Open Geospatial Consortium

Publication Date: 2014-05-26

Approval Date: 2013-12-06

Submission Date: 2013-07-30

Reference number of this Document: OGC 11-014r3

External Reference URL for this document: http://www.opengis.net/doc/IS/openmi/2.0

Version: 2.0

Category: OGC® Interface Standard

Editors: Stanislav Vanecek, Roger Moore

OGC® Open Modelling Interface Interface Standard

Warning

This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC Standard
Document subtype:	Interface
Document stage:	Approved
Document language:	English

License

Contents

List of Figures

List of Tables

(i) Abstract

The purpose of the Open Modelling Interface (OpenMI) is to enable the runtime exchange of data between process simulation models and also between models and other modelling tools such as databases and analytical and visualization applications. Its creation has been driven by the need to understand how processes interact and to predict the likely outcomes of those interactions under given conditions. A key design aim has been to bring about interoperability between independently developed modelling components, where those components may originate from any discipline or supplier. The ultimate aim is to transform integrated modelling into an operational tool accessible to all and so open up the potential opportunities created by integrated modelling for innovation and wealth creation.

This document defines the requirements that a component must meet to achieve OpenMI compliance. These comprise: 1) a very thin core set of requirements covering the information and functions needed to establish a link and make an exchange between two components and 2) a set of optional extensions for handling more complex situations.

The document does not describe how to implement the standard. This information together with a range of software tools for creating and running OpenMI-compliant components are provided by the OpenMI Association and third-party software vendors – visit www.openmi.org for further documentation.

(ii) Keywords

OpenMI; Open Modelling Interface; standard; integrated modelling; model integration; model component; interface; model linking; model coupling; data exchange; API.

(iii) Preface

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

(iv) Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium.

- Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia
- OpenMI Association (OA), International

(v) Submitters

All questions regarding this submission should be directed to the editor (see cover page, top right hand corner) or the submitters, who are listed below:

Name	Representing	OGC member	Email
Standa Vanecek	OA and DHI	No	s.vanecek@dhigroup.com
Stef Hummel	OA and Deltares	Yes	stef.hummel@deltares.nl
Adrian Harper	OA and Innovyze	No	adrian.harper@innovyze.com
Gennadii Donchyts	OA and Deltares	Yes	gennadii.donchyts@deltares.nl
Peter Gijsbers	OA and Deltares	Yes	peter.gijsbers@deltares.nl
Jesper Grooss	OA and DHI	No	jgr@dhigroup.com
Johan Hartnack	OA and DHI	No	jnh@dhigroup.com
Rob Knapen	OA and Alterra, Wageningen UR	No	rob.knapen@wur.nl
Onno Roosenschoon	OA and Alterra, Wageningen UR	No	onno.roosenschoon@wur.nl
Peter Schade	OA and BAW	No	peter.schade@baw.de
Jon Goodall	OA and University of South Carolina	No	goodall@engr.sc.edu
Andrea Antonello	OA and Università di Trento/ HydroloGIS	No	andrea.antonello@gmail.com
Jan Gregersen	OA and HydroInform	No	gregersen@hydroinform.com
Simon Cox	OA and CSIRO	Yes	simon.cox@csiro.au
Paul Cleverley	OA and HR Wallingford	Yes	p.cleverley@hrwallingford.com
Robert Szczepanek	OA and Cracow University of Technology / NT		robert@szczepanek.pl
Roger Moore	OA	No	rvm@bgs.ac.uk
David Fortune	OA and XPSolutions	No	David.fortune@xpsolutions.com
Quillon Harpham	OA and HR Wallingford	Yes	q.harpham@hrwallingford.co.uk
Robert Millington	OA and Innovyze	No	robert.millington@innovyze.com
David Lemon	OA and CSIRO	Yes	david.lemon@csiro.au
Andrew Hughes	OA and NERC/BGS	Yes	aghug@bgs.ac.uk
Bert Jagers	OA and Deltares	Yes	bert.jagers@deltares.nl
Michiel Blind	OA and Deltares	Yes	michiel.blind@deltares.nl

1 Scope

1.1 Background

The Open Modelling Interface standard (more usually referred to as the OpenMI) makes possible runtime data exchange between independently developed modelling components. It is an enabling technology that facilitates the simulation of process interactions, where the interactions may either lie within or across the traditional boundaries of scientific disciplines. When the components are models, they may be simple or complex, be based on the same or different concepts and come from the same or different suppliers, whether commercial or open source.

The OpenMI's development has been co-funded by the European Union through two projects (HarmonIT (EC contract: EVK1-CT-2001-00090) and Open-Life (Grant agreement number LIFE06 ENV/UK/000409)) and by the commercial and academic partners of those projects. They were managed by the Natural Environment Research Council (UK) with technical leadership coming from DHI (DK), Deltares (NL) and HR Wallingford (UK), the last with the assistance of its former subsidiary Wallingford Software (now Innovyze (USA)). Future development and the OpenMI's publication as an Open Geospatial Consortium international standard are now the responsibility of the OpenMI Association, which is a legal entity established under Dutch law to take ownership of the Intellectual Property Rights (IPR) relating to the OpenMI – see Annex D for details of the OpenMI Association's IPR policy in relation to the OpenMI standard.

1.2 What is it for?

The standard exists to enable the exchange of data between modelling components at runtime; components being anything from a single constant, e.g. Pi, via measurement devices, functions, models, databases, visualization tools and analytical tools, to complex 3D time-variant modelling applications. In more practical terms, components can be anything necessary to build simulation models or decision support systems (DSS). These enable scientists to improve our understanding of the Earth as a system, help policy-makers to find effective and sustainable responses to societal challenges, help entrepreneurs, consultants and developers to explore, develop and deliver new ideas, products and services and provide facilities for teaching and demonstrating our growing knowledge of a joined-up world.

1.3 What is it?

The OpenMI is an interface standard and consists of a core group of requirements and optional extensions. The core is very thin and defines the requirements for describing components and the data they can exchange, linking and exchanging data; extensions deal with the more sophisticated data exchange requirements, such as those involving interactions over time and space. At the time of writing, there is one extension, the TimeSpace Extension, that forms part of the OpenMI 2.0 Standard. Clause 6 will explain which requirements are core and which are extensions.

The purpose of the core and extension concept is to allow for the future incremental development of the OpenMI. It is also intended to provide a path for its harmonization and integration with other known standards for example, those of the OGC®.

Users can develop their own extensions and are welcome to put these forward to the OpenMI Association for adoption as part of the formal standard. Such proposals should be made to the OpenMI Association Technical Committee (oatc@openmi.org) or be submitted via the OGC®.

1.4 What does it do?

The key feature of the standard is to enable the creation of links between components, where a link matches a variable in one component with its equivalent in another. These variables are referred to as either input or output exchange items. Related to the links are the GetValues and SetValues calls.

These calls enable components to obtain ('get')[1] the values of a variable from one component or change ('set') them in another (at particular locations and/or times should the component compute values over time and/or space). Bi-directional links are also possible (i.e. exchanges between two components can be made in both directions).

Adaptors are used to handle unit transformations and to handle differences in model temporal and spatial resolutions and representations (e.g. vector/raster/non-spatial) – see Figure 1. Adaptors can also be designed to exchange data with model components using alternative interfaces.

Further details are available in What's New in OpenMI 2.0.

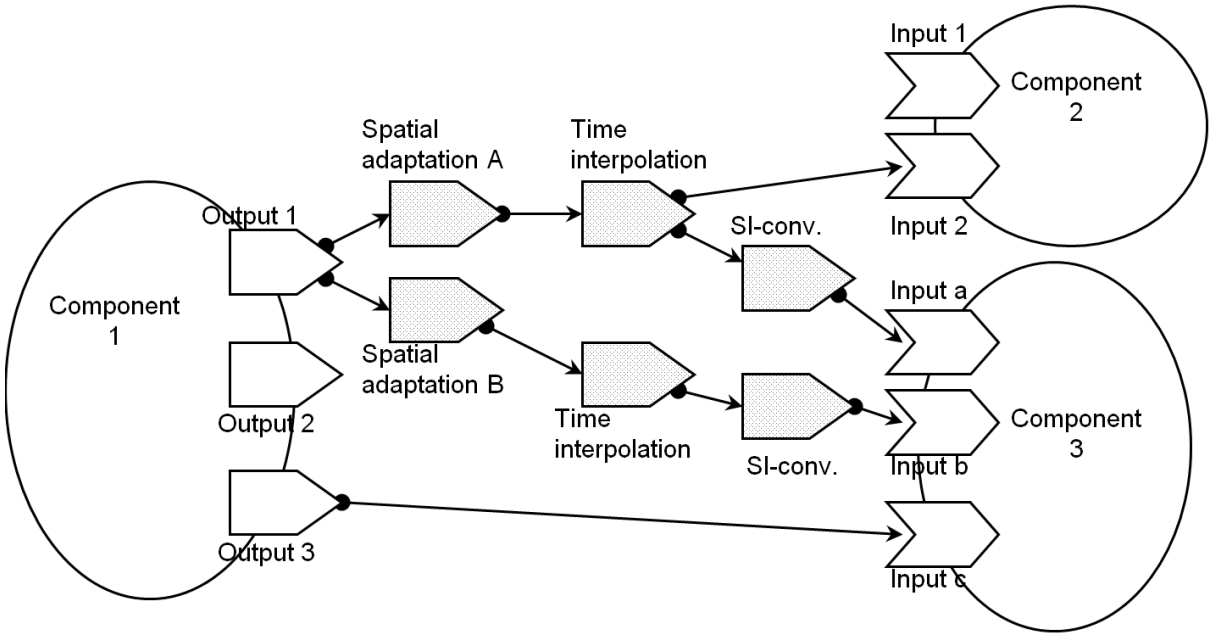


Figure 1 Linkages between components and the use of adaptors

1.5 How is the OpenMI implemented?

The process begins by analysing the component's description and code and identifying the variables[2] whose values are to be 'accepted' from or 'provided' to other modelling components via the OpenMI interface. Some modelling components can accept or provide data for a variable at multiple locations, e.g. points in a grid or nodes in a network. In these cases, decisions are needed as to which of the locations are to be able exchange data with other components. It might be decided to expose only one or two points, e.g. the inlet and/or outlet of a process, or any number of points.

Next, if the model component already exists as a callable entity, then the structure of its code needs to be examined. If the code is not in the form 'initialize, run, finalize', then the code must be rearranged into this pattern. This structure facilitates the handing over of partial control to the outside world. For example, it makes possible the design of interfaces that allow another component to request data and, if necessary, cause the component to run until it is able to provide the requested data. If the component does not exist, then it should be designed to have 'initialize, run, finalize' structure.

[1] 'Get' in the context of the OpenMI covers the activities of 'requesting', 'providing' and 'accepting' the values of exchange items.

[2] These are called 'exchange items'.

Finally, the code is 'wrapped', i.e. given an OpenMI interface. There are two main options for doing this. One is to download the OpenMI software development kit (SDK) from the OpenMI Association website and/or use the available third-party tools which simplify this task. These are available from the web at no charge under an open source licence. The other is to write your own code following the specification and the guidance in the user documentation available on the OpenMI Association website at www.openmi.org. The use of the tools is entirely optional and they do not form part of the standard.

Third-party open source tools are also available for building and running compositions, i.e. a set of linked modelling components. The OA intends, subject to resources, to make a set of aids available should there be no third-party software. The use of the tools is entirely optional and they do not form part of the standard.

1.6 Future development

1.6.1 Background

In 2008, the OpenMI Association (OA) was approached by the OGC® and asked if it would be prepared to publish the OpenMI Standard as one of the OGC® family of standards. Two reasons were given for the inquiry. The first was the increasing importance of the time dimension in the OGC's work and the consequent need to grow the OGC's knowledge and experience in this area; while the second was to address the growing need, common to all disciplines, to link sensors, datasets, models and related modelling components in order to understand and explore process interactions better. Such understanding could then be applied in improved simulation and decision support systems. It was the early perception of this latter need that led David Schell to found the OGC®.

The OpenMI Association welcomed the invitation because accreditation by an internationally-recognized standards organization would remove an important barrier to use of the OpenMI by public bodies and industry. It would also facilitate the uptake and development of integrated modelling, which is the OA's ultimate goal.

As the Association had just agreed a major upgrade of the OpenMI and was well on with the process of implementing that upgrade, the OGC® and the OA agreed that the starting point for their joint work would be the OpenMI Version 2.0. This was formally released at the International Summit on Integrated Environmental Modelling in Washington in December 2010. The period leading up to and just after the release was used to grow the relationship between the OA and the OGC® and, in particular, to work out how they would collaborate.

1.6.2 The OpenMI-OGC Memorandum of Understanding

The outcome of this period was a memorandum of understanding (MoU) on the way forward, the key points of which are outlined below.

The OpenMI and OGC agree to pursue the following items within available resources:

1. OpenMI and OGC will develop procedures to co=operatively share relevant standards documentation necessary for the accomplishment of objectives under this agreement. Document sharing will be limited to designated representatives of OGC and OpenMI, and will be consistent with the policies and procedures of each organization.
2. OpenMI and OGC will work to jointly advance international consensus standards of mutual interest. A first priority will be the facilitation of OpenMI 2.0 as an open international consensus standard under the OGC process framework.
3. OpenMI and OGC will pursue the development of use cases of critical scientific and technical interest to their respective memberships.
4. OpenMI and OGC shall work jointly to develop architectural frameworks, extensions, ontologies, and service and model standards relevant to simulations and computational models, and their interoperability with each other and with OGC services and data models.
5. OpenMI and OGC shall collaborate on the development and conduct of outreach activities to raise awareness within and beyond their respective communities.
6. OpenMI and OGC shall investigate requirements and opportunities, including joint applications for funding, for advancement of model interoperability through joint testbeds, experiments, and pilot activities.
7. The OGC will provide the OpenMI with an Associate Membership in the OGC. In return, the OpenMI will provide OGC with a complimentary membership in OpenMI. OGC membership access will be limited to no more than two members of OpenMI staff and/or Board, and two OpenMI member delegates. OGC's membership in OpenMI will be limited to no more than two OGC staff, and two OGC member delegates.

1.6.3 Outline plan

The combined aim of the OpenMI Association and the OGC® is to transform integrated modelling from being a research tool largely confined to academia to an operational tool readily useable by the public and private sectors and ultimately the public. To do this a number of challenges must be overcome; these are:

- Raising and maintaining awareness and building confidence in integrated modelling
- Establishing a minimum set of standards and ensuring their interoperability
- Achieving a critical mass of linkable modelling components
- Ensuring availability, accessibility and usability of integrated modelling techniques, tools and standards
- Building the skills base
- Establishing an underpinning R & D programme
- Growing take-up by government, industry and the public
- Securing resources

It has been appreciated for many years that a key first step in transforming integrated modelling is to establish a recognized standard interface for data exchange. This first standard will not be all-

encompassing, will be imperfect and may well not be compatible with other standards. However, it will enable independent model developers to create linkable components. This is the purpose in making the OpenMI V2.0 an OGC® standard. It is expected that the initial rate of development will be slow but accelerating as competency and confidence grow. Once a critical mass of linkable components is in place rapid development can be expected.

While that is happening it is proposed to start addressing the challenges through the OGC® Domain Working Groups (DWG), particularly the Workflow DWG. The Workflow DWG is seen as particularly appropriate in that it is trans-disciplinary and therefore provides a neutral forum where people from any discipline can meet to discuss their shared need. A strong reason for partnering with the OGC® is that it already has a wide membership spanning many disciplines drawn from the public, private and academic sectors. Many either have or will have an interest in integrated modelling.

Exploratory work has suggested that it is feasible to design adaptors that will allow the linking of OpenMI-compliant components with components using other interface standards. An immediate task will be to bring together the authors of other relevant interface standards, including those for modelling and sensors, in order to establish if operational versions of such adaptors can be designed and developed. This will further increase the pool of sensors and modelling components that can be linked. A number of the key players are already OGC® members.

Attention can then be turned to developing a detailed plan for addressing the longer-term challenges. Quite detailed outline plans are known to exist among environmental and health modellers and almost certainly exist in other disciplines. These and the MoU above will form the starting point for the OA-OGC plan.

While it would be wrong to anticipate the detail of that plan, it is almost certain that it will contain a review of all the current interface standards and take a forward look at where additional standards would help advance integrated modelling. If, as is almost certain to be the case, modifications or new standards are required, then these will be progressed through the OGC® Standards Working Group system. Where existing standards are used, the model established for incorporating and publishing NetCDF and the OpenMI as OGC® standards will be followed.

1.7 Further reading

The following OpenMI Association documents from The OpenMI Document Series provide further background reading and detailed information on its implementation:

- What's New in OpenMI 2.0 (OpenMI Association, 2010)
- The OpenMI 'in a Nutshell' for the OpenMI (Version 2.0) (OpenMI Association, 2010)
- Scope for the OpenMI (Version 2.0) (Moore, et al., 2010)
- Migrating Models for the OpenMI (Version 2.0) (OpenMI Association, 2010)
- OpenMI Standard 2 Specification for the OpenMI (Version 2.0) (OpenMI Association, 2010)
- OpenMI Standard 2 Reference for the OpenMI (Version 2.0) (OpenMI Association, 2010)

For further details, please see the Bibliography at the end of this document. The cited documents are available through the OpenMI Association website at www.openmi.org.

1.8 Document structure

The document that follows defines the Open Modelling Interface. Clause 2 sets out the rules for conformance. Clauses 3, 4 and 5 cover normative references[3], terms and definitions, and conventions. Clause 6 specifies in detail the classes for creating an Open Modelling Interface. Testing is described in the Annexes.

[3] Normative references are references to other standards upon which this standard builds.

2 Conformance

To conform to this specification and hence be termed 'OpenMI-compliant', a model component ***shall*** implement a set of interfaces that can connect to and interact with the OpenMI component interface "IBaseLinkableComponent" (or its specializations, e.g. the "ITimeSpaceComponent" for time and space dependent components). These interfaces are described in Clause 6 'OpenMI Requirements classes'.

The requirements for compliance are as follows:

1. An OpenMI-compliant component ***shall*** implement the mandatory 'Core' requirements according to specifications provided in this document – see Clause 6.
2. The OpenMI Association provides two additional core interfaces that OpenMI-compliant components may or may not implement: the "IManageState" interface and the "IByteStateConverter" interface. However, if these interfaces are implemented, each method and attribute ***shall*** be implemented according to the instructions given in this document.
3. An OpenMI-compliant component may also implement one or more of the optional OpenMI extensions. If implemented, the implementation ***shall*** conform to the specifications provided in this document.
4. An OpenMI-compliant component including its extensions ***shall***, when compiled, reference the OpenMI.Standard2*.DLLs (.Net Framework 2.0 or higher) or OpenMI-standard2*.jars (java 1.5 or higher). These DLLs/jars contain only interfaces. They are compiled and released by the OpenMI Association. The OpenMI Association's downloadable standard zip file provides the only recognized version of OpenMI Version 2 Standard interfaces.

If an OpenMI-compliant component is to run in one or more of the existing GUIs for linking OpenMI-components, it ***shall*** be associated with an XML file, referred to as the 'OMI file', which conforms to and can be validated with the LinkableComponent.xsd schema – see Annex B.The interfaces are specified in language independent UML and are available in C# and Java. Both C# and Java compilers will ensure that the client code correctly calls the methods and will ensure type safety for the objects obtained from the method call.

Conformance with the specification ***shall*** be checked by applying the abstract tests specified in Annex A of this specification.

3 Normative References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to or revisions of any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

1) OGC 08-015r2, The OpenGIS Abstract Specification, Topic 2: Spatial Referencing by Co-ordinates, 2010-04-27.
   http://portal.opengeospatial.org/files/?artifact_id=39049
2) Documents associated with Unified Modeling Language™ (UML®), August 2011.
   http://www.omg.org/spec/UML/2.4.1/
3) Extensible Markup Language (XML) 1.0 (Fifth Edition), 26 November 2008.
   http://www.w3.org/TR/xml/

4 Terms and Definitions

In the context of the OpenMI and for the purposes of this document, the following terms and definitions apply:

Adaptee
An *output exchange item* whose values are to be adapted. In an OpenMI context 'adapt' means:

- to convert units of measurement
- to aggregate or disaggregate output values over time or space
- to interpolate between output values over time or space
- any other operation to convert the output of a providing *model component* to match the input requirements of the accepting *model component*.

Adaptor
In the OpenMI context, *adaptors* are used to convert the output values of the providing component to the form required by the requesting component. *Adaptors* can be used to handle any or all of unit conversions, spatial transformations and temporal transformations. Users can either use *adaptors* provided by third parties or write their own.

Compliancy information file
An XML file containing information about a component whose compliance you would like to be registered with and published by the OpenMI Association – see Annex C.

Component
See *Model component.*

Composition
A set of linked components set up to simulate a particular scenario.

Element
Data exchange between components in the OpenMI is nearly always related to one or more of a set of elements in a space that may or may not be geo-referenced. For example, these elements might represent towns, pathways in the human body, segments of transmission lines or a cellular representation of the atmosphere or a water body for which values are requested or set – see also *element set* and *version*.

Element set
An element set can comprise any number of elements and the geometry of each element can be represented in any way from a one-dimensional array of points, line segments, poly lines or polygons, through to an array of three-dimensional volumes. As a special case, a cloud of Id-based elements (i.e. they do not have co-ordinates or their co-ordinates are not being used) is also supported. This allows data exchange in contexts where spatial position is unimportant or irrelevant as might arise in an economic model – see also *element* and *version*.

Engine
A synonym for *model component* often used in OpenMI documentation.

