7 //3. create an ontology
8   Ontology anOntology = wsmoFactory.createOntology(
9     wsmoFactory.createURIRef(
10      "http://wsmo4j.sourceforge.net/examples/" +
11      "ontology01.wsml"));
12 //3.1 setup a namespace
13   anOntology.addNamespace("my",
14     "http://wsmo4j.sourceforge.net/examples/");

```

...and attach some NFPs:

```

15 // 3.2 attach some NFPs
16   anOntology.addNFPValue(NFP.DC_TITLE,
17     "International_Train_Connections_Ontology");
18   anOntology.addNFPValue(NFP.DC_CREATOR, "DERI_International");

```

```

19 anOntology.addNFPValue(NFP.DC_SUBJECT, "Train");
20 anOntology.addNFPValue(NFP.DC_DESCRIPTION,
21     "International_Train_Itineraries");
22 anOntology.addNFPValue(NFP.DC_PUBLISHER, "DERI_International");
23 anOntology.addNFPValue(NFP.DC_CONTRIBUTOR, "Ruben_Lara");
24 anOntology.addNFPValue(NFP.DC_CONTRIBUTOR, "Holger_Lausen");
25 anOntology.addNFPValue(NFP.DC_DATE, "2004-06-28");
26 anOntology.addNFPValue(NFP.DC_TYPE,
27     "http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos");
28 // etc....

```

The new ontology can import existing ontologies and mediators:

```

29 // 3.2 adds some references to mediators and imported ontologies
30 anOntology.addImportedOntology(wsmoFactory.createURIRef(
31     "http://www.wsmo.org/ontologies/dateTime"));
32 anOntology.addNamespace("dt",
33     "http://www.wsmo.org/ontologies/dateTime#");
34
35 anOntology.addUsedMediator(wsmoFactory.createURIRef(
36     "http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/" +
37     "resources/owlPersonMediator.wsml"));

```

Now, concepts and attributes can be added to the ontology:

```

38 //4. add concepts and instances to the ontology
39 // create some concepts into our ontology
40
41 //4.1 use xsd:string as attribute range
42 Concept XSD_STRING = Factory.getFactory().createConcept(null,
43     Factory.getFactory().createURIRef(
44     "http://www.w3.org/2001/XMLSchema#string"));
45
46 //4.2 create the Location concept
47 URIRef uriLocation = anOntology.createURIRef("loc",
48     "location");
49 Concept cLocation = wsmoFactory.createConcept(anOntology,
50     uriLocation);
51
52 //create the Station concept, with Location as super-concept
53 Concept cStation = wsmoFactory.createConcept(anOntology,
54     anOntology.createURIRef("my", "station"));
55 cStation.addSuperConcept(cLocation);
56 cStation.addNFPValue(NFP.DC_DESCRIPTION, "Train_station");
57
58 //4.3 add the Code and LocatedIn attributes to Station
59 Attribute attrCode =
60     cStation.createAttribute(anOntology.createURIRef("code"));
61 attrCode.setRange(XSD_STRING, false);

```

```

62 attrCode.addNFPValue(NFP.DC_DESCRIPTION,
63     "Code_of_the_station");
64
65 Attribute attrLocatedIn = cStation.createAttribute(
66     anOntology.createURIRef("locatedIn"));
67 attrLocatedIn.setRange(cLocation, true);
  
```

Axioms can be added too:

```

68 //4.5 add some axioms to the ontology
69 Axiom axiom = wsmoFactory.createAxiom(
70     anOntology.createURIRef("my", "stationCountry"));
71 axiom.addNFPValue(NFP.DC_DESCRIPTION,
72     "Integrity_constraint:_if_a_station_is_located_in_a_" +
73     "place,_which_is_located_in_a_given_country,_the_" +
74     "country_of_the_station_is_the_same");
75 axiom.setDefinedBy(wsmoFactory.createLogicalExpression(
76     "constraint_?S" +
77     "[locatedIn_hasValue_?L,_country_hasValue_?C]_and_" +
78     "not_?L[country_hasValue_?C].");
79 anOntology.addAxiom(axiom);
  
```

Finally, instances can be added:

```

80 //5. add instances to the ontology
81 Instance instInnsbruck = wsmoFactory.createInstance(anOntology,
82     anOntology.createURIRef("my", "innsbruckHbf"), cStation);
83
84 //5.1 set values to the CODE and LOCATED IN attributes
85 attrCode = instInnsbruck.getMemberOf().findAttribute(
86     anOntology.createURIRef("code"));
87 instInnsbruck.addAttributeValue(attrCode,
88     wsmoFactory.createValue(wsmoFactory.createLiteral("INN",
89     (URIRef)XSD.STRING.getIdentifier())));
90
91 attrLocatedIn = instInnsbruck.getMemberOf().findAttribute(
92     anOntology.createURIRef("locatedIn"));
93 instInnsbruck.addAttributeValue(attrLocatedIn,
94     wsmoFactory.createValue(anOntology.createURIRef("loc",
95     "innsbruck")));
  
```

*Additional information:*

- [full source code](#)

## 4.2 Web Service Example

This example shows the creation of a WSMO service, based on the *Locations* ontology from [Stollberg *et al.* 04].

The first step is the **wsmo4j** initialisation. The *Factory* class (which is a meta-factory) is responsible for instantiating a specific implementation of the *WSMO-Factory*. In this case the default **wsmo4j** implementation will be used (but other implementations of the WSMO API may be used as well as specified in the *Factory.FACTORY\_PROVIDER\_CLASS* initialisation parameter).

```

1 //get the default WSMO factory
2 WSMOFactory wsmoFactory = Factory.getWSMOFactory();

```

...and then create a **capability** describing the web service:

```

3 Capability capability = wsmoFactory.createCapability(
4     newURI("capability"));
5 Axiom axiom1;
6
7 axiom1 = wsmoFactory.createAxiom(newURI("axiom1"));
8 LogicalExpression logExpre =
9     wsmoFactory.createLogicalExpression(
10         "?Buyer_memberOf_po:buyer_and" +
11         "?Trip_memberOf_tc:trainTrip[" +
12         ".....tc:start_hasValue_?Start," +
13         ".....tc:end_hasValue_?End," +
14         ".....tc:departure_hasValue_?Departure" +
15         " ]_and" +
16         "(?Start.locatedIn_=_austria_or_" +
17         "?Start.locatedIn_=_germany)_and" +
18         "(?End.locatedIn_=_austria_or_" +
19         "?End.locatedIn_=_germany)_and" +
20         "dt:after(?Departure,currentDate).");
21 axiom1.setDefinedBy(logExpr);
22 capability.addPreCondition(axiom1);

```

Creation of assumptions, effects, etc. is performed in a similar manner. The specification of the non-functional properties, namespaces, imported ontologies and interfaces is skipped for clarity but it is available in the [complete example](#).

*Additional information:*

- [full source code](#)

## 4.3 DataStore Example

This example shows the serialisation of a WSMO service into a datastore, in this case the datastore is FDS — the default file-system-based datastore provided by **wsmo4j**, but working with other datastore does not require any changes to the code (apart from the datastore initialisation).

The first step is the **wsmo4j** initialisation. The *Factory* class (which is a meta-factory) is responsible for instantiating a specific implementation of the *WSMO-Factory*. In this case the default **wsmo4j** implementation will be used (but other implementations of the WSMO API may be used as well as specified in the *Factory.FACTORY\_PROVIDER\_CLASS* initialisation parameter).

```
1 //get the default WSMO factory
2 WSMOFactory wsmoFactory = Factory.getWSMOFactory();
```

... and then create a simple *WebService*:

```
3 WebService service = ...
4 URIRef serviceID = (URIRef)service.getIdentifier();
```

... then proceed with the datastore initialisation:

```
5 // Fill the parameter map for creating a
6 // FileDataStore instance.
7 HashMap dsParams = new HashMap();
8 dsParams.put(DataStoreFactory.DS_PROVIDER_CLASS,
9             FILESYSTEMDATASTORE_PROVIDER);
10 dsParams.put(FileDataStore.DATA_STORE_LOCATION,
11             "/some/empty_folder/test_datastore");
12
13 // Initialise the data store to use.
14 DataStore ds = DataStoreFactory.getDataStore(dsParams);
```

... now the web service may be stored in the datastore:

```
15 //save service
16 ds.save(service);
```

... and loaded from the datastore:

```
17 //load service
18 WebService service2 = (WebService)ds.load(serviceID);
```

*Additional information:*

- [full source code](#)

## 5 CONCLUSION

This document represents a reference-style documentation of the WSMO API, including a short introduction about its role and structure and complemented by the descriptions of a few samples. WSMO API provides Java interfaces for manipulation of WSMO descriptions, including ontologies. Together with its reference implementation, it is a part of the `wsmo4j` open-source project, which is currently in its *release candidate* stage. The `wsmo4j` reference implementation provides:

- implementations of the interfaces from the packages `Ontology`, `Goal`, `Service`, and `Mediator`, which allow for straightforward in-memory manipulation and modification of the WSMO primitives;
- an implementation of the `Parser` interface for the WSML human-readable syntax, [deBruijn *et al.* 04], which allows parsing (import) and serialization (export) of WSML;
- an implementation of the `Datastore` interface, allowing for easy and reasonably efficient storage and retrieval of WSMO elements;
- an implementation of the `Locator` interface.

`wsmo4j` will be developed further to follow the development of WSMO and WSML, as well as to make it more mature and more efficient and to meet the requirements for its usage in various scenarios and applications. The API will find place in all the technical and use-case workpackages of DIP. Most directly, `wsmo4j` is a fundamental part of the WSMO Studio — the SWS browser and the integrated development environment (IDE) of DIP. As a seed of the ontology representation and data integration framework (ORDI) it will fit in the basis of all ontology management tools and infrastructure. ORDI will further develop the ontology-related interfaces in multiple directions, including the ability to allow for a definition of comprehensive query-answering mechanism and implementations of wrappers of ontology repositories and reasoners. A limited `Parser` implementation for RDF/XML syntax of OWL is planned in the course of development of the ontology management infrastructure. Last but not least, `wsmo4j` is already considered for usage in a number of other projects related to WSMO, among which `KnowledgeWeb` and `InfraWebs`.

**Acknowledgement.** The authors are grateful to the reviewers Alexander Wahler and Joachim Quantz, who were fair and direct enough to enforce the adjustment of the original scope of the deliverable. Even more, they had the patience to give a careful review of the adjusted version.

---

## BIBLIOGRAPHY

[deBruijn *et al.* 04]

J. de Bruijn, H. Lausen, and D. Fensel. The WSML Family of Representation Languages, v0.2. WSML working draft, DERI, Nov 2004. Available at <http://www.wsmo.org/2004/d16/d16.1/v0.2/20041126/>.

[dub03]

Dublin Core Metadata Element Set, Version 1.1: Reference Description. Technical report, Dublin Core Metadata Initiative, Jun 2003. Available at <http://dublincore.org/documents/dces/>.

[Hauswirth *et al.* 04]

M. Hauswirth, R. Schmidt, M. Altenhofen, C. Drumm, C. Bussler, M. Moran, M. Zaremba, L. Vasiliu, J. Quantz, L. Henocque, A. Haller, B. Sakpota, E. Kilgariff, S. Petkov, D. Aiken, E. Oren, M. Ohlendorf, and A. Mocan. Dip architecture. Deliverable d6.2, DIP project, DATF, Dec 2004.

[Kirov & Kiryakov 04]

V. Kirov and A. Kiryakov. Dip component apis. Deliverable d6.3, Ontotext Lab, Sirma AI, Dec 2004.

[Kiryakov *et al.* 04]

A. Kiryakov, V. Kirov, and D. Ognyanov. Ontology representation and data integration (ordi) framework. Deliverable d2.2, Ontotext Lab, Sirma AI, Jul 2004.

[Roman *et al.* 04]

D. Roman, U. Keller, H. Lausen, E. Oren, C. Bussler, M. Kifer, and D. Fensel. Web Service Modeling Ontology, v1.0. WSMO working draft, DERI, Sep 2004. Available at <http://www.wsmo.org/2004/d2/v1.0/20040920/>.

[Stollberg *et al.* 04]

M. Stollberg, H. Lausen, A. Polleres, R. Lara, U. Keller, M. Zaremba, A. Haller, D. Fensel, and M. Kifer. WSMO Use Case 'Virtual Travel Agency', v0.1. WSMO working draft, DERI, Oct 2004. Available at <http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/>.