Exchange item
A variable exposed by a *model component* through the OpenMI interface, whose values can be provided to or accepted from other *model components*. A specific exchange item will be referred to as being either an *input exchange item* or an *output exchange item*. The terms *quantity* and *quality* are often used in place of the word variable; *quantity* if the values are numeric and *quality* where

values are qualitative: for example, 'hot' and 'cold' for temperature or 'sand', 'clay' and 'peat' for soil type.

Link
When used as a verb, it means to connect an *output exchange item* of one *model component* to an *input exchange item* of another.

Linkable component

A *model component* which implements the OpenMI "IBaseLinkableComponent" interface according to specifications provided in this document.

Model
A *model component* that has read in the data that describes the situation to be simulated, analysed, visualized or otherwise processed: e.g. a generic *model component* for simulating the flow of water down an open channel that has been set up to model the specific behaviour of all or part of the River Rhine under given conditions.

Model application
An entire modelling software system that can be installed on a computer.

Model component
A distinct part of a *model application* where computation takes place – in this context the 'computation' might be simulating a process, analysing or visualizing results or some other process. The computation may be very simple (for example, a linear equation) or extremely complex (for example, a dynamic model of airflow through the nose and mouth to the lungs).

Model linking
The process by which one or more data transfer links are established between the *output exchange items* of one *model component* and the *input exchange items* of another *model component*.

Modified Julian day
Modified Julian day is the Julian Day minus 2400000.5. A Modified Julian Day represents the number of days since midnight 17 November 1858 Universal Time on the Julian calendar. The Modified Julian Day has been selected as a reference, since few models operate in a time horizon before 1858. Any date before 17 November 1858 will be represented as a negative value.

(See RECOMMENDATION ITU-R TF.457-2. USE OF THE MODIFIED JULIAN DATE BY THE STANDARD-FREQUENCY AND TIME-SIGNAL SERVICES – which may be found at: http://www.itu.int/dms_pubrec/itu-r/rec/tf/R-REC-TF.457-2-199710-I!!PDF-E.pdf)

OMI file
The OMI file is an XML file containing metadata about a component – see Annex B. Third-party GUIs can use the information the file contains to simplify the linking of components.

Quality

In the context of the OpenMI, a *quality* usually refers to an exchange item whose values are qualitative **(**for example, 'hot' and 'cold' for temperature) or categorical (for example, 'sand', 'clay' and 'peat' for soil type).

Quantity

In the context of the OpenMI, a *quantity* usually refers to an exchange item whose values are numeric.

UTC
Co-ordinated Universal Time, the primary time standard by which the world regulates clocks and time - see the definition of Co-ordinated Universal Time (UTC) in Clause 2.1.12 of ISO-8601, http://dotat.at/tmp/ISO_8601-2004_E.pdf.

Version

Although most models assume a static spatial world, some advanced models make provision for a dynamic world, i.e. one in which objects move and/or change shape (e.g. waves). To enable the tracking of spatial changes over time, the *Version* number has been introduced. If the version changes then the spatial definition may need to be re-queried – see also *element* and *element set*.

5 Conventions

5.1 Identification

This document and the standard it defines follow the guidelines of the OGC® Naming Authority for all identifiers. The key identifiers are given below.

5.1.1 The external identifier of this OGC® document

This may be found at the top right hand corner of the front page of this document.

5.1.2 The identifier of this OGC® standard

This standard is identified as:

http://www.opengis.net/spec/openmi/2.0

5.1.3 The base URI for requirement and conformance classes

The base URI for all relative URIs that denote requirements and conformance classes is:

http://www.opengis.net/spec/openmi/2.0

5.1.4 The OpenMI XML namespaces

The namespace for OpenMI 2.0 is:

http://www.openmi.org/v2_0

The XML schema definitions (XSD) are located in:

http://www.openmi.org/schemas/v2_0

5.2 Symbols (and abbreviated terms)

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
API	Application programming interface
DSS	Decision support system
DWG	Domain Working Group of the OGC®
Id	Identifier
GUI	Graphical user interface
MoU	Memorandum of understanding
OA	OpenMI Association
OATC	OpenMI Association's Technical Committee
OCT	OATC Conformance Tool
OGC	Open Geospatial Consortium
OMG	Object Modelling Group
OMI	Open Modelling Interface
OpenMI	Open Modelling Interface
SDK	Software development kit
SI	Systeme Internationale (FR) or International System of Units

SWG Standards Working Group of the OGC®

UML Unified Modelling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTC Co-ordinated Universal Time (for most purposes this is the same as Greenwich Mean Time (GMT))

XML Extensible Markup Language

XSD XML Schema Definition

5.3 Unified modelling language (UML)

This document follows the Object Modelling Group's (OMG) recommendations and specifications for UML which may be found at http://www.omg.org/spec/UML/2.4.1/.

Specific UML terms used in this document or used to define other UML terms are:

Attribute A named property of a *type*. The UML diagrams explicitly identify those attributes that are 'read-only'.

Class A description of a set of *objects.*

Enumeration A list of named values used as the range of a particular *attribute* type. For example, Colour = {Red, Green, Blue}.

Event A significant occurrence. An event has a location in time and space and may have *parameters*. In the context of state diagrams, an event is an occurrence that can trigger a *state* transition.

Instance An individual member described by a *type* or a *class*. Usage note: According to a strict interpretation of the metamodel, an individual member of a *type* is an instance and a member of a *class* is an object. In less formal usage it is acceptable to refer to a member of a class as an *object* or an instance.

Interface The use of a *type* to describe the externally visible behaviour of a *class*, *object*, or other entity. In the case of a *class* or *object*, the interface includes the *signatures* of the *methods*.

Member A part of a *type* or *class* denoting either an *attribute* or a method.

Method The implementation of an operation or service that can be requested from an *object* to effect behaviour. A method has a *signature*, which may restrict the actual *parameters* that are possible

Namespace A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning.

Object An entity with a well-defined boundary and identity that encapsulates state and behaviour. State is represented by *attributes* and relationships, behaviour is represented by *methods*. An object is an instance of a *class*.

Parameter The specification of a variable that can be changed, passed or returned. A parameter may include a name, *type* and direction. Parameters are used for *methods*, messages and *events*.

Signature The name and *parameters* of a *method*, message or *event*. *Parameters* may include an optional returned *parameter*.

State A condition or situation during the life of an *object* during which it satisfies some condition, performs some activity or waits for some *event*.

Type A description of a set of instances that share the same *methods*, abstract *attributes* and relationships and semantics. A type may define a *method* specification (such as a *signature*) but not a *method* implementation. Usage note: Type is sometimes used synonymously with *interface*, but it is not an equivalent term.

http://www.iai.uni-bonn.de/III/lehre/vorlesungen/SWT/SS96/Material/UML1.0/glossary.html#

5.4 Extensible Markup Language (XML)

This document follows the W3C recommendation of 26 November 2008 Extensible Markup Language (XML) 1.0 (Fifth Edition) which may be found at http://www.w3.org/TR/xml/.

6 OpenMI Requirements Classes

The OpenMI Standard comprises:

- a set of 'Core' requirements classes
- an extension set of requirements classes referred to as the 'TimeSpace' extension.

Listed below are the required classes and their component requirements for the Core and the TimeSpace extension. All the Core requirements must be implemented except those marked as optional. Implementation of the TimeSpace extension is optional but, if implemented, all its requirements must be met.

The requirements for each are as follows:

Name	Core/ Extension	Mandatory/ Optional	Clause
Requirements Class 1: Component Instantiation			#Clause 6.1
Requirement 1.1: Component Instantiation/Valid XML	Core	Mandatory	#Clause 6.1
Requirements Class 2: Describable Identifiable			#Clause 6.2
Requirement 2.1: Describable Identifiable/Describable	Core	Mandatory	#Clause 6.2
Requirement 2.2: Describable Identifiable/Identifiable	Core	Mandatory	#Clause 6.2
Requirements Class 3: Value Definition			#Clause 6.3
Requirement 3.1: Value Definition/Value Definition	Core	Mandatory	#Clause 6.3
Requirement 3.2: Value Definition/Unit	Core	Mandatory	#Clause 6.3
Requirement 3.3: Value Definition/Quantity	Core	Mandatory	#Clause 6.3
Requirement 3.4: Value Definition/Quality	Core	Mandatory	#Clause 6.3
Requirement 3.5: Value Definition/Category	Core	Mandatory	#Clause 6.3
Requirement 3.6: Value Definition/Dimension	Core	Mandatory	#Clause 6.3
Requirement 3.7: Value Definition/Dimension Base	Core	Mandatory	#Clause 6.3
Requirements Class 4: Spatial Definition			#Clause 6.4
Requirement 4.1: Spatial Definition/Spatial Definition	TimeSpace	Mandatory	#Clause 6.4
Requirement 4.2: Spatial Definition/Element Set	TimeSpace	Mandatory	#Clause 6.4
Requirement 4.3: Spatial Definition/Element Type	TimeSpace	Mandatory	#Clause 6.4
Requirements Class 5: Temporal Definition			#Clause 6.5
Requirement 5.1: Temporal Definition/Time	TimeSpace	Mandatory	#Clause 6.5
Requirement 5.2: Temporal Definition/Time Set	TimeSpace	Mandatory	#Clause 6.5
Requirements Class 6: Value Set			#Clause 6.6
Requirement 6.1: Value Set/Base Value Set	Core	Mandatory	#Clause 6.6
Requirement 6.2: Value Set/Time Space Value Set	TimeSpace	Mandatory	#Clause 6.6
Requirements Class 7: Argument			#Clause 6.7
Requirement 7.1: Argument/Argument	Core	Mandatory	#Clause 6.7

Name	Core/ Extension	Mandatory/ Optional	Clause
Requirements Class 8: Linkable Component Status			#Clause 6.8
Requirement 8.1: Linkable Component Status/Linkable Component Status	Core	Mandatory	#Clause 6.8
Requirement 8.2: Linkable Component Status/Linkable Component Behaviour	Core	Mandatory	#Clause 6.8
Requirement 8.3: Linkable Component Status/Event Status Changed	Core	Mandatory	#Clause 6.8
Requirements Class 9: Exchange Item			#Clause 6.9
Requirement 9.1: Exchange Item/Base Exchange Item	Core	Mandatory	#Clause 6.9
Requirement 9.2: Exchange Item/Base Input	Core	Mandatory	#Clause 6.9
Requirement 9.3: Exchange Item/Base Output	Core	Mandatory	#Clause 6.9
Requirement 9.4: Exchange Item/Time Space Exchange Item	TimeSpace	Mandatory	#Clause 6.9
Requirement 9.5: Exchange Item/Time Space Output	TimeSpace	Mandatory	#Clause 6.9
Requirement 9.6: Exchange Item/Time Space Input	TimeSpace	Mandatory	#Clause 6.9
Requirement 9.7: Exchange Item/Event Exchange Item Changed	Core	Mandatory	#Clause 6.9
Requirements Class 10: Adapted Output			#Clause 6.10
Requirement 10.1: Adapted Output/Base Adapted Output	Core	Mandatory	#Clause 6.10
Requirement 10.2: Adapted Output/Times Space Adapted Output	TimeSpace	Mandatory	#Clause 6.10
Requirement 10.3: Adapted Output/Adapted Output Factory	Core	Mandatory	#Clause 6.10
Requirements Class 11: Manage State			#Clause 6.11
Requirement 11.1: Manage State/Manage State	Core	Optional	#Clause 6.11
Requirement 11.2: Manage State/Byte State Converter	Core	Optional but requires the implement-ation of the Manage State Interface	#Clause 6.11
Requirements Class 12: Linkable Component			#Clause 6.12
Requirement 12.1: Linkable Component/Base Linkable Component	Core	Mandatory	#Clause 6.12
Requirement 12.2: Linkable Component/Time Space Component	TimeSpace	Mandatory	#Clause 6.12
Requirement 12.3: Linkable Component/Time Extension	TimeSpace	Mandatory	#Clause 6.12

In the following clauses the requirement class descriptions are grouped according to the aspect of the OpenMI to which they relate. Each description begins with a general introduction, which includes a table of the URIs for the requirement class and the specific requirements relating to it. This is followed by a UML diagram presenting the dependencies between the classes. Where appropriate, the UML diagram is supported by one or more tables describing the members (methods, attributes and events) of any interfaces. The description concludes with a table listing the specific requirements that must be met for compliance to be achieved.

Note: In so far as it is possible, the interface descriptions that follow are language independent. However, the UML diagrams and the tables describing the interface members are derived from diagrams and tables generated by program from the C# implementation. A consequence of this is

that the description uses the concept of an 'attribute'. Some languages use the term 'property' and as the terms attribute and property are equivalent in this context, there is no problem. Other languages have no attribute concept. However, in these cases, the same functionality can be achieved through the use of methods or functions. Thus, an attribute "Caption" can be represented by two methods, "GetCaption()", which returns the caption value as a string and "SetCaption (string)" which assigns the value in the string to the caption. If the attribute is 'read-only' then no 'Set' method is required.

Developers may find it helpful to view the supported implementations of the standard available in C# and Java and accessible via the OpenMI Association's website at www.openmi.org.

6.1 Component instantiation

An OpenMI Linkable Component is instantiated, for example, by loading a .NET assembly and creating an instance of a class that is incorporated into that assembly. The information on the assembly to be loaded and the class to be instantiated is specified in a registration file called the OMI file, which can be located anywhere on disk.

This file also holds the arguments that should be provided to the component when it is initialized.

In addition to its interfaces, the OpenMI Standard therefore also defines an XML Schema Definition (xsd) for the OMI file. Figure 2 provides a graphical view of the file structure according to the XML Schema Definition. The full Schema Definition for the OMI file can be found in Annex B.

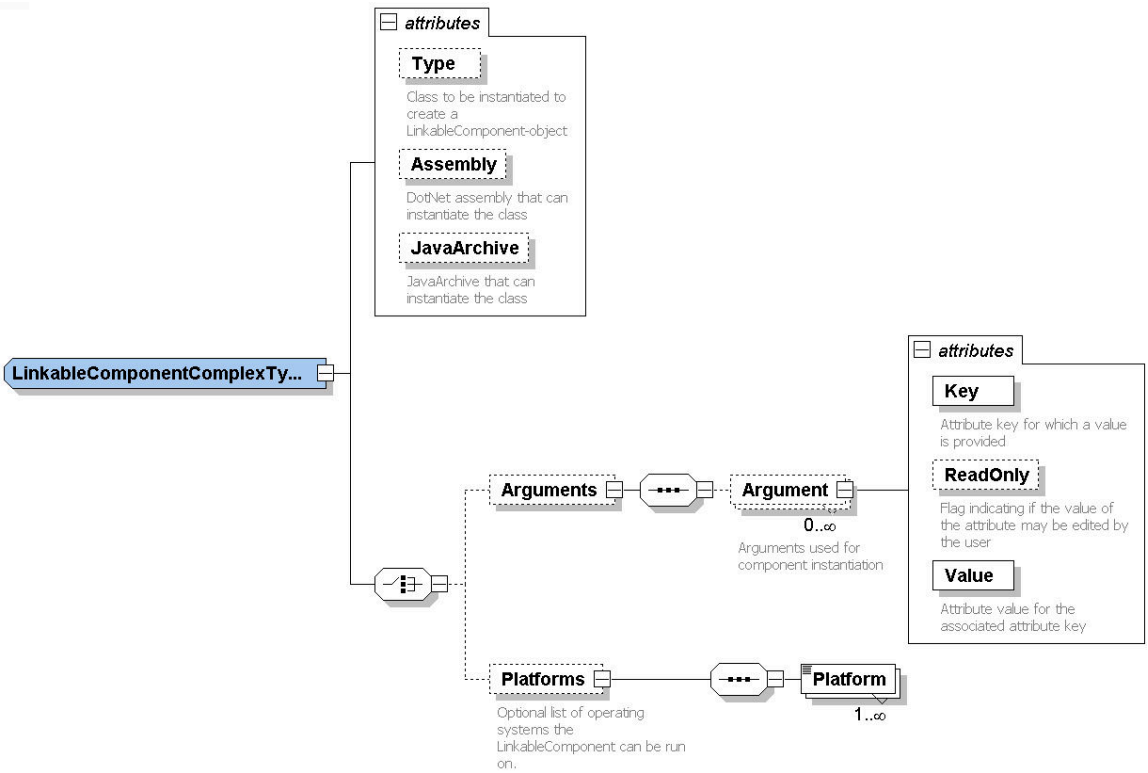


Figure 2 A graphical view of the OMI file structure

Requirements Class 1: Component Instantiation	
/req/component-instantiation	
Target type	OpenMI Component
Dependency	
Requirement 1.1	/req/component-instantiation/ValidXML

6.1.1 Requirements for Component Instantiation

Requirement 1.1: Component Instantiation/Valid XML
/req/component-instantiation/ValidXML
An OpenMI component ***shall*** be described by a valid XML document that describes the arguments to be provided when the component is instantiated and initialized. 'Valid' means that the XML document should adhere to the XML schema definition as defined in http://www.openmi.org/schemas/v2_0/LinkableComponent.xsd, a copy of which can be found in Annex B.

6.2 The Describable and Identifiable interfaces

Many of the OpenMI interfaces describe entities to which users will need to refer through the user interface. Therefore, these entities need a caption and a more detailed description where a caption cannot provide all the information needed.

In addition, some interfaces describe an entity whose specific instances need to be referenced and these therefore also require descriptors and identifiers. An example of their use might be in establishing and storing a link between two components by pairing the identifier of a specific output exchange item of the providing component with a specific input exchange item of the accepting component.

To ensure consistency in the identification and description of all kinds of entity instances, two interfaces are provided: "IDescribable" and, derived from that, "IIdentifiable" – see Figure 3. The majority of the OpenMI Standard Version 2.0 interfaces are derived from one of these interfaces.

Requirements Class 2: Describable Identifiable	
/req/describable-identifiable	
Target type	OpenMI component
Dependency	
Requirement 2.1	/req/describable-identifiable/IDescribable

Requirement 2.2	/req/describable-identifiable/IIdentifiable

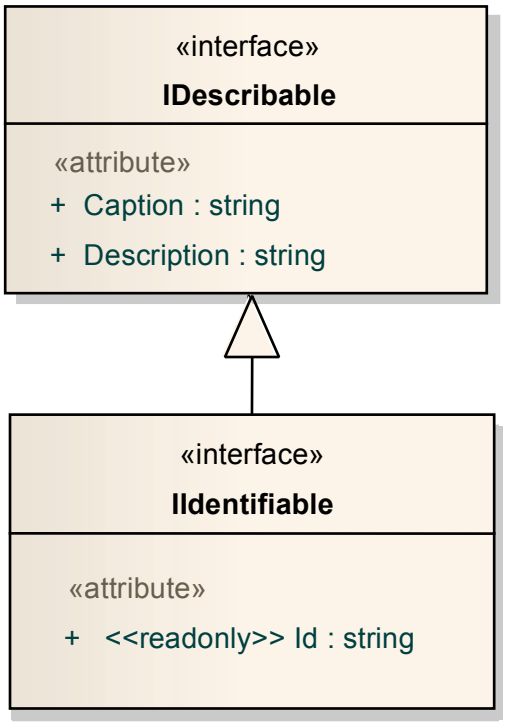


Figure 3 UML Diagram for IDescribable and IIdentifiable

6.2.1 Attributes and methods for Describable and Identifiable interfaces

Table 1 Members of the IDescribable interface

Member	Notes	Parameters
Caption string Public	Sets and returns the value of the "Caption" attribute as a string (not to be used as an identifier).	
Description string Public	Sets and returns additional descriptive information about the entity.	

Table 2 Members of the IIdentifiable interface

Member	Notes	Parameters
Id string Public	Returns the value of the "Id" attribute as a string. The Id must be unique within its context but does not need to be globally unique: e.g. the Id of an input exchange item must be unique in the list of inputs of an "IBaseLinkableComponent", but an identical Id might be used by an exchange	

Member	Notes	Parameters
	item of another "IBaseLinkableComponent".	

6.2.2 Requirements for Describable and Identifiable interfaces

Requirement 2.1: Describable Identifiable/Describable
/req/describable-identifiable/IDescribable
An OpenMI class that represents a described entity ***shall*** implement the IDescribable interface based on the definition in Figure 3 and Table 1.

Requirement 2.2: Describable Identifiable/Identifiable
/req/describable-identifiable/IIdentifiable
An OpenMI class that represents an identifiable entity ***shall*** implement the IIdentifiable interface based on the definition in Figure 3 and Table 2.

6.3 The Value Definition interfaces

The OpenMI leaves the modeller free to decide the names used to label exchange items. It does, however, require some minimal information to be provided about each exchange item, partly to reduce the chance of erroneous links being made and partly to check that unit conversions are provided where the output units of the providing component are different to those of the accepting component.

Figure 4 illustrates the information required and the structure in which the information is held.

Requirements Class 3: Value Definition	
/req/value-definition	
Target type	OpenMI component
Dependency	/req/describable-identifiable/IDescribable
Requirement 3.1	/req/value-definition/IValueDefinition
Requirement 3.2	/req/value-definition/IUnit
Requirement 3.3	/req/value-definition/IQuantity
Requirement 3.4	/req/value-definition/IQuality
Requirement 3.5	/req/value-definition/ICategory
Requirement 3.6	/req/value-definition/IDimension
Requirement 3.7	/req/value-definition/IDimensionBase

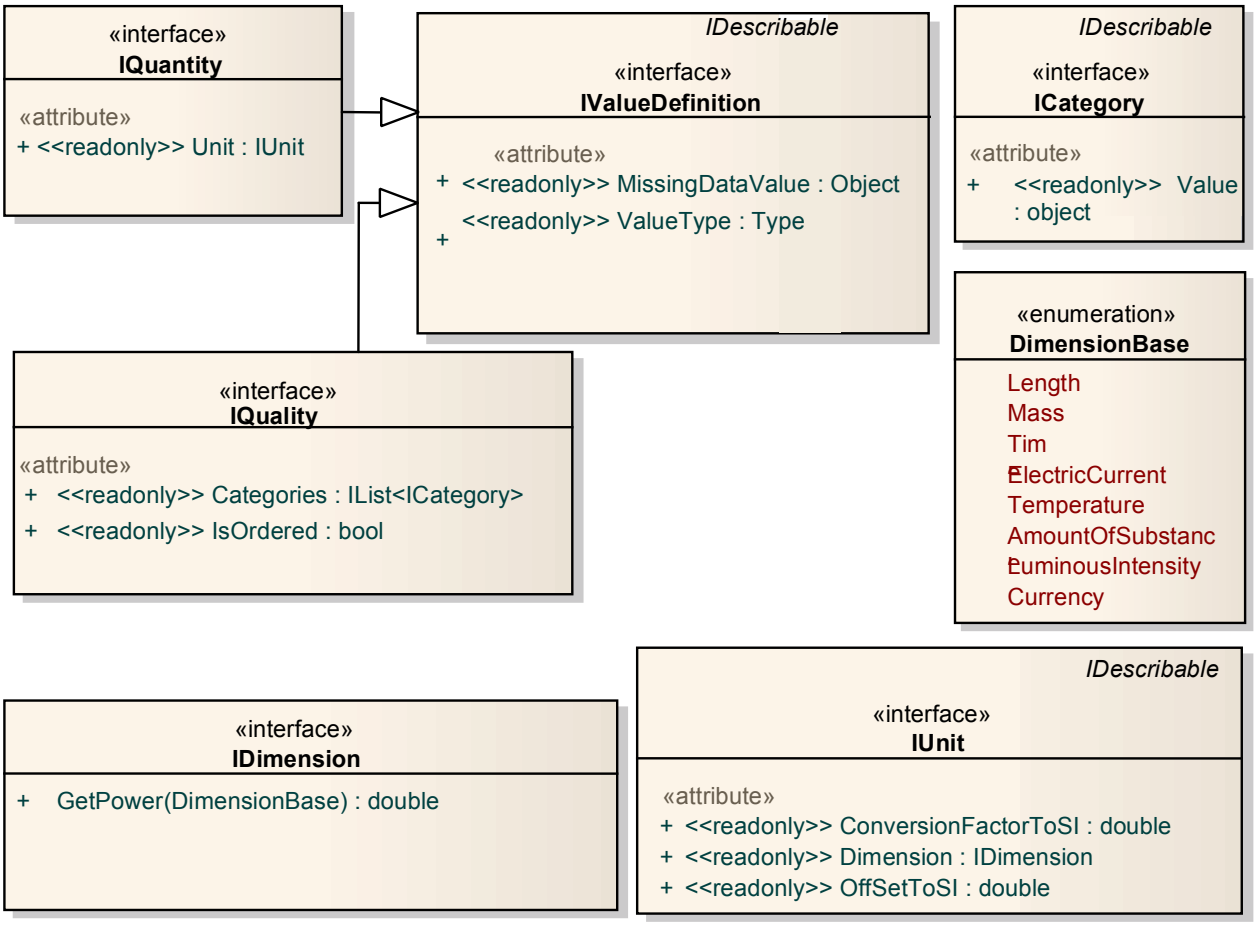


Figure 4 UML Diagram for Value Definition

The "IValueDefinition" interface establishes the data type of a specific exchange item's values and how those values will be represented when missing, e.g. by '-9999'. A value definition is an IDescribable and therefore has a caption, for example 'Flow', and a description (more extensive information for correct interpretation of the caption and other information such as units, etc.)

The "ValueType" attribute indicates the type of object returned when retrieving one of the values of the set, i.e. the value for a certain element and time. Typical values of the "ValueType" attribute will be "double", "bool", "string" or a dedicated class type. Further details will follow below in the descriptions of the "IQuantity" and "IQuality" interfaces.

One of the inheritors of the"IValueDefinition" interface is the "IQuantity" interface. This interface applies to exchange items whose values are numerical and have a "ValueType" of, for example, "double", "int", "boolean" or a class type. It also establishes the units of measurement in which values of the exchange item will be accepted or returned.

The "IUnit" interface provides additional information for checking that linked quantities have the same dimensions and unit conversion. For a given value v of a specific exchange item variable, the conversion to the SI value s can be made using the following equation:

s = Unit.ConversionFactorToSI() * v + Unit.GetOffsetToSI()

The "IDimension" interface has been defined to enable physical dimension[4] checks between quantities. A dimension is expressed as a combination of the basic dimensions, derived from the SI system[5]. A minor extension has been made for currencies. Table 3 illustrates the base quantities/dimensions and the associated SI units. The "IUnit" interface provides a method to obtain the power for each base dimension of a specific exchange item. For example, a discharge expressed in units of m^3/s has dimension $Length^3Time^{-1}$ for which the method would return 3 and -1.

A further method checks if the dimensions of two exchange items are equal.

Table 3 Base units in the OpenMI (derived from SI with an extension for currency)

Base quantity or dimension	SI base unit	Symbol
Length	metre (or meter (US English))	m
Mass	kilogram	kg
Time	second	s
Electric current	ampere	A
Temperature	kelvin	K
Amount of substance	mole	mol
Luminous intensity	candela	cd
Currency[6]	Euro	E

Note that some units are dimensionless, represent logarithmic scales or present other problems when expressed in SI. In such cases, extra attention should be paid to the descriptive part of the unit, to ensure that a user defining a link has a proper understanding of the meaning of the values for the given exchange item.

The other inheritor of the "IValueDefinition" interface is the "IQuality" interface and it applies to exchange items whose values are qualitative. The "IQuality" interface specifies the list of categories used to record the state of the variable: for example, 'hot' and 'cold' for temperature or 'sand', 'clay' and 'peat' for soil type. The "ValueType" for these values will be "string".

The "IsOrdered" attribute indicates whether or not an ordering can be recognized in the categories. If "IsOrdered" is True, one category represents a 'higher' value than another category. The sequence 'very light', 'light', 'heavy', 'very heavy' is an example of an ordered categorization.

[4] A dimension describes the type of thing being measured, without specifying the magnitude. Thus the centimetre, kilometre, inch and foot all have dimensions of length.

[5] More information on the SI system can be found at the National Institute of Standards and Technology (http://physics.nist.gov/cuu/Units/).

[6] Currency has no base quantity in the SI system. Note that currency has conversion units that may vary over time.

6.3.1 Attributes and methods for Value Definition interfaces

Table 4 Members of the IValueDefinition interface

Member	Notes	Parameters
MissingDataValue Object Public	Returns the value of the "MissingDataValue" attribute representing data that are missing.	
ValueType Type Public	Returns the object types of values that will be available in the "IBaseValueSet" that is returned by the "Values" attribute and the "GetValues()" method of the "IBaseExchangeItem".	

Table 5 Members of the IUnit interface

Member	Notes	Parameters
ConversionFactorToSI double Public	Returns the conversion factor to SI units 'A' in the expression: SI-value = **A** * quantity-value + B	
Dimension IDimension Public	Returns the unit's dimensions.	
OffSetToSI double Public	Returns the offset to SI units 'B' in the expression: SI-value = A * quantity-value + **B**	

Table 6 Members of the IQuantity interface

Member	Notes	Parameters
Unit IUnit Public	Returns the unit of measurement in which the quantity's values are expressed.	

Table 7 Members of the IQuality interface

Member	Notes	Parameters
Categories IList<ICategory> Public	Returns a list of the possible category values allowed for this quality. If the categories are not ordered, the list contains the category values in an unspecified order. When it is ordered the list contains the category values in sequence.	

Member	Notes	Parameters
IsOrdered bool Public	Returns a boolean indicating whether or not the quality is defined by an ordered set of category values.	

Table 8 Members of the ICategory interface

Member	Notes	Parameters
Value Object Public	Returns the value for this category. For example "blue" from the allowed set of values "red", "green" and "blue".	

Table 9 Members of the IDimension interface

Member	Notes	Parameters
GetPower() double Public	Returns the power for the requested dimension. For example, for a quantity such as flow, which may have the units m^3/s, the "GetPower()" method must work as follows: myDimension.GetPower(DimensionBase.AmountOfSubstance) → returns 0 myDimension.GetPower(DimensionBase.Currency) → returns 0 myDimension.GetPower(DimensionBase.ElectricCurrent) → returns 0 myDimension.GetPower(DimensionBase.Length) → returns 3 myDimension.GetPower(DimensionBase.LuminousIntensity) → returns 0 myDimension.GetPower(DimensionBase.Mass) → returns 0 myDimension.GetPower(DimensionBase.Temperature) → returns 0 myDimension.GetPower(DimensionBase.Time) → returns -1	**DimensionBase**[in] **dimensionBase**

Table 10 The DimensionBase enumeration

DimensionBase	Notes
Length Public	Base dimension length.
Mass Public	Base dimension mass.
Time Public	Base dimension time.
ElectricCurrent Public	Base dimension electric current.
Temperature Public	Base dimension temperature.
AmountOfSubstance Public	Base dimension amount of substance.
LuminousIntensity Public	Base dimension luminous intensity.
Currency Public	Base dimension currency.

6.3.2 Requirements for Value Definition interfaces

Requirement 3.1: Value Definition/Value Definition
/req/value-definition/IValueDefinition
An OpenMI class that represents a definition of values that can be exchanged between OpenMI linkable components ***shall*** be derived from the IValueDefinition interface based on the definition in Figure 4 and Table 4 and ***shall*** implement it.

Requirement 3.2: Value Definition/Unit
/req/value-definition/IUnit
An OpenMI class that represents a unit of a quantity ***shall*** implement the IUnit interface based on the definition in Figure 4 and Table 5.

Requirement 3.3: Value Definition/Quantity
/req/value-definition/IQuantity
An OpenMI class that represents a unit of a quantity ***shall*** implement the "IQuantity" interface based on the definition in Figure 4 and Table 6.

Requirement 3.4: Value Definition/Quality
/req/value-definition/IQuality
An OpenMI class that represents a quality ***shall*** implement the IQuality interface based on the definition in Figure 4 and Table 7.

Requirement 3.5: Value Definition/Category
/req/value-definition/ICategory
An OpenMI class that represents a category item of a quality ***shall*** implement the ICategory interface based on the definition in Figure 4 and Table 8.

Requirement 3.6: Value Definition/Dimension
/req/value-definition/IDimension
An OpenMI class that represents the dimension of a unit ***shall*** implement the IDimension interface based on the definition in Figure 4 and Table 9, and ***shall*** express the dimension as a combination of basic dimensions derived from the SI system and defined in the DimensionBase enumeration.

Requirement 3.7: Value Definition/Dimension Base
/req/value-definition/DimensionBase
An OpenMI class that represents the dimension of a unit ***shall*** implement the DimensionBase enumeration as specified in Table 10 and described in Table 3.

6.4 The Spatial Definition interfaces

Many modelling components record, simulate analyse or visualize the behaviour not just of a single variable at a single location but the behaviour of many variables for many elements of the system. Data exchange requests in the OpenMI are nearly always directed at one or more of such elements in the providing component.

Elements will usually be spread over 1, 2 or 3 dimensional space. Often but not always the space will be geographic and the location, shape and extent of each element will be defined by geographic co-

ordinates. M co-ordinates are also supported which allow for linear referencing along a line, for example, distance along a segment of river or road. On other occasions there may be no spatial aspect to the model and values are computed for a set of Id-based elements. Examples of elements might be a town, a section of an artery or nerve, a reach of river or a segment of road, a cell or grid intersection in a climate model or a particular animal or person. The collective term used to refer to all the elements represented in a modelling component is "element set".

An element set can comprise any number of elements and the geometry of the elements can be represented in any way from a one-dimensional array of points, line segments, poly lines or polygons, through to an array of three-dimensional volumes. As a special case, a cloud of Id-based elements (i.e. they do not have co-ordinates or their co-ordinates are not being used) is also supported. This allows data exchange in contexts where spatial position is unimportant or irrelevant as might arise in an economic model.

As will be explained further below, although the geometry of most current models remains the same during a model run, the OpenMI makes provision for the possibility that the location, shape and extent of each element may change over time.

Requirements Class 4: Spatial Definition	
/req/spatial-definition	
Target type	OpenMI component
Dependency	
Requirement 4.1	/req/spatial-definition/ISpatialDefinition
Requirement 4.2	/req/spatial-definition/IElementSet
Requirement 4.3	/req/spatial-definition/IElemetType

The "ISpatialDefinition" interface is the general spatial construct of which all other spatial constructions are extensions – see Figure 5. It makes available the number of elements in the element set, the spatial reference system used for defining locations and the version of each element when these are dynamic. Although most models assume a static spatial world, some advanced models may make provision for a dynamic world, i.e. one in which objects move and/or change shape (for example, waves). To enable the tracking of spatial changes over time, the "Version" number has been introduced into the "ISpatialDefinition". If the version changes then the spatial definition may need to be re-queried.

Of the extensions, the one most likely to be used is the "IElementSet"[7]. The "IElementSet" interface has been defined to describe, in a finite element sense, the location where each exchange item value applies. To interpret the co-ordinates correctly, it is necessary to obtain the spatial reference system to which they relate from the "ISpatialDefinition" interface. Unless the spatial reference

[7] In previous versions of the standard, the “IElementSet” was the only spatial construction, and all other spatial constructions had to be wrapped into it, whereas in the current version the “IElementSet” interface is an extension of the “ISpatialDefinition” interface.

system specifies differently, the X and Y axes are assumed to lie in the horizontal plane and the Z axis is vertical.

Note that the "IElementSet" interface can be used to query the geometric description of a model schematization, for example the locations of sampling points or a river network, but this description may not necessarily provide all the topological knowledge on inter-element connections.

The elements in an element set are identified by a string Id, and are therefore IIdentifiables. Where practicable, the element Id's should be designed to be meaningful to end users. The element set does not need to be identifiable, because it is always attached to an input or output exchange item that will have an identity. However, the element set is an IDescribable and therefore can have a caption and a description; these can be helpful to the end user in composing configurations, i.e. building a linked model.

The properties of an element (its vertices and/or faces) are obtained using an integer index for example, "elementIndex", "faceIndex" or "vertexIndex". This functionality has been introduced because an element set is basically an ordered list of elements, an element may have faces and an element (or a face) is an ordered list of vertices. The integer index indicates the location of the element/vertex in the array list.

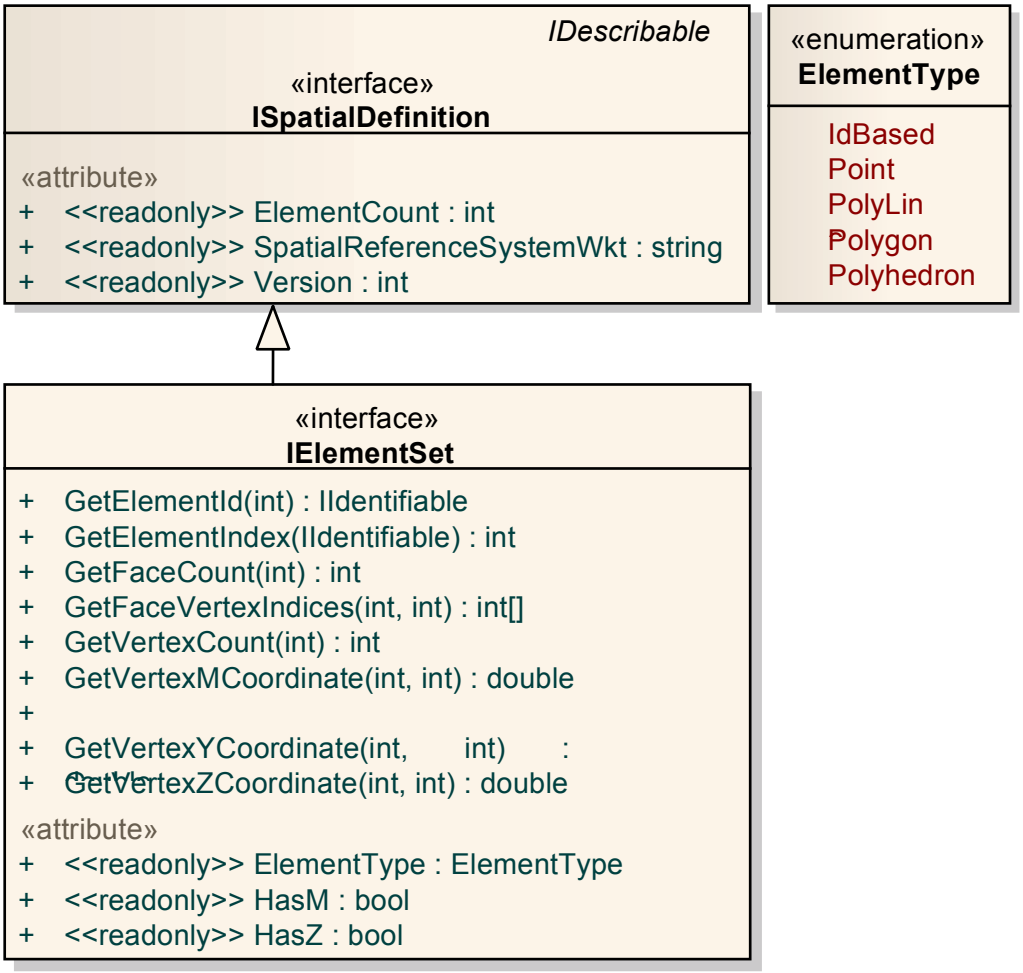


Figure 5 UML Diagram for Spatial Definition

6.4.1 Dependencies, attributes and methods for Spatial Definition interfaces

Table 11 Dependencies for ISpatialDefinition

Connector	Source	Target	Notes
Generalization Source -> Destination	Public IElementSet	Public ISpatialDefinition	
Generalization Source -> Destination	Public ISpatialDefinition	Public IDescribable	

Table 12 Members of the ISpatialDefinition interface

Member	Notes	Parameters
ElementCount int Public	Returns the number of data elements on the spatial axis, e.g. the number of points or the number of polygons.	
SpatialReferenceSystemWkt string Public	Returns the "SpatialReferenceSystemWkt" which specifies the OGC® Well Known Text representation of the spatial reference system to be used in association with the co-ordinates in the "ISpatialDefinition" interface. For the list of WKT strings see: "http://spatialreference.org/".	
Version int Public	Returns the version number identifying the set of spatial data that describes the current position, shape and extent of the elements in the element set. The version must be incremented if any of the spatial data are changed.	

Table 13 Members of the IElementSet interface

Member	Notes	Parameters
ElementType ElementType Public	Returns the "ElementType" attribute of the element set. All elements in the set are of this type.	
GetElementId() IIdentifiable Public	Returns the Id of the "index"th element in the element set. Indexes start from zero. If the element type of the element set is not "IdBased", a null or an empty string may be returned.	**int** [in] **index** The element index for which the element caption is requested. If the element index is outside the range [0, number of elements minus 1], an exception must be thrown.
GetElementIndex() int Public	Returns the index of the element with the identifier "elementId" in the element set, or -1 if the Id was not found. Indexes start from zero. There are no restrictions on how elements are ordered.	**IIdentifiable** [in] **elementId** Identification string for the element for which the element index is requested. If no element in the element set has the specified elementId, -1 must be returned.
GetFaceCount() int Public	Returns the number of faces in a 3D element. For 2D elements this returns 0.	**int** [in] **elementIndex** Index for the element If the element index is outside the range [0, number of elements minus 1], an exception must be thrown.
GetFaceVertexIndices() int Public	Returns the array of the vertex indices for a face. Remark: The vertex indices for a face must be locally numbered for the element (containing numbers in the range [0;"GetVertexCount" (elementIndex)-1]).	**int** [in] **elementIndex** Element index. **int** [in] **faceIndex** Face index.
GetVertexCount() int Public	Returns the number of vertices for the element specified by the "elementIndex". If the "GetVertexCount()" method is invoked for element sets of type "IdBased", an exception must be thrown.	**int** [in] **elementIndex** The element index for the element for which the number of vertices is requested. If the element index is

Member	Notes	Parameters
		outside the range [0, number of elements minus 1], an exception must be thrown.
GetVertexMCoordinate() double Public	Returns the M co-ordinate for the vertex with VertexIndex of the element with "elementIndex".	**int** [in] **elementIndex** Element index. **int** [in] **vertexIndex** Vertex index in the element with index "elementIndex".
GetVertexXCoordinate() double Public	Returns the X co-ordinate for the vertex with vertexIndex of the element with "elementIndex".	**int** [in] **elementIndex** Element index. **int** [in] **vertexIndex** Vertex index in the element with index "elementIndex".
GetVertexYCoordinate() double Public	Returns the Y co-ordinate for the vertex with vertexIndex of the element with "elementIndex".	**int** [in] **elementIndex** Element index. **int** [in] **vertexIndex** Vertex index in the element with index "elementIndex".
GetVertexZCoordinate() double Public	Returns the Z co-ordinate for the vertex with vertexIndex of the element with "elementIndex".	**int** [in] **elementIndex** Element index. **int** [in] **vertexIndex** Vertex index in the element with index "elementIndex".
HasM bool Public	Returns a boolean which is true if the element set contains M co-ordinates.	
HasZ bool Public	Returns a boolean which is true if the element set contains Z co-ordinates.	

Table 14 The ElementType enumeration

ElementType	Convention
IDBased	Id-based (string comparison).
Point	Geo-referenced point in the horizontal plane or in the 3-dimensional space.
PolyLine	Geo-referenced polyline connecting at least two vertices in the horizontal plane or in the 3-dimensional space. The begin- and end-vertex indicate the direction of any fluxes. Open entity with begin- and end-vertex not being identical.
Polygon	Geo-referenced polygons in the horizontal plane or in the 3-dimensional space. Vertices defined anti-clockwise. Closed entity with one face, begin- and end-vertices being identical.
Polyhedron	Geo-referenced polyhedra in 3-dimensional space. Vertices defined anti-clockwise for each face. Closed entity with many faces, begin- and end-vertices being identical

6.4.2 Requirements for Spatial Definition interfaces

Requirement 4.1: Spatial Definition/Spatial Definition
/req/spatial-definition/ISpatialDefinition
An OpenMI component ***shall*** implement the ISpatialDefinition interface based on the definition in Figure 5 and Table 12.

Requirement 4.2: Spatial Definition/Element Set
/req/spatial-definition/IElementSet
An OpenMI component ***shall*** implement the IElementSet interface derived from ISpatialDefinition based on the definition in Figure 5 and Table 13.

Requirement 4.3: Spatial Definition/Element Type
/req/spatial-definition/IElementType
An OpenMI component ***shall*** implement the ElementType enumeration for the known element types as specified in Table 14.

6.5 The Temporal Definition interfaces

Time in the OpenMI is defined by the "ITimeSet" interface - see Figure 6. A time set contains a list of times, where time is specified by the "ITime" interface, containing a Modified Julian Day value and duration. If the duration is zero, the time is a time stamp. If it is greater than zero it is a time span.

Requirements Class 5: Temporal Definition	
/req/temporal-definition	
Target type	OpenMI component
Dependency	
Requirement 5.1	/req/temporal-definition/ITime
Requirement 5.2	/req/temporal-definition/ITimeSet

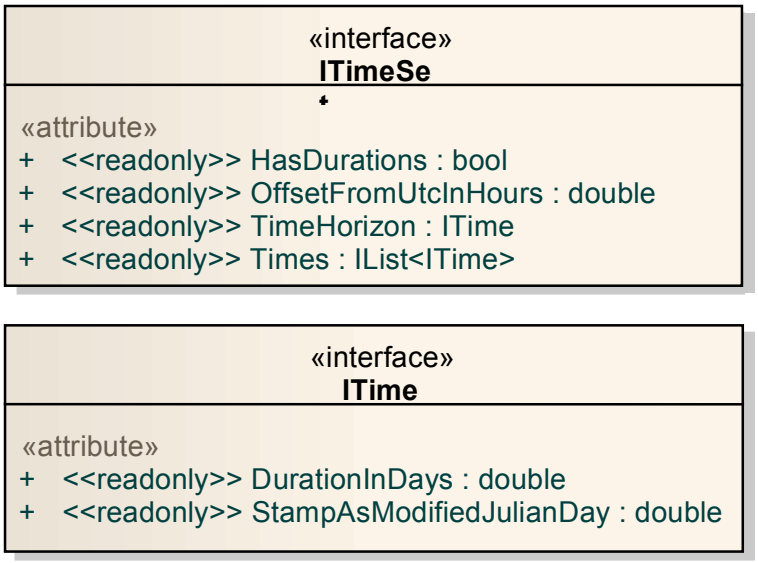


Figure 6 UML Diagram for Temporal Definition

6.5.1 Attributes and methods for Temporal Definition interfaces

Table 15 Members of the ITimeSet interface

Member	Notes	Parameters
HasDurations bool Public	Returns a boolean which is true if the "Times" have durations, i.e. are time spans. In this case, a duration value greater than zero is expected for every "ITime" in the "Times" list.	
OffsetFromUtcInHours double Public	Returns the time zone offset from UTC, expressed as the number of hours. Because some of the world's time zones differ by half an hour from their neighbours, the value is specified as a double.	
TimeHorizon ITime Public	Returns the time horizon for an input item i.e. the period of time over which it may request values. This means that the providers of this input can assume that the input item will never request data for times earlier than the time horizon's begin time, "TimeHorizon.StampAsModifiedJulianDay" . Also, it will never request data for times after the time horizon's end time. "TimeHorizon.StampAsModifiedJulianDay+ TimeHorizon.DurationInDays". For an output item, and thus for an adapted output item, the time horizon indicates in what time span the item can provide values. Specific values: TimeHorizon.StampAsModifiedJulianDay == Double.NegativeInfinity : far back in time TimeHorizon.Duration == Double.PositiveInfinity : far in the future.	
Times IList<ITime> Public	Returns the time stamps or time spans as available in the values of an output item, or as required by an input item. Specific values: If for an output item TimeSet.Times.Count == 0 , the output item is time-dependent, but there are no values available yet. If for an input item TimeSet.Times.Count == 0, the input item is time-dependent item, but currently there are no values	

Member	Notes	Parameters
	required yet.	

The "ITimeSet" interface contains additional information about the time stamps or spans that it contains or will contain. The "OffsetFromUtcInHours" attribute indicates the time zone in terms of an offset from UTC time.

The value of the "HasDurations" attribute specifies whether the time set's times are time stamps (False) or time spans (True).

The "TimeHorizon" attribute provides information on the timeframe during which an exchange item will interact with other exchange items. For an input item, the attribute specifies for what time span the input item can be expected to request values during the computation. This means that the providers of this input can assume that the input item never goes back further in time than the time horizon's begin time, "StampAsModifiedJulianDay". Also, it will never go further ahead than the time horizon's end time, "StampAsModifiedJulianDay + DurationInDays".

For an output item, and also for an adapted output item, the time horizon indicates the time span in which the output will be able to provide values.

To indicate that an input item may ask for values far back in time, or that an output item can provide values as far back in time as requested, the begin time of the time horizon should be set to infinitely back in time, i.e. the "StampAsModifiedJulianDay" should be set to the 'negative infinity value' of a double precision number. Comparably, if an input item may request values far in future, or if an output item can provide values far in the future, the end time of the time horizon should be set to infinitely far ahead in time, i.e. the "DurationInDays" should be set to the 'positive infinity value' of a double precision number.

Table 16 Members of the ITime interface

Member	Notes	Parameters
DurationInDays double Public	Returns the duration in days. Zero if time is a time stamp.	
StampAsModifiedJulianDay double Public	Returns a time stamp as a modified julian day value.	

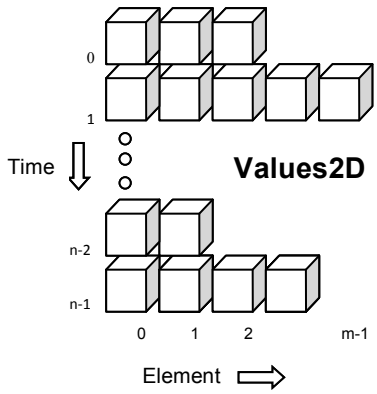
6.5.2 Requirements for Temporal Definition interfaces

Requirement 5.1: Temporal Definition/Time
/req/temporal-definition/ITime
An OpenMI component supporting time handling ***shall*** implement the ITime interface based on the definition in Figure 6 and Table 16.

Requirement 5.2: Temporal Definition/Time Set
/req/temporal-definition/ITimeSet
An OpenMI component supporting time handling ***shall*** implement the ITimeSet interface based on the definition in Figure 6 and Table 15.

6.6 The Value Set interfaces

The volume and structure of the data values exchanged between components can vary from a single value to large n-dimensional arrays. Therefore a generic solution, the "IBaseValueSet" interface, has been developed which component developers can configure to meet their specific requirements. However, in most instances, the full power of the generic solution will not be required. Therefore the time and space extension OpenMI.Standard2.TimeSpace contains a preconfigured version of the "IBaseValueSet" interface, the "ITimeSpaceValueSet" interface. This will be described first as it provides a useful introduction to the more abstract generic solution.



The "ITimeSpaceValueSet" interface sets and returns values in the form of an ordered 'list of lists', called "Values2D". This can be visualised as shown in the adjoining diagram. The first dimension (the rows) represents the times for which values are to be set or were requested. There is one row for each time. The cells along each row (which can be thought of as the second dimension) each contain the value relating to precisely one element in the element set (which was specified when setting or requesting for the values). In other words, the i^{th} value in that dimension of the value set corresponds to the i^{th} element in the specified element set.

Note: the list of lists structure allows the rows to be of different lengths, though in many cases they will all be the same length. The 'missing value' value, e.g. -999, must be used to ensure that values are located the correct column for the element to which they relate.

The following pseudo code illustrates how values can be located in the "Values2D" list of lists. Note that all indices are zero-based.

The number of distinct times for which values are returned (= the number of rows):

```
numTimes = valueSet.Values2D.Count
```

The number of elements for which values are returned for the first time (including missing value' represented by the 'missing value' value, e.g. -999):

```
numElementsForFirstTime = valueSet.Values2D[0].Count
```

The value returned for the second time and fifth element (note the indices start at zero):

```
valueForSecondTimeAndFifthElement = valueSet.Values2D[1][4]
```

The values for all elements for the sixth time:

```
valuesForAllElementsOnSixthTime =  valueSet.Values2D[5]
```

The values for the fourth element for all times:

```
ValuesForFourthElementForAllTimes = GetTimeSeriesValuesForElement(3);
```

Methods and attributes to simplify access to "Values2D" will be found in Table 18. For example, Three pairs of Set/Get-Value methods are provided for requesting or setting exchange item values. These allow the user to set or get the values for:

- A specific time and/or element (Set/Get-Value)
- All times for a given element (Set/Get-TimeSeriesValuesForElement)
- All elements for a given time (Set/Get-TimeSeriesValuesForTime)

Not all models or components require the representation of time and/or space. Even if they do, they may still need to exchange data that do not have spatial or temporal dimensions. An example might be the need to set or request the values of model parameters during calibration. There may also be data that do have spatial and temporal dimensions but which do not fit the model above, for example, a time-series of 3-dimensional gridded values or other data types altogether such as complex numbers.

The "IBaseValueSet" interface provides the developer with a blob through which values can be set or returned by the "GetValue()" and "SetValue()" methods – see Table 17.

The developers can configure this blob as a list or list of lists, nested as many times as are necessary to handle the structure of the values to be exchanged. If more convenient, the blob may be viewed as a multi-dimensional array, whose dimensionality can be discovered through the "NumberOfIndices" attribute. The blob is "Idescribable" and so the developer can use the caption and description attributes to document the structure of the blob. In particular, the description should explain the meaning of each level in the lists or each dimension of the array. For example, they could be used to represent x, y, z and t. However, it is important to stress that the axes are not confined to representing time or space. They may represent any quantity. The following pseudo code shows how a 3 level list of lists of lists, i.e. a 3-dimensional array can be accessed using the methods and attributes in Table 17.

The number of dimensions to the array:

```
int nDimensions = NumberOfIndices;
```

The number of slices in the array:

```
int nSlices = GetIndexCount(new int{});
```

The number of rows in slice 3 of the array:

```
int nRowsInSlice3 = GetIndexCount(new int{2});
```

The number of cells in row2 of slice 3 of the array:

```
int nCellsInSecondRowOfSlice3 = GetIndexCount(new int{2,1});
```

The value in the 5th cell in row2 of slice 3 of the array:

```
valueInFifthCellOfSecondRowOfSlice3 = valueSet.GetValue(new int{ 2, 1, 4})
```

Moving on from the structure in which exchanged values, the specification will now explain the interpretation of those values, where the exchange item value indicates the movement of something, for example the flow of water, from the "source/providing" component to the "target/requesting/accepting" component. To prevent the misunderstanding of positive and negative values, the following conventions, which are illustrated in Figure 8, have been adopted:

- Values are positive if the matter leaves the source component and enters the target component.

- The 'right-hand rule' applies for fluxes through a plane or polygon[8].
- The direction of fluxes along a polyline is defined as positive from the begin node to the end node.
- The 'right-hand rule' applies for fluxes perpendicular to a polyline[9].

Software developers who do not comply with these conventions should make software users aware that have adopted a different rule..

Requirements Class 6: Value Set	
/req/value-set	
Target type	OpenMI component
Dependency	
Requirement 6.1	/req/value-set/IBaseValueSet
Requirement 6.2	/req/value-set/ITimeSpaceValueSet

[8] Curl your right hand in the vertex order of the plane or polygon. The thumb points in the positive direction

[9] Put your hand along the line in the positive direction, turn your wrist clockwise. The thumb will point in the positive direction perpendicular to the (poly)line.

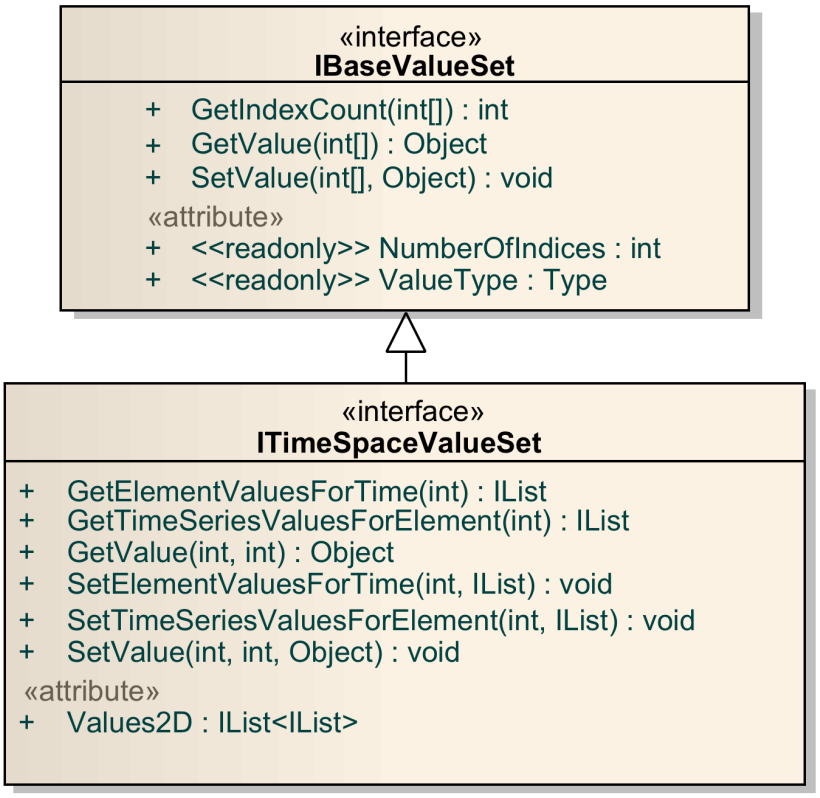


Figure 7 UML Diagram for Value Set

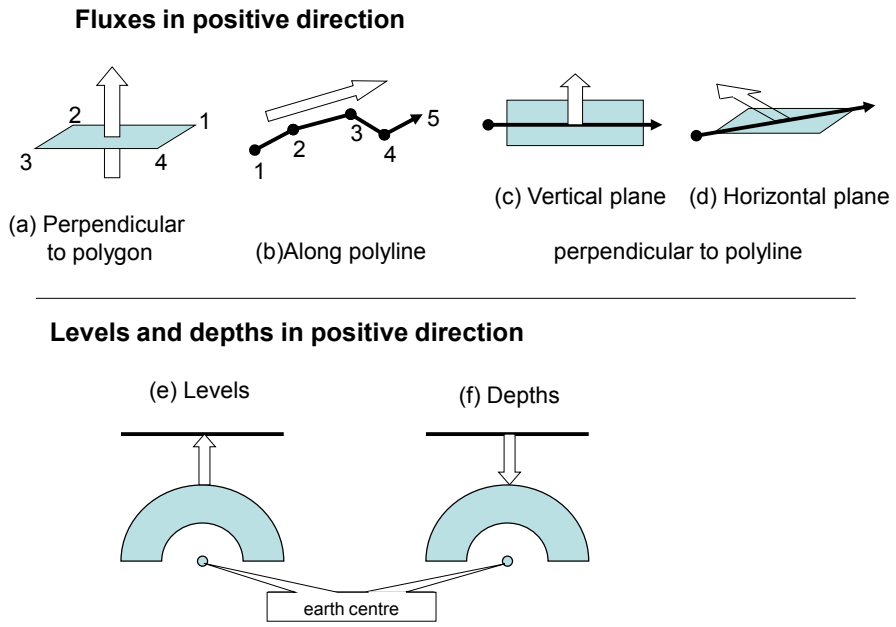


Figure 8 Illustration of directions to interpret positive values of fluxes, levels and depths

6.6.1 Attributes and methods for Value Set interfaces

Table 17 Members of the IBaseValueSet interface

Member	Notes	Parameters
GetIndexCount() int Public	Returns the length (max index count) of the dimension specified by the given indices. To obtain the size of the first dimension, use a zero-length integer array as input argument. Length of indices must be a least one smaller than the "NumberOfIndices".	**int[]** [in] **indices** Indices specifying the dimension whose length is to be obtained
GetValue() Object Public	Returns the value object specified by the given array of indices. The length of the array of indices must be the equal to the number of dimensions, so that the index for each dimension is specified. Otherwise an IllegalArgumentException must be thrown.	**int[]** [in] **indices** Index value for each dimension
NumberOfIndices int Public	Returns the number of possible indices (dimensions) for the value set.	
SetValue() void Public	Sets the value object specified by the given array of indices. The length of the array of indices must be the equal to the number of dimensions, so that the index for each dimension is specified. Otherwise an IllegalArgumentException must be thrown.	**int[]** [in] **indices** Value for each dimension **Object** [in] **value** The value object for the given indices
ValueType Type Public	Returns the object type of the values that will be available in the value set that is returned by the Values attribute and the "GetValues()" method.	

Table 18 Members of the ITimeSpaceValueSet interface

Member	Notes	Parameters
GetElementValuesForTime() IList Public	Returns values from the "Values2D" attribute, for all elements, for the specified "timeIndex". If the data are time independent, "timeIndex" must be specified as 0.	**int** [in] **timeIndex**
GetTimeSeriesValuesForElement() IList Public	Returns values from the "Values2D" attribute, for all times, for the specified "elementIndex". If the data are not related to a location, "elementIndex" must be specified as 0.	**int** [in] **elementIndex**
GetValue() Object Public	Returns the value for the specified "timeIndex" and "elementIndex" from "Values2D". If the data are time independent, "timeIndex" must be specified as 0. If the data are not related to a location, "elementIndex" must be specified as 0.	**int** [in] **timeIndex** **int** [in] **elementIndex**
SetElementValuesForTime() void Public	Sets values for the "Values2D" attribute, for all elements, for the specified "timeIndex". If the data are time independent, "timeIndex" must be specified as 0.	**int** [in] **timeIndex** **IList** [in] **values**
SetTimeSeriesValuesForElement() void Public	Sets values in the "Values2D" attribute, for all times, for the specified "elementIndex". If the data are not related to a location, "elementIndex" must be specified as 0.	**int** [in] **elementIndex** **IList** [in] **values**
SetValue() void Public	Sets the value in the "Values2D" attribute, for the specified "timeIndex" and "elementIndex". If the data are time independent, "timeIndex" must be specified as 0. If the data are not related to a location, "elementIndex" must be specified as 0.	**int** [in] **timeIndex** **int** [in] **elementIndex** **Object** [in] **value**
Values2D IList<IList> Public	Returns and sets a two-dimensional list of values. The first IList represents time, and the contained IList represents the	

Member	Notes	Parameters
	elements in the "IElementSet".	

6.6.2 Requirements for Value Set interfaces

Requirement 6.1: Base Value Set
/req/value-set/IBaseValueSet
An OpenMI component ***shall*** implement the IBaseValueSet interface based on the definition in Figure 7 and Table 17.

Requirement 6.2: Time Space Value Set
/req/value-set/ITimeSpaceValueSet
An OpenMI component exchanging time-based data ***shall*** implement the ITimeSpaceValueSet interface based on the definition in Figure 7 and Table 18.

6.7 The Argument interface

Both the adapted output and the linkable component contain arguments that are used to provide information to let the adapted output do its work. This is achieved by means of the "IArgument" interface – see Figure 9.

Requirements Class 7: Argument	
/req/argument	
Target type	OpenMI component
Dependency	/req/describable-identifiable/IIdentifiable
Requirement 7.1	/req/argument/IArgument

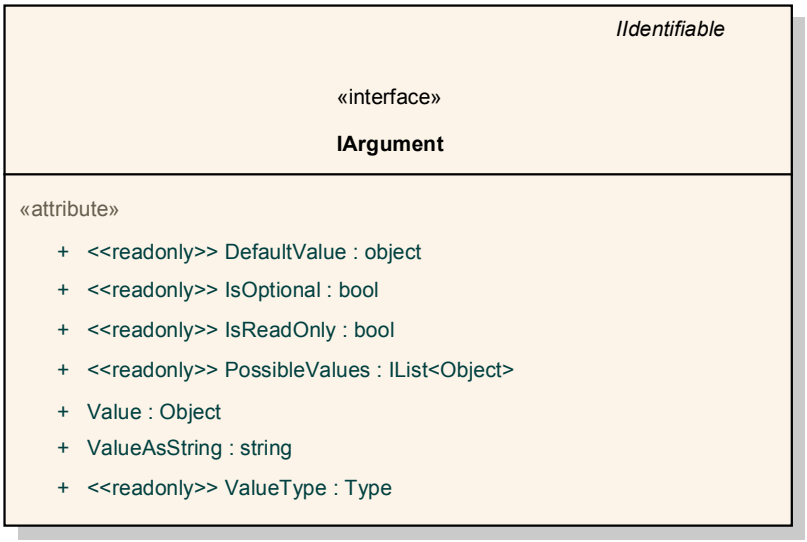


Figure 9 UML Diagram for Argument

6.7.1 Attributes and methods for the Argument interface

Table 19 Members of the IArgument interface

Member	Notes
DefaultValue Object Public	Returns the default value of the argument.
IsOptional bool Public	Returns whether or not the argument is optional. If the "Values" attribute returns null and IsOptional == false, a value has to be set before the argument can be used.
IsReadOnly bool Public	Returns whether or not the "Values" attribute may be edited. This is used to let an "IBaseLinkableComponent" or an "IBaseAdaptedOutput" present the actual value of an argument that cannot be changed by the user, but is needed to determine the values of other arguments or is informative in any other way.
PossibleValues IList<Object> Public	Returns a list of possible allowed values for this argument. If for integral types or component specific types all possible values are allowed, null is returned. A list with length 0 indicates that there is indeed a limitation on the possible values, but that currently no values are possible. Effectively this means that the values will not and cannot be set.
Value Object Public	Returns or sets the current value of the argument. If no value has been set yet, a default value is returned. If null is returned, this means that the default value is null.
ValueAsString string Public	Returns or sets the argument's value, represented as a string. If "ValueType" indicates that the argument's value is not of the type string, the "ValueAsString" attribute offers the possibility to treat it as a string, e.g. to let the GUI persist the value in the composition file.
ValueType Type Public	Returns the type of the value of the argument, e.g. an integral type such as a string, integer or double, or a non-integral type, such as a time series object.

6.7.2 Requirements for the Argument interface

Requirement 7.1: Argument/Argument
/req/argument/IArgument
An OpenMI component exchanging time-based data ***shall*** implement the IArgument interface based on the definition in Figure 9 and Table 19.

6.8 Linkable Component Status

This clause should be read in conjunction with the information in Clause 6.9 on the Exchange Item as some component states, e.g. 'updating' may lead to Exchange Item Change events.

The linkable component status has two purposes: an informative purpose (what is the component currently doing, i.e. what state is it in) and a control flow decision purpose. The OpenMI explicitly specifies the possible states in which a linkable component can be found. When a component moves from one status to another, the component must raise an event through the "IBaseLinkableComponent" interface. The "LinkableComponentStatusChangeEventArgs" class contains the information that will be passed.

Example states for a component are: created, initializing, updating, finishing and finished. The complete list of these terms forms the "LinkableComponentStatus" enumeration; explanations of the terms are given in Table 2. Figure 10 shows the sequence in which they can occur. Readers will probably find it helpful to view the diagram first.

Requirements Class 8: Linkable Component Status	
/req/linkable-component-status	
Target type	OpenMI component
Dependency	
Requirement 8.1	/req/linkable-component-status/LinkableComponentStatus
Requirement 8.2	/req/linkable-component-status/LinkableComponentBehaviour
Requirement 8.3	/req/linkable-component-status/EventStatusChanged

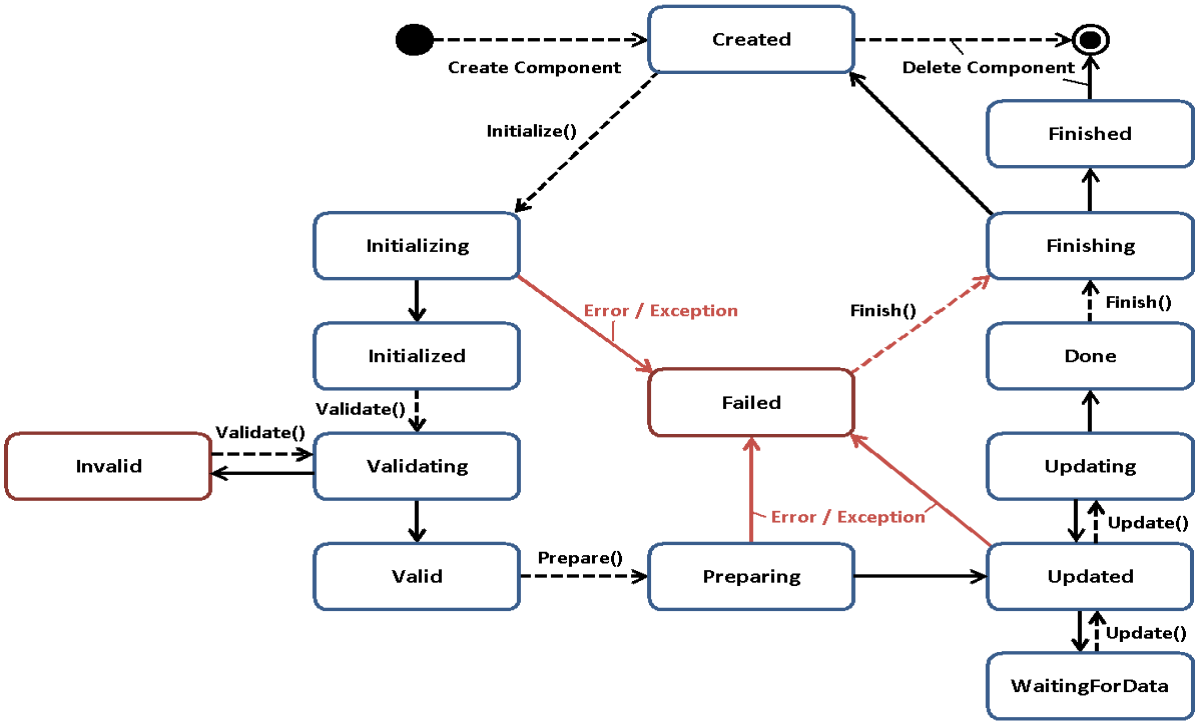


Figure 10 Component Status Change Diagram

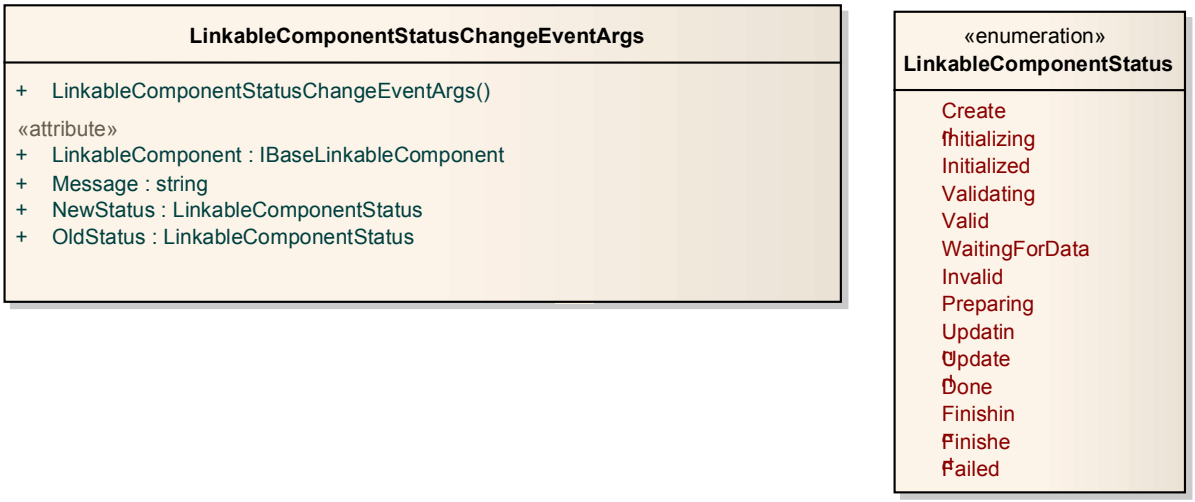


Figure 11 UML Diagram for Linkable Component Status

Table 20 The LinkableComponentStatus enumeration

Attribute	Notes
Created Public	The linkable component instance has just been created. This status must and will be followed by "Initializing".
Initializing Public	The linkable component is initializing itself. This status will end in a status change to "Initialized" or "Failed".
Initialized Public	The linkable component has successfully initialized itself. The connections between its inputs/outputs and those of other components can now be established.
Validating Public	After links between the component's inputs/outputs and those of other components have been established, the component is validating whether its required input will be available when it updates itself, and whether indeed it will be able to provide the required output during this update. This Validating status will end in a status change to "Valid" or "Invalid".
Valid Public	The component is in a valid state. When updating itself the component's required input will be available, and the component will now be able to provide the required output.
WaitingForData Public	The component wants to update itself, but is not yet able to perform the actual computation, because it is still waiting for input data from other components.
Invalid Public	The component is in an invalid state. When updating itself not all required input will be available, and/or it will not be able to provide the required output. After the user has modified the connections between the component's inputs/outputs and those of other components, the "Validating" state can be entered again.
Preparing Public	The component is preparing itself for the first "GetValues()" call. This Preparing state will end in a status change to "Updated" or "Failed".
Updating Public	The component is updating itself. It has received all required input data from other components, and is now performing the actual computation. This Updating state will end in a status change to "Updated", "Done" or "Failed".
Updated Public	The component has successfully updated itself.
Done Public	The last update process that the component performed was the final one. A next call to the Update method will leave the component's internal state unchanged.
Finishing Public	The "IBaseLinkableComponent" was requested to perform the actions to be performed before it will either be disposed or re-initialized again. Typical actions would be writing the final result files, close all open files, free memory, etc. When all required actions have been performed, the status switches to "Created" when re-initialization is possible. The status switches to "Finished" when the component is to be disposed.

Attribute	Notes
Finished Public	The "IBaseLinkableComponent" has successfully performed its finalization actions. Re-initialization of the component instance is not possible and should not be attempted. Instead the instance should be disposed, e.g. through the garbage collection mechanism.
Failed Public	The linkable component has failed to initialize itself, to prepare itself for computation or to complete its update process.

6.8.1 Attributes and methods for the Linkable Component Status Change Event Args class

Table 21 Members of the LinkableComponentStatusChangeEventArgs class

Member	Notes	Parameters
LinkableComponent IBaseLinkableComponent Public	Returns and sets the "LinkableComponent" that raised the status change event.	
LinkableComponentStatusChangeEventArgs() Public	Constructor.	
Message string Public	Returns and sets the "Message" attribute providing additional information on the status change. If there is no message, an empty string is returned.	
NewStatus LinkableComponentStatus Public	Returns and sets the linkable component's "NewStatus" attribute after the status changes.	
OldStatus LinkableComponentStatus Public	Returns and sets the linkable component's "OldStatus" attribute thus recording its status before the status change.	

6.8.2 Requirements for Linkable Component Status

Requirement 8.1: Linkable Component Status/Linkable Component Status
/req/linkable-component-status/LinkableComponentStatus
An OpenMI component ***shall*** provide its status using one of the status conditions from the LinkableComponentStatus enumeration and defined in Figure 11 and Table 20.

Requirement 8.2: Linkable Component Behaviour
/req/linkable-component-status/LinkableComponentBehaviour
An OpenMI component ***shall*** change its status according to the rules given in Figure 10 and Table 19.

Requirement 8.3: Linkable Component Status/Event Status Changed
/req/linkable-component-status/EventStatusChanged
If the implementation programming language provides an event-handling mechanism, then, when a component moves from one state to another, the component ***shall*** raise a LinkableComponentStatusChange event through the "IBaseLinkableComponent" interface. This event shall pass the information contained in the "LinkableComponentStatusChangeEventArgs" class, as described in Figure 11 and Table 21.**Error! Reference source not found.**

6.9 Input and output interfaces

Correctly interpreting an exchange item value requires supporting information in order to understand what it represents, where it applies, when it applies and how it may be processed. This information is obtained through the "IBaseExchangeItem" interface and the various interfaces that are derived from it. Figure 12 shows schematically how these interfaces relate to the passage of data between two linkable components and Figure 13 presents a UML diagram of the interfaces.

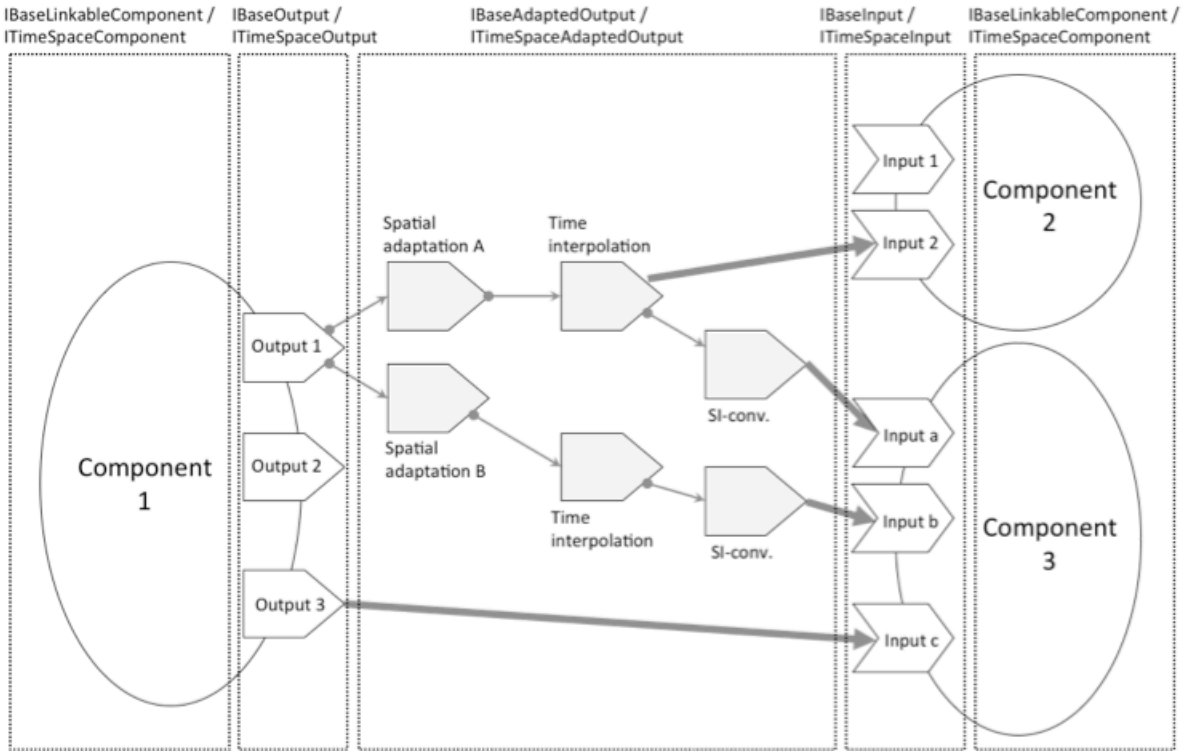


Figure 12 OpenMI Interfaces and the passage of data between two components

For a linkable component, an exchange item is either an input item, also known as an "IBaseInput", or an output item, also known as an "IBaseOutput". Time- and space-dependent linkable components will provide and recognize the "ITimeSpaceExchangeItem" which is derived from the "IBaseExchangeItem". A time space exchange time is either an "ITimeSpaceInput" or an "ITimeSpaceOutput".

An input is connected to an output by calling the output item's "AddConsumer()" method. This method will take the internal actions needed to ensure that values can be provided once the computation starts, and will add the input to the "Consumers" list. At the same time, this method sets the output item as the provider of the input item.

If a connection is no longer needed, the input is removed as a consumer by calling the output item's "RemoveConsumer()" method. This method may perform internal clean-up actions and will remove the input from the "Consumers" list. At the same time, this method sets the "Provider" of the input item to null.

Once computation starts, any of a component's output exchange items may be requested to supply values by another component. The other component does so by invoking the first component's output exchange item's "GetValues()" method. The query specification argument will nearly always be a consumer of the output item. In some situations, however, it may be another instance of an exchange item specifying what is required, as might arise during testing or visualization.

When any aspect of an exchange item changes such as, for example, its value definition, time set or its values, the component must raise an event through the "IBaseExchangeItem" interface. The ExchangeItemChangeEventArgs class contains the information that will be passed.

Requirements Class 9: Exchange Item	
/req/exchange-item	
Target type	OpenMI component
Dependency	
Requirement 9.1	/req/exchange-item/IBaseExchangeItem
Requirement 9.2	/req/exchange-item/IBaseInput
Requirement 9.3	/req/exchange-item/IBaseOutput
Requirement 9.4	/req/exchange-item/ITimeSpaceExchangeItem
Requirement 9.5	/req/exchange-item/ITimeSpaceOutput
Requirement 9.6	/req/exchange-item/ITimeSpaceInput
Requirement 9.7	/req/exchange-item/EventExchangeItemChanged

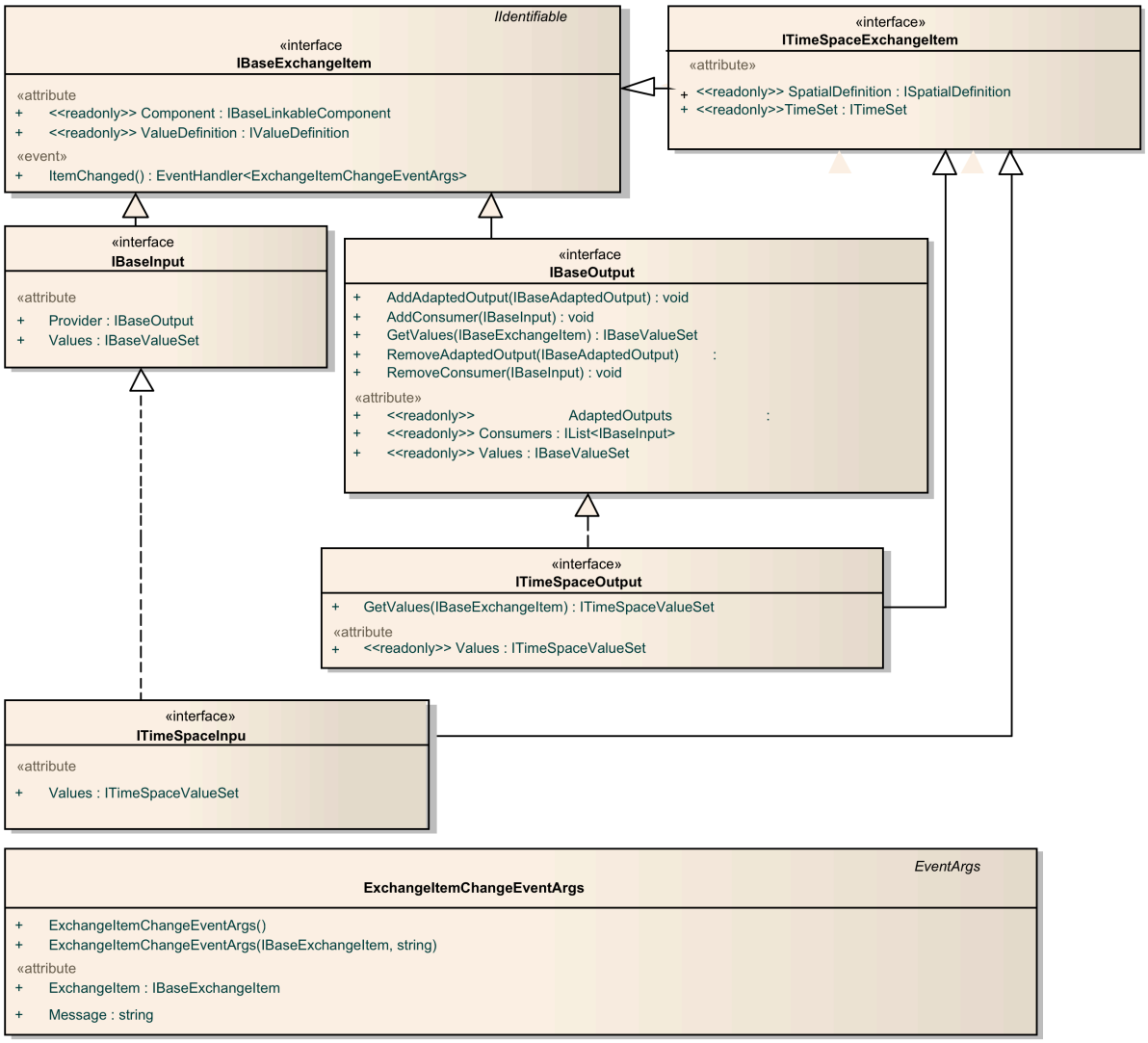


Figure 13 UML Diagram for Exchange Item

6.9.1 Attributes and methods for Exchange Item interfaces

6.9.1.1 The IBaseExchangeItem interface

Table 22 Members of the IBaseExchangeItem interface

Member	Notes	Parameters
Component IBaseLinkableComponent Public	Returns the owner of the exchange item. For an output exchange item this is the component responsible for providing the content of the output item. It is possible for an exchange item to have no owner; in this case the method will return null.	
ItemChanged() EventHandler<ExchangeItemChangeEventArgs> Public	The ItemChanged event is raised when the content of an exchange item has changed. This might be because its "ValueDefinition" has changed, its "TimeSet" has changed, its "ElementSet" has changed, its "Values" have changed, or any permutation of these properties.	
ValueDefinition IValueDefinition Public	Returns the definition of the values in the exchange item. Remark: The "IValueDefinition" should never be returned directly; all implementing classes should return either an "IQuality", an "IQuantity", or a custom-derived value definition interface.	

6.9.1.2 The ExchangeItemChangeEventArgs class

If the implementation programming language provides an event-handling mechanism, then when, during computation, any aspect of an exchange item alters, the component must raise an ExchangeItemChanged event through the "IBaseExchangeItem" interface - see "ItemChanged" in Table 22.

The ExchangeItemChangeEventArgs class contains the information that will be passed when the "IBaseExchangeItem" raises the ExchangeItemChanged event.

Raising an ExchangeItemChanged event depends upon whether the language used implements the event concept, so it should not be used as a mechanism upon which to build critical functionality.

Table 23 Members of the ExchangeItemChangeEventArgs class

Member	Notes	Parameters
ExchangeItem IBaseExchangeItem Public	Returns and sets the exchange item whose status has been changed.	
ExchangeItemChangeEventArgs() Public	Default constructor. Creates a new instance with an empty message and null as "ExchangeItem". Properties need to be set before actually using the instance.	
ExchangeItemChangeEventArgs() Public	Constructor that also initializes the "ExchangeItem" and the "Message" attribute.	**IBaseExchangeItem** [in] **exchangeItem** **string** [in] **message**
Message string Public	Returns and sets a message that describes the way in which the status of the exchange item has been changed.	

6.9.1.3 The ITimeSpaceExchangeItem interface

A time/space dependent item that can be exchanged, either as an input or as an output.

Table 24 Members of the ITimeSpaceExchangeItem interface

Member	Notes	Parameters
SpatialDefinition ISpatialDefinition Public	Returns spatial information (usually in the form of an element set) about the values that are available in an output exchange item or required by an input exchange item.	
TimeSet ITimeSet Public	Returns temporal information about the values that are available in an output exchange item or required by an input exchange item.	

6.9.1.4 The IBaseInput interface

An input item that can accept values for an "IBaseLinkableComponent"

Table 25 Members of the IBaseInput interface

Member	Notes	Parameters
Provider IBaseOutput Public	Returns and sets the provider of this "IBaseInput".	

Member	Notes	Parameters
Values IBaseValueSet Public	Returns and sets the exchange item's values.	

6.9.1.5 The ITimeSpaceInput interface

An input item that can accept values for an "ITimeSpaceComponent". The item is a combination of an "IValueDefinition", an "IElementSet" and an "ITimeSet". This combination specifies which type of data is required, where and when, as input for an "ITimeSpaceComponent".

Table 26 Members of the ITimeSpaceInput interface

Member	Notes	Parameters
Values ITimeSpaceValueSet Public	Returns and sets the exchange item's values, as a specialized "ITimeSpaceValueSet"	

6.9.1.6 The IBaseOutput interface

An output exchange item that can deliver values from an "IBaseLinkableComponent".

If an output does not provide the data in the way a consumer would like to receive it, the output can be adapted by an "IBaseAdaptedOutput". This can transform the data according to the consumer's wishes (e.g. by performing interpolation in time, spatial aggregation, unit conversion, etc.)

Table 27 Members of the IBaseOutput interface

Member	Notes	Parameters
AdaptedOutputs IList<IBaseAdaptedOutput> Public	Returns the list of adapted outputs that have the current output item as "Adaptee". As soon as the output item's values have been updated, for each adapted output its "IBaseAdaptedOutput.Refresh()" method must be called. The list is read-only. Add and remove from the list by using the "AddAdaptedOutput()" and the "RemoveAdaptedOutput()" methods.	
AddAdaptedOutput() void Public	Adds an "IBaseAdaptedOutput" to the current output item. Every adapted output that uses data from this output item first needs to add itself as an adaptee first. If an adapted output is added that cannot be handled or that is incompatible with the already added adapted outputs, an exception will be thrown.	**IBaseAdaptedOutput** [in] adaptedOutput "adaptedOutput" consumer that has to be added.
AddConsumer() void Public	Adds a consumer to the current output item. Every input item that wants to call the "GetValues()" method, needs to add itself as a consumer first. If a consumer is added that cannot be handled, or that is incompatible with the already added consumers, an exception will be thrown. The "AddConsumer()" method must automatically set the current output item as the provider of the added consumer – see "IBaseInput.Provider".	**IBaseInput** [in] consumer
Consumers IList<IBaseInput> Public	Returns the list of input items that will consume the values by calling the "GetValues()" method. Every input item that will call this method needs to call the "AddConsumer()" method first. If the input item is no longer interested in calling the "GetValues()" method, it should remove itself by calling the "RemoveConsumer()" method. The list is read-only. Add and remove from the list by using the "AddConsumer()" and "RemoveConsumer()" methods. Remark: Please be aware that the "raw"	

Member	Notes	Parameters
	values in the output item, provided by the read-only "Values" attribute, may be called anyway, even if there are no values available.	
GetValues() IBaseValueSet Public	Provides the values matching the value definition specified by the "querySpecifier". Extensions can overwrite this base version to include more details in the query, e.g. time and space. Remark: Usually the querySpecifier will be of the type "IBaseInput" or "ITimeSpaceInput", being an input item that has first added itself as consumer. However, any "IBaseExchangeItem" or derived exchange item suffices to specify what is required.	**IBaseExchangeItem** [in] querySpecifier
RemoveAdaptedOutput() void Public	Removes an "IBaseAdaptedOutput". If an adapted output is no longer interested in this output item data, it should remove itself by calling RemoveAdaptedOutput.	**IBaseAdaptedOutput** [in] adaptedOutput Adaptee that has to be removed.
RemoveConsumer() void Public	Removes a consumer. If an input item is no longer interested in calling the "GetValues()" method, it should remove itself by calling RemoveConsumer.	**IBaseInput** [in] consumer Consumer that has to be removed.
Values IBaseValueSet Public	Returns the exchange item's values.	

6.9.1.7 The ITimeSpaceOutput interface

An output exchange item that can deliver values from an ITimeSpaceComponent. The output is a combination of an "IValueDefinition" interface, an "IElementSet" interface and an "ITimeSet" interface. This combination specifies which types of data can be provided, where and when by the ITimeSpaceComponent.

If an output does not provide the data in the way a consumer would like to have it, the output can be adapted by an "ITimeSpaceAdaptedOutput", which can transform the data according to the consumer's wishes, e.g. by performing interpolation in time, spatial aggregation or unit transformations.

Table 28 Members of the ITimeSpaceOutput interface

Member	Notes	Parameters
GetValues() ITimeSpaceValueSet Public	This "GetValues()" method returns an "ITimeSpaceValueSet" and is an overridden version of the "IBaseOutput.GetValues()" method, which returns an "IBaseValueSet".	**IBaseExchangeItem** [in] querySpecifier
Values ITimeSpaceValueSet Public	Returns the exchange item's values, as a specialized "ITimeSpaceValueSet".	

6.9.2 Requirements for Exchange Item interfaces

Requirement 9.1: Exchange Item/Base Exchange Item
/req/exchange-item/IBaseExchangeItem
An OpenMI component ***shall*** implement the IBaseExchangeItem interface based on the definition in Figure 13 and Table 22.

Requirement 9.2: Exchange Item/Base Input
/req/exchange-item/IBaseInput
If an OpenMI component needs to accept input it ***shall*** implement the IBaseInput interface based on the definition in Figure 13 **Error! Reference source not found.**and Table 25.

Requirement 9.3: Exchange Item/Base Output
/req/exchange-item/IBaseOutput
If an OpenMI component needs to provide output it ***shall*** implement the IBaseOutput interface based on the definition in Figure 13 and Table 27.

Requirement 9.4: Exchange Item/Time Space Exchange Item
/req/exchange-item/ITimeSpaceExchangeItem
A time-dependent OpenMI component ***shall*** implement the ITimeSpaceExchangeItem interface based on the definition in Figure 13 and Table 24.

Requirement 9.5: Exchange Item/Time Space Output
/req/exchange-item/ITimeSpaceOutput
If an OpenMI component provides time-dependent output it ***shall*** implement the ITimeSpaceOutput interface based on the definition in Figure 13 and Table 28.

Requirement 9.6: Exchange Item/Time Space Input
/req/exchange-item/**ITimeSpaceInput**
If an OpenMI component accepts time-dependent inputs it ***shall*** implement the ITimeSpaceInput interface based on the definition in Figure 13 and Table 26.

Requirement 9.7: Event Exchange Item Changed
/req/exchange-item/EventExchangeItemChanged
If the implementation programming language provides an event-handling mechanism, then when, during computation, any aspect of an exchange item alters, the component ***shall*** raise an ExchangeItemChanged event through the IBaseExchangeItem interface which passes the information contained in the ExchangeItemChangeEventArgs class based on the definition in Figure 13 and Table 23.

6.10 The Adapted Output interfaces

Many situations occur where the raw data available at the source component does not match the request from the target component. For instance, the units of a requested quantity might differ from the units in which the source component provides values for that quantity or the discrete values of a quality may have to be translated into numerical quantity values. For time- and/or space-dependent components, the locations and times for which output values are produced may not match those requested by the input item. In these cases additional data operations may be required including spatial and temporal aggregation, interpolation, unit conversion and many others.

In the clause that follows, the term:

> "To adapt" covers the operations of: unit conversion, spatial and temporal aggregation and disaggregation, interpolation, etc.
>
> "Adaptee" is used for the output exchange item whose values are to be adapted.

For situations where values need adaptation, the "IBaseAdaptedOutput" interface and the "ITimeSpaceAdaptedOutput" (in OpenMI.Standard2.TimeSpace) have been defined – see Figure 14.

To create adapted outputs, the "IAdaptedOutputFactory" has been defined – see Figure 15.

Requirements Class 10: Adapted Output	
/req/adapted-output	
Target type	OpenMI component
Dependency	
Requirement 10.1	/req/adapted-output/IBaseAdaptedOutput
Requirement 10.2	/req/adapted-output/ITimeSpaceAdaptedOutput
Requirement 10.3	/req/adapted-output/IAdaptedOutputFactory

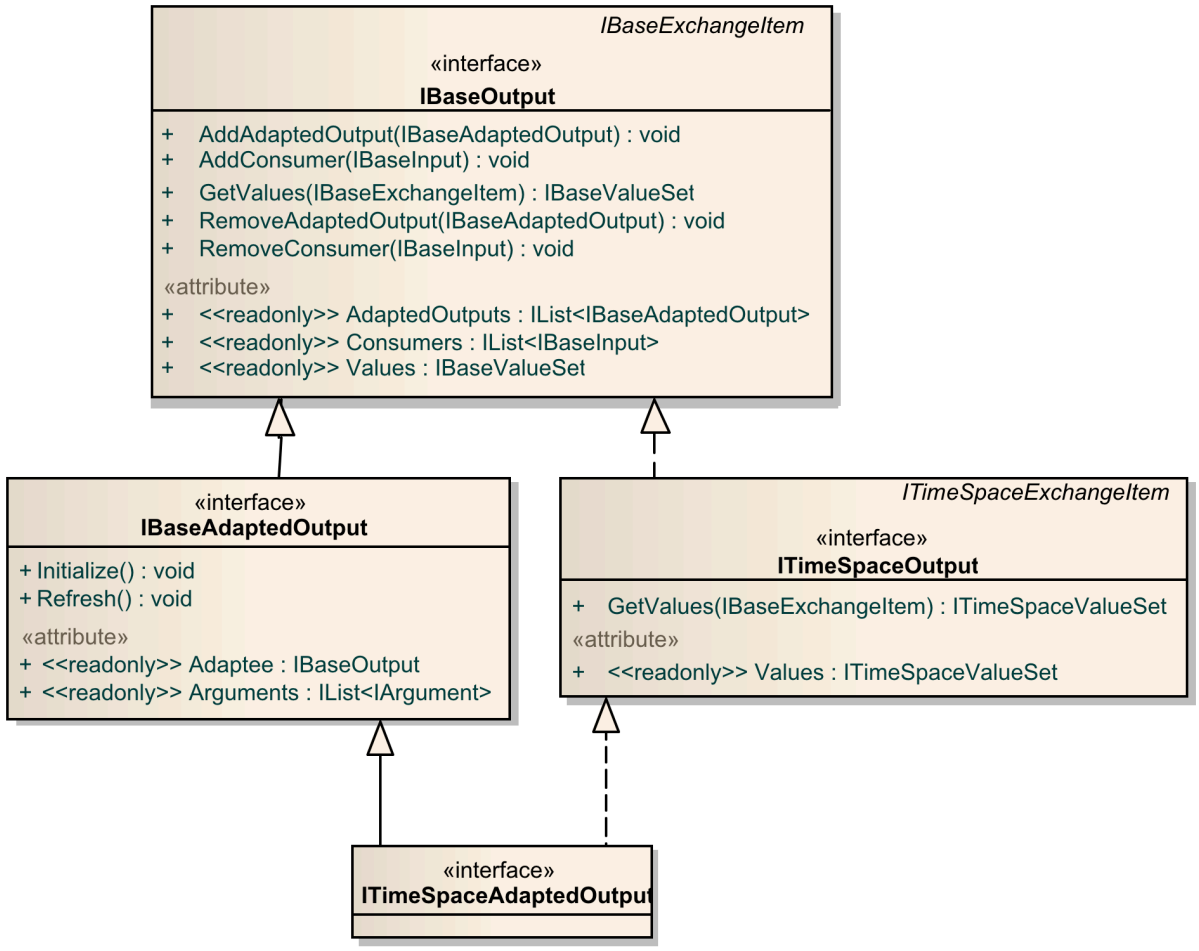


Figure 14 UML Diagram for Adapted Output

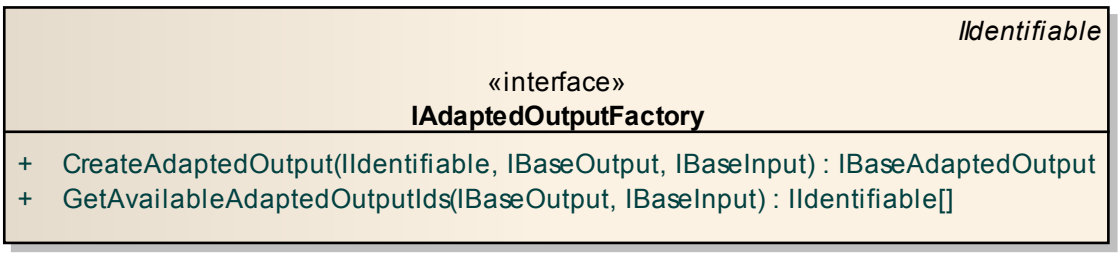


Figure 15 UML Diagram for Adapted Output Factory

Each adapted output may have a number of arguments to manipulate the behaviour of the adapted output. Each argument is specified by means of the "IArgument" interface, a key-value pair – see below. During configuration time the arguments are specified. Before the actual computation starts, during the prepare phase, a linkable component calls the "Initialize()" method of all its adapted outputs. In case of stacked adapted outputs, the adaptee must be initialized first. The "Adaptee" is the output item from which the adapted output takes the source values that it needs to be able to perform its action.

If the values of the adaptee have changed or may have been changed, the adapted output needs to take action. To enable this, "IBaseAdaptedOutput" contains a "Refresh()" method that must be called each time the adaptee has been changed. The linkable component that owns an output item has the responsibility to call the "Refresh()" method of all the output item's "AdaptedOutputs". "IBaseAdaptedOutput" instances are created by means of an "IAdaptedOutputFactory".

6.10.1 Attributes and methods for Adapted Output interfaces

6.10.1.1 The IBaseAdaptedOutput interface

An "IBaseAdaptedOutput" adds one or more data operations on top of an output item. It is in itself an "IBaseOutput". The "IBaseAdaptedOutput" extends an output item with functionality such as spatial interpolation, temporal interpolation, unit conversion etc. "IBaseAdaptedOutput" instances are created by means of an "IAdaptedOutputFactory".

The "IBaseAdaptedOutput" is based on the adaptor design pattern. It adapts an "IBaseOutput" or another "IBaseAdaptedOutput" to make it suitable for a new use or purpose. The object being adapted is typically called the "adaptee".

Table 29 Members of the IBaseAdaptedOutput interface

Member	Notes
Adaptee IBaseOutput Public	Returns the output item that this "adaptedOutput" extracts content from. In the adaptor design pattern, the adaptee is the item being adapted.
Arguments IList<IArgument> Public	Returns the list of arguments needed to let the adapted output do its work. An unmodifiable list of the (modifiable) arguments should be returned that can be used to obtain information on the arguments and to modify argument values. Validation of changes is performed when they occur (e.g. by notifying the user).
Initialize() void Public	Causes the adapted output to initialize using the current argument values. An "Initialize()" call must precede all calls of the "Refresh()" method. A component must invoke the "Initialize()" method of all its adapted outputs at the end of the component's Prepare phase. In case of stacked adapted outputs, the adaptee must be initialized first.
Refresh() void Public	Causes the adapted output to refresh itself; it will be called by the adaptee, when it has been refreshed/updated. In the implementation of the "Refresh()" method, the adapted output should update its contents according to the changes in the adaptee. After updating itself the adapted output must call the "Refresh()" method for all its adapted outputs, so that whole the chain of outputs is refreshed.

6.10.1.2 The ITimeSpaceAdaptedOutput interface

An "ITimeSpaceAdaptedOutput" adds one or more data operations on top of those of an output item. It is in itself an "IBaseAdaptedOutput". The adapted output extends an output item with functionality such as spatial interpolation, temporal interpolation, unit conversion, etc.

The "ITimeSpaceAdaptedOutput" makes the "GetValues()" method and the Values attribute return an "ITimeSpaceValueSet" instead of an "IBaseValueSet".

6.10.1.3 The IAdaptedOutputFactory interface

An "IAdaptedOutputFactory" can be asked what types of adapted outputs are available, given a certain output item and a target input item. This is done by calling "GetAvailableAdaptedOutputIdentifiers()". The method returns a list of identifiers for the available types.

Table 30 Members of the IAdaptedOutputFactory interface

Member	Notes	Parameters
CreateAdaptedOutput() IBaseAdaptedOutput Public	Creates an "IBaseAdaptedOutput" which can adapt the values output by the "adaptee" so that they are provided in the form required by the target. The "adaptedOutputId" used must be one of the "IIdentifiable" instances returned by the "GetAvailableAdaptedOutputIds()" method. The returned "IBaseAdaptedOutput" must and will already be registered with the "adaptee" by calling the adaptee's "AddAdaptedOutput()" method.	**IIdentifiable** [in] adaptedOutputId The identifier of the adaptedOutput to create. **IBaseOutput** [in] adaptee The "IBaseOutput"to adapt. **IBaseInput** [in] target The "IBaseInput" to which the adapted output values will be provided. Can be null.
GetAvailableAdaptedOutputIds() IIdentifiable Public	Returns a list of identifiers of the available "IBaseAdaptedOutput"s that can transform the values output by the "adaptee" to match those requested by the "target". If the "target" is null, the identifiers of all "IBaseAdaptedOutput"s that can adapt the "adaptee" are returned. The "Adaptee" is the output exchange item whose values are to be adapted.	**IBaseOutput** [in] adaptee "IBaseOutput" to adapt. **IBaseInput** [in] target The "IBaseInput" to which the adapted output values will be provided. Can be null.

6.10.2 Requirements for Adapted Output interfaces

Requirement 10.1: Adapted Output/Base Adapted Output
/req/adapted-output/IBaseAdaptedOutput
An OpenMI component which supports the adaptation of its output values ***shall*** implement the IBaseAdaptedOutput interface according to the definition in Figure 14 and Table 29.

Requirement 10.2: Adapted Output/Times Space Adapted Output
/req/adapted-output/ITimeSpaceAdaptedOutput
A time-dependent OpenMI component which supports the adaptation of its output values ***shall*** implement the ITimeSpaceAdaptedOutput interface based on the definition in Figure 14 and Table 29.

Requirement 10.3: Adapted Output/Adapted Output Factory
/req/adapted-output/IAdaptedOutputFactory
To be able create instances of adaptors, an OpenMI component ***shall*** implement the IAdaptedOutputFactory interface based on the definition in Figure 15 and Table 30.

6.11 The Manage State interfaces

An OpenMI linkable component may implement two optional interfaces: the "IManageState" interface, which handles feedback loops by storing and restoring states (see Figure 16), and the "IByteStateConverter" interface, which provides the ability to make these states persistent.

Requirements Class 11: Manage State	
/req/manage-state	
Target type	OpenMI component
Dependency	
Requirement 11.1	/req/manage-state/IManageState
Requirement 11.2	/req/manage-state/IByteStateConverter

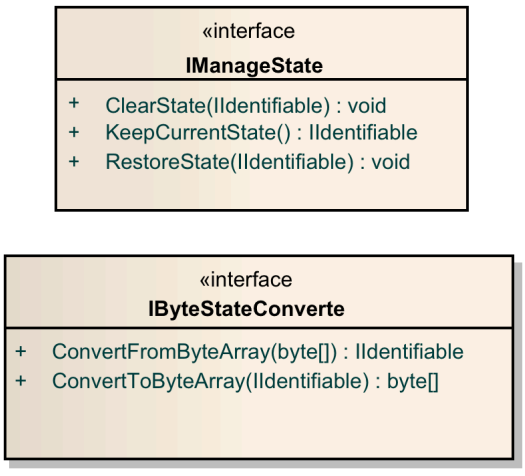


Figure 16 UML Diagram for Manage State

6.11.1 Attributes and methods of the Manage State interfaces

6.11.1.1 The IManageState interface

This optional interface can be implemented by components in addition to the "IBaseLinkableComponent" interface. It provides additional methods for handling a component's

state so that it can be saved, restored and cleared. The component may store its state by writing it to file or by keeping it in memory.

Table 31 Members of the IManageState interface

Member	Notes	Parameters
ClearState() void Public	Clears a state from the linkable component's memory. If the state identifier identified by "stateId" is not known by the linkable component then an IllegalArgumentException exception should be thrown.	**IIdentifiable** [in] stateId Identifier of the state to be cleared.
KeepCurrentState() IIdentifiable Public	Stores the linkable component's current State. Returns the identifier of the stored state.	
RestoreState() void Public	Restores the state identified by the parameter stateId. If the state identifier identified by stateId is not known by the linkable component an IllegalArgumentException exception should be thrown.	**IIdentifiable** [in] stateId Identifier of the state to be restored.

6.11.1.2 The IByteStateConverter interface

If the "IManageState" interface described above is implemented, then the "IByteStateConverter" may also be implemented if required. Its use is optional but depends upon the "IManageState" interface. The "IByteStateConverter" defines methods for converting the states as handled by the "IManageState" interface to and from a byte stream. This facilitates external modules, e.g. a GUI or an operational control system, to return or set a model's state as a simple bytestream that can be made persistent by writing it to file.

Table 32 Members of the IByteStateConverter interface

Member	Notes	Parameters
ConvertFromByteArray() IIdentifiable Public	Creates a state from a byte stream and returns the identifier of this state. The state does not become the current state of the "IBaseLinkableComponent". For state management the "IManageState" interface is to be used. Returns "IIdentifiable" identifying the state.	**byte[]** [in] byteArray State as a byte stream.
ConvertToByteArray() byte Public	Converts the state with the "stateId" into a byte stream. Returns the state identified by "stateId" as an array of bytes.	**IIdentifiable** [in] stateId Id of the state.

6.11.2 Requirements for Manage State interfaces

Requirement 11.1: Manage State/Manage State
/req/manage-state/IManageState
If an OpenMI component supports the optional managing of state, it ***shall*** implement the IManageState interface based on the definition in Table 31 and Figure 16.

Requirement 11.2: Manage State/Byte State Converter
/req/manage-state/IByteStateConverter
If an OpenMI component implements the IManageState interface, then it may also implement the IByteStateConverter interface if required. If the IByteStateConverter interface is implemented then the implementation ***shall*** be based on the definition in Table 32 and Figure 16.

6.12 Linkable Component

All interfaces mentioned above come together in the main interface of the OpenMI, the basic interface for accessing a model component. This "IBaseLinkableComponent" interface includes a section for initialization, a section for introspection and linkage configuration (the description of exchange items and the creation of links) and a section for run-time data exchange – see Figure 17**Error! Reference source not found.**.

Requirements Class 12: Linkable Component	
/req/linkable-component	
Target type	OpenMI component
Dependency	
Requirement 12.1	/req/linkable-component/IBaseLinkableComponent
Requirement 12.2	/req/linkable-component/ITimeSpaceComponent
Requirement 12.3	/req/linkable-component/ITimeExtension

Since all access to a component is through this interface, generic OpenMI implementation environments (e.g. GUIs) can be made independent of the underlying type of engine or component being used. This approach allows the addition of new components without modifications to the environment.

It is important to note that the OpenMI is non-exclusive; that is, it does not prevent a component implementing other interfaces as well. However, where data exchange is to be effected between components through the OpenMI, the interface by which it is achieved is the "IBaseLinkableComponent" interface. By having this one generic interface, the process of

assembling components into a composition is greatly simplified. Further, component developers can make changes within their component without impacting the rest of the composition - assuming, of course, that the component continues to meet its specification. In addition, sensitivity testing becomes much simpler because of the ease with which one component can be replaced by another.

The most important properties of the linkable component are those defining and describing its inputs and outputs; they determine what can potentially be exchanged and what will be exchanged in a specific context. What will be exchanged is established by linking an input item of one component (the consumer or target) to one of the outputs of another component (the provider or source). Data exchange is performed by invoking the "GetValues()" method of an output item. If the component has already computed the requested value, then the value can be returned immediately. If not, the "Update()" method will be invoked. This will cause the component to run until it can return the value, e.g. by progressing a simulation by one time step. The "Update()" method can be called repeatedly until the component reaches the end of its processing.

Quite often, the values produced by an output item are not in the form that the input item requires. In such situations, the result must be adapted by adding an adapted output item to the output item. The adapted outputs are usually provided by the linkable component but they can also be provided by other components.

For some situations, e.g. to enable iteration, it is useful if a linkable component can manage its state and a state management interface is provided for this purpose. The implementation of this interface, "IManageState", is optional. It is up to the code developer to decide if states need to be saved and, if so, which state-related data are 'saved' and where (e.g. in memory, a file or elsewhere).

There are situations, e.g. in operational forecasting systems, where the system may need to store one or more model states. For instance, a forecasting model, which has run ahead in time in order to construct a forecast, may need to revert to time 'now', so that it can it pick up any newly available sensor readings and then prepare the next forecast. This is supported by the "IManageState" and "IByteStateConverter" interfaces mentioned above.

Additional functionality can be achieved by implementing any appropriate extension to the interface.

An OpenMI-compliant component can also comply with one or more extensions, by implementing both the "IBaseLinkableComponent" interface and the extension interfaces to which it wishes to comply, e.g. the "ITimeExtension".

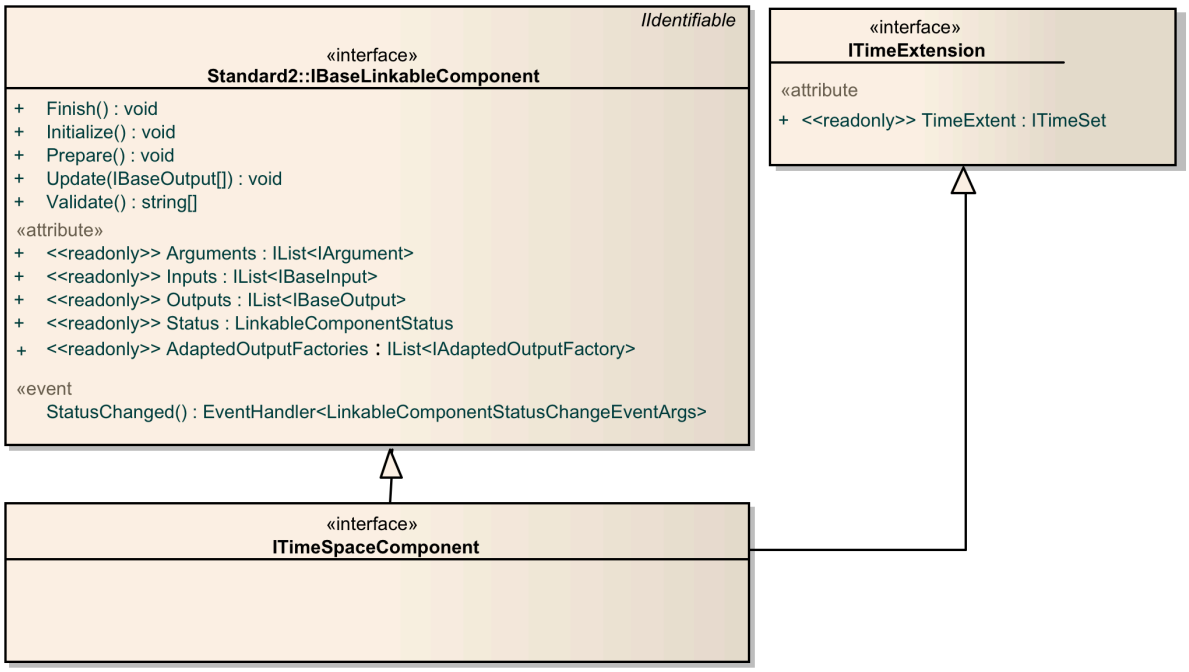


Figure 17 UML Diagram for Linkable Component

6.12.1 Attributes and methods for IBaseLinkableComponent interfaces

6.12.1.1 The IBaseLinkableComponent interface

The "IBaseLinkableComponent" is the main interface in the OpenMI standard.

Table 33 Members of the IBaseLinkableComponent interface

Member	Notes	Parameters
AdaptedOutputFactories IList<IAdaptedOutputFactory> Public	Returns a list of "IAdaptedOutputFactory", each of which allows the creation of an "IBaseAdaptedOutput" item. These are used to convert the provider's output to the form required by the requesting consumer. Factories can be added to and removed from the list thus allowing third-party factories and "IBaseAdaptedOutput" classes to be introduced.	
Arguments IList<IArgument> Public	Returns the arguments needed to let the component do its work. An unmodifiable list of (modifiable) arguments is returned which may be used to obtain information about the arguments and to set argument values. Validation of changes may be performed either when the changes occur (e.g. using notifications) or when the "Initialize()" method is called. Initialize will always be called before any call to the "Update()" method of the "IBaseLinkableComponent". This attribute must be available as soon as the linkable component instance is created. Arguments describe the arguments that can be set before the "Initialize()" method is called.	
Finish() void Public	This method is and must be invoked as the last of any methods in the "IBaseLinkableComponent" interface. This method must become accessible after the "Prepare()" method has been invoked. If this method is invoked before the "Prepare()" method has been invoked and the "LinkableComponent" cannot handle this, an exception must be thrown. Immediately after the method is invoked, it changes the linkable component's status to "LinkableComponentStatus.Finishing". Once "Finish()" is completed, the component changes its status to "LinkableComponentStatus.Finished" if it cannot be restarted, or "LinkableComponentStatus.Created" if it can.	
Initialize() void Public	Initializes the "LinkableComponent". The "Initialize()" method must be invoked before any other method or attribute of the "IBaseLinkableComponent" interface is	

Member	Notes	Parameters
	invoked or accessed, except for the "Arguments" attribute. Immediately after the method is invoked, it changes the linkable component's Status to "LinkableComponentStatus.Initializing". When the method is executed and an error occurs, the Status of the component will change to "LinkableComponentStatus.Failed", and an exception will be thrown. If the component initializes successfully, the status is changed to "LinkableComponentStatus.Initialized". When the "Initialize()" method has been finished and the Status is "LinkableComponentStatus.Initialized", the attributes "Id", "Caption", "Description", "Inputs" and "Outputs" will have been set, and the method "Validate()" can be called. It is only required that the method "Initialize()" be invoked once. If the "Initialize()" method is invoked more than once and the "LinkableComponent" cannot handle this; an exception must be thrown. Remarks: The method will typically populate the component based on the values specified in its arguments, which can be retrieved through the accessor method "IBaseLinkableComponent.Arguments". Settings can be used to read input files, allocate memory and organize input and output exchange items.	
Inputs IList<IBaseInput> Public	Returns the list of input items for which a component can receive values. Remarks: This attribute must be accessible after the "Initialize()" method has been invoked and until the "Validate()" method has been invoked. If this attribute is accessed before the "Initialize()" method has been invoked or after the "Validate()" method has been invoked and the "LinkableComponent" cannot handle this, an exception must be thrown. This method returns references to "IBaseInput" items. There is no guarantee that the list of objects is not altered by other components after it has been returned. It is the responsibility of the "LinkableComponent" to make sure that such possible alterations do	

Member	Notes	Parameters
	not subsequently corrupt the "LinkableComponent".	
Outputs IList<IBaseOutput> Public	Returns the list of output items for which a component can produce results. Remarks: This attribute must be accessible after the "Initialize()" method has been invoked and until the "Validate()" method has been invoked. If this attribute is accessed before the "Initialize()" method has been invoked or after the "Validate" method has been invoked and the "LinkableComponent" cannot handle this, an exception must be thrown. The list only contains the core "IBaseOutput" or "ITimeSpaceOutput" items of the component, not the "IBaseAdaptedOutput" or "ITimeSpaceAdaptedOutput" items that adapt the core output items. To obtain a complete list of outputs, traverse the chain of adapted outputs for each core output in the list. The "Outputs()" method basically returns references to "IBaseOutput" items. There is no guarantee that the list of objects is not altered by other components after it has been returned. It is the responsibility of the "LinkableComponent" to make sure that such possible alterations do not subsequently corrupt the "LinkableComponent".	
Prepare() void Public	Prepares the "IBaseLinkableComponent" for calls to the "Update()" method Before "Prepare()" is called, the component is not required to honour any type of action that retrieves values from the component. After "Prepare()" is called, the component must be ready to provide values. This method must be accessible after the "Initialize()" method has been invoked and until the "Finish()" method has been invoked. If this method is accessed before the "Initialize()" method has been invoked or after the "Finish()" method has been invoked and the "LinkableComponent" cannot handle this, an exception must be thrown. Immediately after the method is invoked, it changes the linkable component's status to "LinkableComponentStatus.Preparing". When the method has finished, the status of	

Member	Notes	Parameters
	the component is changed to either "LinkableComponentStatus.Updated" or "LinkableComponentStatus.Failed". It is only required that the "Prepare()" method can be invoked once. If the "Prepare()" method is invoked more than once and the "LinkableComponent" cannot handle this, an exception must be thrown.	
Status LinkableComponentStatus Public	Returns the current status of the linkable component. See "LinkableComponentStatus" in Figure 11 for the possible values. The first status that a component sets is "LinkableComponentStatus.Created". It is done as soon as it has been created. When a component has this status, "Arguments" is the only attribute that may be accessed.	
StatusChanged() EventHandler<LinkableComponentStatusChangeEventArgs> Public	The "StatusChanged" event is raised when the status of the component changes. See "LinkableComponentStatus" in Figure 11 for the possible states.	
Update() void Public	This method is called to let the component update itself and so reach its next state – see Figure 10. Immediately after the method is invoked, the linkable component's status changes to "LinkableComponentStatus.Updating". The type of actions a component takes during the "Update" method depends on the type of component. A numerical model that progresses in time will typically compute a time step. A database would typically look at the consumers of its output items, and perform one or more queries to be able to provide the values that the consumers require. For example, a GIS system would typically re-evaluate the values in a grid coverage, so that its output items can provide up-to-date values. If the "Update()" method is performed successfully, the component sets its status to "LinkableComponentStatus.Updated", unless after this update action the component is at the end of its computation, in which case its status will be set to "LinkableComponentStatus.Done". If during the "Update()" method a problem arises, the component sets its status to	**IBaseOutput[]** [in] **requiredOutput** This optional parameter lets the caller specify the specific output items that should be updated. If it is omitted or if the length is 0, the component will at least update its output items that have consumers, or all its output items, depending on the component's implementation.

Member	Notes	Parameters
	"LinkableComponentStatus.Failed" and throws an exception.	
Validate() string Public	Validates the populated instance of the "LinkableComponent". This method must be accessible after the "Initialize()" method has been invoked and until the "Finish()" method has been invoked. If this attribute is accessed before the "Initialize()" method has been invoked or after the "Finish()" method has been invoked and the "LinkableComponent" cannot handle this, an exception must be thrown. The method must be invoked after the various provider/consumer relations between this component's exchange items and the exchange items of other components have been added to the composition. Immediately after the method is invoked, it changes the linkable component's "Status" to "LinkableComponentStatus.Validating". When the "Validate()" method has finished, the status of the component will change to either "LinkableComponentStatus.Valid" or "LinkableComponentStatus.Invalid". Returns null or an array of strings of length null if there are no messages at all. If there are messages while the components Status is "LinkableComponentStatus.Valid", the messages are purely informative. If there are messages while the component's status is "LinkableComponentStatus.Invalid", at least one of the messages indicates a fatal error.	

6.12.1.2 The ITimeExtension interface

The ITimeExtension provides methods that apply only to time aware components.

Table 34 Members of the ITimeExtension interface

Member	Notes	Parameters
TimeExtent ITimeSet Public	Returns the "TimeExtent" attribute which describes in what time span the component can operate. This can be used to support the user when creating a composition.	

6.12.1.3 The ITimeSpaceComponent interface

The ITimeSpaceComponent is an "IBaseLinkableComponent" that also implements the ITimeExtension interface.

6.12.2 Requirements for Linkable Component interfaces

Requirement 12.1: Linkable Component/Base Linkable Component
/req/linkable-component/IBaseLinkableComponent
An OpenMI component ***shall*** implement the IBaseLinkableComponent interface based on the definition in Figure 17 and Table 33.

Requirement 12.2: Linkable Component/Time Space Component
/req/linkable-component/ITimeSpaceComponent
A time-dependent OpenMI component ***shall*** implement the ITimeSpaceComponent interface based on the definition in Figure 17 and Table 33.

Requirement 12.3: Linkable Component/Time Extension
/req/linkable-component/ITimeExtension
A time-dependent OpenMI component ***shall*** implement the ITimeExtension interface based on the definition in Figure 17 and Table 34.

Annex A Conformance Class Abstract Test Suite

An OpenMI Linkable Component encoding must satisfy the following characteristics to be conformant with this specification. Note that two encoded extensions are needed in order to actually execute the tests:

- The compiled OpenMI 2.0 interface specification, either C# or Java, referred to here as the 'interface binaries'.
- A testing environment, referred to here as the 'test tool', that facilitates loading an OpenMI linkable component and inspecting its behaviour, its input and output items, the quantities and element sets of these items, etc.

Currently, there are two testing tools available:

- The Pipistrelle end user environment as provided by the Fluid Earth project – see http://sourceforge.net/projects/fluidearth.
- The tool provided by the OpenMI Association's Technical Committee (OATC), the OATC Conformance Tool (OCT) – see http://www.openmi.org.

Test identifiers in the conformance test classes below are relative to http://www.opengis.net/spec/openmi/2.0/.

Conformance Class 1: Component Instantiation		
/conf/component-instantiation/		
Requirements	/req/component-instantiation	
Test 1.1	/conf/component-instantiation/ValidXML	
	Requirement	/req/component-instantiation/ValidXML
	Test purpose	To ensure a valid .OMI file has been provided for the candidate component.
	Test method	Visual inspection to check that one or more .OMI files are present. Validation of the OMI file against the xsd schema in Annex B.
	Test type	Basic

Conformance Class 2: Describable Identifiable		
/conf/describable-identifiable		
Requirements	describable-identifiable	
Test 2.1	/conf/describable-identifiable/IDescribable	
	Requirement	/req/describable-identifiable/IDescribable
	Test purpose	To ensure the candidate component implements the IDescribable interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 2.2	/conf/describable-identifiable/IIdentifiable	
	Requirement	/req/describable-identifiable/IIdentifiable
	Test purpose	To ensure the candidate component implements the IIdentifiable interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 3: Value Definition		
/conf/value-definition		
Requirements	/req/value-definition	
Test 3.1	/conf/value-definition/IValueDefinition	
	Requirement	/req/value-definition/IValueDefinition
	Test purpose	To ensure the candidate component implements the IValueDefinition interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 3.2	/conf/value-definition/IUnit	
	Requirement	/req/value-definition/IUnit
	Test purpose	To ensure the candidate component implements the IUnit interface
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 3.3	/conf/value-definition/IQuantity	
	Requirement	/req/value-definition/IQuantity
	Test purpose	To ensure the candidate component implements the IQuantity interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Test 3.4	/conf/value-definition/IQuality	
	Requirement	/req/value-definition/IQuality
	Test purpose	To ensure the candidate component implements the IQuality interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 3.5	/conf/value-definition/ICategory	
	Requirement	/req/value-definition/ICategory
	Test purpose	To ensure the candidate component implements the ICategory interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 3.6	/conf/value-definition/IDimension	
	Requirement	/req/value-definition/IDimension
	Test purpose	To ensure the candidate component implements the IDimension interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Test 3.7	/conf/value-definition/DimensionBase	
	Requirement	/req/value-definition/IDimensionBase
	Test purpose	To ensure the candidate component implements the DimensionBase enumeration.
	Test method	Visually inspect the code to ensure the enumeration has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 4: Spatial Definition		
/conf/spatial-definition		
Requirements	/req/spatial-definition	
Test 4.1	/conf/spatial-definition/ISpatialDefinition	
	Requirement	/req/spatial-definition/ISpatialDefinition
	Test purpose	To ensure the candidate component implements the IElementSet interface.
	Test method	Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 4.2	/conf/spatial-definition/IElementSet	
	Requirement	/req/spatial-definition/IElementSet
	Test purpose	To ensure the candidate component implements the IElementSet interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 4.3	/conf/spatial-definition/ElementType	
	Requirement	/req/spatial-definition/IElementType
	Test purpose	To ensure the candidate component implements the ElementType enumeration.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 5: Temporal Definition		
/conf/temporal-definition		
Requirements	/req/temporal-definition	
Test 5.1	/conf/temporal-definition/ITime	
	Requirement	/req/temporal-definition/ITime
	Test purpose	To ensure the candidate component implements the ITime interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 5.2	/conf/temporal-definition/ITimeSet	
	Requirement	/req/temporal-definition/ITimeSet
	Test purpose	To ensure the candidate component implements the ITimeSet interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 6: Value Set		
/conf/value-set		
Requirements	/req/value-set	
Test 6.1	/conf/value-set/IBaseValueSet	
	Requirement	/req/value-set/IBaseValueSet
	Test purpose	To ensure the candidate component implements the IBaseValueSet interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 6.2	/conf/value-set/ITimeSpaceValueSet	
	Requirement	/req/value-set/ITimeSpaceValueSet
	Test purpose	To ensure the candidate component implements the ITimeSpaceValueSet interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 7: Argument		
/conf/argument		
Requirements	/req/argument	
Test 7.1	/conf/argument/IArgument	
	Requirement	/req/argument/ IArgument
	Test purpose	To ensure the candidate component implements the IArgument interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 8: Linkable Component Status		
/conf/linkable-component-status		
Requirements	/req/linkable-component-status	
Test 8.1	/conf/linkable-component-status/LinkableComponentStatus	
	Requirement	/req/linkable-component-status/LinkableComponentStatus
	Test purpose	To ensure the candidate component implements the LinkableComponentStatus enumeration.
	Test method	Visually inspect the code to ensure the enumeration has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 8.2	/conf/linkable-component-status/LinkableComponentStatus	
	Requirement	/req/linkable-component-status/LinkableComponentBehaviour
	Test purpose	To ensure the candidate component implements the required behaviour.
	Test method	Visually inspect the code to ensure the behaviour required has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 8.3	/conf/linkable-component-status/EventStatusChanged	
	Requirement	/req/linkable-component-status/EventStatusChanged
	Test purpose	To ensure the candidate component implements the IBaseLinkableComponent.StatusChanged event and the LinkableComponentStatusChangeEventArgs class.
	Test method	Visually inspect the code to ensure the event and class have been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 9: Exchange Item		
/conf/exchange-item		
Requirements	/req/	
Test 9.1	/conf/exchange-item/IBaseExchangeItem	
	Requirement	/req/exchange-item/IBaseExchangeItem
	Test purpose	To ensure the candidate component implements the IBaseExchangeItem interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 9.2	/conf/exchange-item/IBaseInput	
	Requirement	/req/exchange-item/IBaseInput
	Test purpose	To ensure the candidate component implements the IBaseInput interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 9.3	/conf/exchange-item/IBaseOutput	
	Requirement	/req/exchange-item/IBaseOutput
	Test purpose	To ensure the candidate component implements the IBaseOutput interface
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Test 9.4	/conf/exchange-item/ITimeSpaceExchangeItem	
	Requirement	/req/exchange-item/ITimeSpaceExchangeItem
	Test purpose	To ensure the candidate component implements the ITimeSpaceExchangeItem interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 9.5	/conf/exchange-item/ITimeSpaceOutput	
	Requirement	/req/exchange-item/ITimeSpaceOutput
	Test purpose	To ensure the candidate component implements the ITimeSpaceOutput interface
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 9.6	/conf/exchange-item/ITimeSpaceInput	
	Requirement	/req/exchange-item/ITimeSpaceInput
	Test purpose	To ensure the candidate component implements the ITimeSpaceInput interface.
	Test method	Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Test 9.7	/conf/exchange-item/EventExchangeItemChanged	
	Requirement	/req/exchange-item/EventExchangeItemChanged**Error! Reference source not found.**
	Test purpose	To ensure the candidate component implements the ExchangeItemChanged event where the implementation language supports a mechanism for handling events.
	Test method	Visually inspect the code to ensure the event has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 10: Adapted Output		
/conf/adapted-output		
Requirements	/req/adapted-output	
Test 10.1	/conf/adapted-output/IBaseAdaptedOutput	
	Requirement	/req/adapted-output/IBaseAdaptedOutput
	Test purpose	To ensure the candidate component implements the IBaseAdaptedOutput interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 10.2	/conf/adapted-output/ITimeSpaceAdaptedOutput	
	Requirement	/req/adapted-output/ITimeSpaceAdaptedOutput
	Test purpose	To ensure the candidate component implements the ITimeSpaceadAptedOutput interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 10.3	/conf/adapted-output/IAdaptedOutputFactory	
	Requirement	/req/adapted-output/IAdaptedOutputFactory
	Test purpose	To ensure the candidate component implements the IAdaptedOutputFactory interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 11: Manage State		
/conf/manage-state		
Requirements	/req/manage-state	
Test 11.1	/conf/manage-state/IManageState	
	Requirement	/req/manage-state/I
	Test purpose	To ensure the candidate component implements the IManageState interface
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 11.2	/conf/manage-state/IByteStateConverter	
	Requirement	/req/manage-state/IByteStateConverter
	Test purpose	To ensure the candidate component implements the IByteStateConverter interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic

Conformance Class 12: Linkable Component		
/conf/linkable-component		
Requirements	/req/linkable-component **Error! Reference source not found.**	
Test 12.1	/conf/linkable-component/IBaseLinkableComponent	
	Requirement	/req/linkable-component/IBaseLinkableComponent
	Test purpose	To ensure the candidate component implements the IBaseLinkableComponent interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	Basic
Test 12.2	/conf/linkable-component/ITimeSpaceComponent	
	Requirement	/req/linkable-component/ITimeSpaceComponent
	Test purpose	To ensure the candidate component implements the ITimeSpaceComponent interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	basic
Test 12.3	/conf/linkable-component/ITimeExtension	
	Requirement	/req/linkable-component/ITimeExtension
	Test purpose	To ensure the candidate component implements the ITimeExtension interface.
	Test method	Visually inspect the code to ensure the interface has been implemented. Compile the code against the openmi/2.0/req interface binaries.
	Test type	basic

Annex B XSD Schema for OMI File

```
<?xml version="1.0"?>
<!--
 ~ Copyright (c) 2005-2010, OpenMI Association
 ~ <http://www.openmi.org/>
 ~
 ~ This file is part of openmi-standard2-2.0.0-beta1.jar
 ~
 ~ openmi-standard2.jar is free software; you can redistribute it and/or
 ~ modify it under the terms of the Lesser GNU General Public License as
 ~ published by the Free Software Foundation; either version 3 of the
 ~ License, or (at your option) any later version.
 ~
 ~ openmi-standard2.jar is distributed in the hope that it will be useful,
 ~ but WITHOUT ANY WARRANTY; without even the implied warranty of
 ~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the Lesser GNU
 ~ General Public License for more details.
 ~
 ~ You should have received a copy of the Lesser GNU General Public License
 ~ along with this program.  If not, see <http://www.gnu.org/licenses/>.
 -->
<!-- OpenMI Linkable component entry point to instantiate the object-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.openmi.org/v2_0" targetNamespace="http://www.openmi.org/v2_0"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="2.0.0.0">
  <xsd:simpleType name="supportedPlatformEnum">
    <xsd:annotation>
      <xsd:documentation>
        Enumeration of all possible operating system platforms a LinkableComponent can be run on.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="win"/>
      <xsd:enumeration value="unix"/>
      <xsd:enumeration value="linux"/>
      <xsd:enumeration value="mac"/>
      <xsd:enumeration value="win32"/>
      <xsd:enumeration value="win64"/>
      <xsd:enumeration value="unix32"/>
```

```
    <xsd:enumeration value="unix64"/>
    <xsd:enumeration value="linux32"/>
    <xsd:enumeration value="linux64"/>
    <xsd:enumeration value="mac32"/>
    <xsd:enumeration value="mac64"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="LinkableComponent" type="LinkableComponentComplexType"/>
<xsd:complexType name="LinkableComponentComplexType">
  <xsd:all>
    <xsd:element name="Arguments" minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Argument" minOccurs="0" maxOccurs="unbounded">
            <xsd:annotation>
              <xsd:documentation>
                Arguments used for component instantiation
              </xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
              <xsd:attribute name="Key" type="xsd:string" use="required" form="unqualified">
                <xsd:annotation>
                  <xsd:documentation>
                    Attribute key for which a value is provided
                  </xsd:documentation>
                </xsd:annotation>
              </xsd:attribute>
              <xsd:attribute name="ReadOnly" type="xsd:boolean" use="optional" form="unqualified">
                <xsd:annotation>
                  <xsd:documentation>
                    Flag indicating if the value of the attribute may be edited by the user
                  </xsd:documentation>
                </xsd:annotation>
              </xsd:attribute>
              <xsd:attribute name="Value" type="xsd:string" use="required" form="unqualified">
                <xsd:annotation>
                  <xsd:documentation>
                    Attribute value for the associated attribute key
                  </xsd:documentation>
                </xsd:annotation>
```

```
          </xsd:attribute>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Platforms" minOccurs="0">
  <xsd:annotation>
   <xsd:documentation>
    Optional list of operating systems the LinkableComponent can be run on.
   </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Platform" type="supportedPlatformEnum" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:all>
<xsd:attribute name="Type" type="xsd:string" form="unqualified">
 <xsd:annotation>
  <xsd:documentation>
   Class to be instantiated to create a LinkableComponent-object
  </xsd:documentation>
 </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="Assembly" type="xsd:string" use="optional" form="unqualified">
 <xsd:annotation>
  <xsd:documentation>
   DotNet assembly that can instantiate the class
  </xsd:documentation>
 </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="JavaArchive" type="xsd:string" use="optional" form="unqualified">
 <xsd:annotation>
  <xsd:documentation>
   JavaArchive that can instantiate the class
  </xsd:documentation>
 </xsd:annotation>
</xsd:attribute>
```

```
  </xsd:complexType>
</xsd:schema>
```

Annex C XSD Schema for the Compliancy Information File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2009 sp1 (http://www.altova.com) by ICT (Stichting Deltares) -->
<!--
 ~ Copyright (c) 2005-2010, OpenMI Association
 ~ <http://www.openmi.org/>
 ~
 ~ This file is part of openmi-standard2-2.0.0-beta1.jar
 ~
 ~ openmi-standard2.jar is free software; you can redistribute it and/or
 ~ modify it under the terms of the Lesser GNU General Public License as
 ~ published by the Free Software Foundation; either version 3 of the
 ~ License, or (at your option) any later version.
 ~
 ~ openmi-standard2.jar is distributed in the hope that it will be useful,
 ~ but WITHOUT ANY WARRANTY; without even the implied warranty of
 ~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the Lesser GNU
 ~ General Public License for more details.
 ~
 ~ You should have received a copy of the Lesser GNU General Public License
 ~ along with this program.  If not, see <http://www.gnu.org/licenses/>.
 -->
<!-- Description of OpenMI Linkable component capabilities and availability -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.openmi.org" targetNamespace="http://www.openmi.org"
 elementFormDefault="qualified" attributeFormDefault="unqualified" version="2.0.0.0">
 <xsd:element name="openMICompliancyInfo" type="OpenMICompliancyComplexType">
  <xsd:annotation>
   <xsd:documentation>
    Schema to provide background information on OpenMI capabilities of the component.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:element>
 <xsd:complexType name="OpenMICompliancyComplexType">
  <xsd:sequence>
   <xsd:element name="generalSoftwareInfo" type="GeneralSoftwareInfoComplexType">
    <xsd:annotation>
     <xsd:documentation>
```

```
        general information on the component and its provider
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name="supportForOpenMI" type="SupportForOpenMIComplexType">
    <xsd:annotation>
      <xsd:documentation>the OpenMI capabilities of the component</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GeneralSoftwareInfoComplexType">
 <xsd:all>
  <xsd:element name="component">
    <xsd:annotation>
      <xsd:documentation>background info on the software component</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
              problem solving capabilities and/or domain(s) supported
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="url" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>URL describing software</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="version" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="contactInfo">
    <xsd:annotation>
      <xsd:documentation>contact details to the software provider</xsd:documentation>
    </xsd:annotation>
```

```
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="supplierName" type="xsd:string"/>
        <xsd:element name="contactPerson" type="xsd:string" minOccurs="0"/>
        <xsd:element name="postalAddress" type="xsd:string" minOccurs="0"/>
        <xsd:element name="supplierEmail" type="xsd:string"/>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
     <xsd:element name="availability">
      <xsd:annotation>
       <xsd:documentation>describe availability / distribution conditions</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="comment" type="xsd:string" minOccurs="0">
         <xsd:annotation>
          <xsd:documentation>comments on the availability</xsd:documentation>
         </xsd:annotation>
        </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="component" type="componentAvailabilityEnum" use="required"/>
       <xsd:attribute name="source" type="sourceAvailabilityEnum" use="required"/>
      </xsd:complexType>
     </xsd:element>
    </xsd:all>
   </xsd:complexType>
   <xsd:complexType name="SupportForOpenMIComplexType">
    <xsd:all>
     <xsd:element name="compliancy">
      <xsd:annotation>
       <xsd:documentation>
        Identifies the technologies and version(s) of the OpenMI Standard supported.
       </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="openMIStandardVersion"
         type="VersionNumberEnumType" maxOccurs="unbounded">
         <xsd:annotation>
```

```
            <xsd:documentation>Available for following versions of the OpenMI Standard.</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="optionalInterfaces">
          <xsd:annotation>
            <xsd:documentation>
              Specification of the optional OpenMI base interfaces supported by the component.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="IManageState">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="comment" type="xsd:string" minOccurs="0">
                      <xsd:annotation>
                        <xsd:documentation>e.g.  in-memory only</xsd:documentation>
                      </xsd:annotation>
                    </xsd:element>
                  </xsd:sequence>
                  <xsd:attribute name="supported" type="xsd:boolean" use="required"/>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="IByteStateConvertor">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="comment" type="xsd:string" minOccurs="0">
                      <xsd:annotation>
                        <xsd:documentation>
                          e.g.  from in-memory stored state only
                        </xsd:documentation>
                      </xsd:annotation>
                    </xsd:element>
                  </xsd:sequence>
                  <xsd:attribute name="supported" type="xsd:boolean" use="required"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
```

```
</xsd:element>
<xsd:element name="optionalExtensions">
 <xsd:annotation>
  <xsd:documentation>
   Specification of the optional OpenMI extensions supported by the component.
  </xsd:documentation>
 </xsd:annotation>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="OpenMITimeSpaceExtension">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="comment" type="xsd:string" minOccurs="0">
       <xsd:annotation>
        <xsd:documentation>
         e.g.  additional notes on the implementation of the extension
        </xsd:documentation>
       </xsd:annotation>
      </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="supported" type="xsd:boolean" use="required">
      <xsd:annotation>
       <xsd:documentation>
        specify true when the component implements all interfaces
        from the OpenMI TimeSpace Extension
       </xsd:documentation>
      </xsd:annotation>
     </xsd:attribute>
     <xsd:attribute name="spatialReferenceSystemWkt"
            type="xsd:string" use="optional" default="">
      <xsd:annotation>
       <xsd:documentation>
        The spatial reference system for all element sets, unless indicated otherwise
       </xsd:documentation>
      </xsd:annotation>
     </xsd:attribute>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
```

```
      </xsd:element>
      <xsd:element name="programmingLanguage"
            type="ProgrammingLanguageTypeEnum" maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            Available for following programming languages
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="platform"
          type="supportedPlatformEnum" maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>Available for following computer platforms.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="comment" type="xsd:string" minOccurs="0">
        <xsd:annotation>
          <xsd:documentation>Any comment(s) on the compliancy</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="arguments">
  <xsd:annotation>
    <xsd:documentation>
      Arguments needed to configure the component.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="argument"
          type="ArgumentComplexType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="exchangeItems">
  <xsd:annotation>
    <xsd:documentation>
      Definition of input and output data of the component.
```

```
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="adaptorFactories" minOccurs="0">
       <xsd:annotation>
        <xsd:documentation>
         Factories the component has available for creating exchange item adaptors.
        </xsd:documentation>
       </xsd:annotation>
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="adaptedOutputFactory"
           type="AdaptedOutputFactoryComplexType"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
      <xsd:choice maxOccurs="unbounded">
       <xsd:element name="input" type="ExchangeItemComplexType">
        <xsd:annotation>
         <xsd:documentation>
          Definition of input data consumed by the component.
         </xsd:documentation>
        </xsd:annotation>
       </xsd:element>
       <xsd:element name="output" type="ExchangeItemComplexType">
        <xsd:annotation>
         <xsd:documentation>
          Definition of output data produced by the component.
         </xsd:documentation>
        </xsd:annotation>
       </xsd:element>
      </xsd:choice>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:all>
</xsd:complexType>
<xsd:complexType name="ArgumentComplexType">
 <xsd:sequence>
```

```
  <xsd:element name="description" type="xsd:string" minOccurs="0">
    <xsd:annotation>
      <xsd:documentation>Descriptive information on the argument</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string" use="optional" form="unqualified">
  <xsd:annotation>
    <xsd:documentation>
      name that will be used when representing the argument on the screen
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="optional" type="xsd:boolean" use="optional" form="unqualified">
  <xsd:annotation>
    <xsd:documentation>
      indication wether the argument can be ommitted or not
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="readOnly" type="xsd:boolean" use="optional" form="unqualified">
  <xsd:annotation>
    <xsd:documentation>
      indication wether the argument can be modified after it is read from XML or not
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="defaultValue" type="xsd:string">
  <xsd:annotation>
    <xsd:documentation>
      sensible default value for the argument to be used when the Value attribute is empty
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="possibleValues">
  <xsd:annotation>
    <xsd:documentation>
      String representations of all possible values for the argument
    </xsd:documentation>
  </xsd:annotation>
```

```
    <xsd:simpleType>
      <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="AdaptedOutputFactoryComplexType">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>
          Descriptive information on the adapted output factory
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="availableAdaptor" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="description" type="xsd:string" minOccurs="0">
            <xsd:annotation>
              <xsd:documentation>Descriptive information on the adaptor</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="ExchangeItemComplexType">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>Descriptive information on the exchange time</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:choice>
      <xsd:element name="valuedefinition" type="ValueDefinitionComplexType"/>
      <xsd:element name="quantity" type="QuantityComplexType"/>
      <xsd:element name="quality" type="QualityComplexType"/>
```

```
    </xsd:choice>
    <xsd:element name="elementSet" type="ElementSetComplexType" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="ValueDefinitionComplexType">
  <xsd:annotation>
    <xsd:documentation>Definition of very specific value types.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>Descriptive information on the quality</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="valueType" type="xsd:string" use="optional" default="object"/>
</xsd:complexType>
<xsd:complexType name="QualityComplexType">
  <xsd:annotation>
    <xsd:documentation>Definition of OpenMI qualitative data.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>Descriptive information on the quality</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="category" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="description" type="xsd:string" minOccurs="0">
            <xsd:annotation>
              <xsd:documentation>Descriptive information on the category</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="value" type="xsd:string"/>
      </xsd:complexType>
```

```
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="valueType" type="xsd:string" use="optional" default="string"/>
  <xsd:attribute name="ordered" type="xsd:boolean"/>
</xsd:complexType>
<xsd:complexType name="QuantityComplexType">
  <xsd:annotation>
    <xsd:documentation>Definition of OpenMI quantitative data.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>Descriptive information on the quantity</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="dimension" type="DimensionComplexType"/>
  </xsd:sequence>
  <xsd:attribute name="valueType" type="xsd:string" use="optional" default="double"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="description" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="DimensionComplexType">
  <xsd:attribute name="length" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="mass" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="time" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="electricCurrent" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="temperature" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="amountOfSubstance" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="luminousIntensity" type="xsd:double" use="optional" default="0.0"/>
  <xsd:attribute name="currency" type="xsd:double" use="optional" default="0.0"/>
</xsd:complexType>
<xsd:complexType name="ElementSetComplexType">
  <xsd:annotation>
    <xsd:documentation>
      Optional: Only specified for TimeSpace specific exchange items.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="0">
      <xsd:annotation>
```

```
      <xsd:documentation>Descriptive information on the element set</xsd:documentation>
     </xsd:annotation>
    </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string" use="optional"/>
   <xsd:attribute name="spatialReferenceSystemWkt" type="xsd:string" use="optional"/>
   <xsd:attribute name="elementType" type="elementTypeEnum" use="required">
    <xsd:annotation>
     <xsd:documentation>elementType of the elementSet</xsd:documentation>
    </xsd:annotation>
   </xsd:attribute>
  </xsd:complexType>
  <xsd:simpleType name="VersionNumberEnumType">
   <xsd:annotation>
    <xsd:documentation>
     Enumeration of all official OpenMI Standard version numbers.
    </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="1.4"/>
    <xsd:enumeration value="2.0"/>
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ProgrammingLanguageTypeEnum">
   <xsd:annotation>
    <xsd:documentation>
     Enumeration of all possible programming languages supported by OpenMI.
    </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="dotNet"/>
    <xsd:enumeration value="Java"/>
   </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="supportedPlatformEnum">
   <xsd:annotation>
    <xsd:documentation>
     Enumeration of all possible operating system platforms a LinkableComponent can be run on.
    </xsd:documentation>
   </xsd:annotation>
```

```
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="all"/>
    <xsd:enumeration value="win"/>
    <xsd:enumeration value="unix"/>
    <xsd:enumeration value="linux"/>
    <xsd:enumeration value="mac"/>
    <xsd:enumeration value="win32"/>
    <xsd:enumeration value="win64"/>
    <xsd:enumeration value="unix32"/>
    <xsd:enumeration value="unix64"/>
    <xsd:enumeration value="linux32"/>
    <xsd:enumeration value="linux64"/>
    <xsd:enumeration value="mac32"/>
    <xsd:enumeration value="mac64"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="componentAvailabilityEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="commercial"/>
    <xsd:enumeration value="restricted"/>
    <xsd:enumeration value="free"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="sourceAvailabilityEnum">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="private"/>
    <xsd:enumeration value="restricted"/>
    <xsd:enumeration value="available"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="elementTypeEnum">
  <xsd:annotation>
    <xsd:documentation>
      Optional: Used by TimeSpace specific exchange items, to specify the element set type.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="IDBased"/>
    <xsd:enumeration value="Point"/>
    <xsd:enumeration value="PolyLine"/>
```

```
      <xsd:enumeration value="Polygon"/>
      <xsd:enumeration value="Polyhedron"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Annex D OpenMi Association Intellectual Property Rights Policy, Trademark and Licences

The OpenMI Association as a legal body will ensure and safeguard the copyrights and intellectual property rights (IPR) of the OpenMI Standard and any related products created by the OpenMI Association. It has registered the logo and name "**OpenMI**" as a trademark and has had the domain names (www.openmi.org and www.openMI.com) registered.

The OpenMI Standard is provided under the Lesser General Public Licence (LGPL). Other products of the OpenMI Asociation (e.g. a Software Develop Kit (SDK) and Graphical User Interface (GUI)) are or will be made available under suitable open source licence conditions. OpenMI-compliant tools and software developed by third parties remain the property of their respective developers. The OpenMI association will impose no restrictions on the use of the OpenMI for research or commercial purposes, and will not impose any royalty charges or licence fees.

Bibliography

Moore, R., Gijsbers, P., Fortune, D., Gregersen, J., Blind, M., Grooss, J., et al. (2010). *Scope for the OpenMI (Version 2.0).* OpenMI Association. Delft: OpenMI Association.

OpenMI Association. (2010). *Migrating Models for the OpenMI (Version 2.0).* OpenMI Association, OpenMI Association Technical Committee (OATC). Delft: OpenMI Association.

OpenMI Association. (2010). *OpenMI Standard 2 Reference for the OpenMI (Version 2.0).* OpenMI Association, OpenMI Association Technical Committee. Delft: OpenMI Association.

OpenMI Association. (2010). *OpenMI Standard 2 Specification for the OpenMI (Version 2.0).* OpenMI Association, OpenMI Association Technical Committee. Delft: OpenMI Association.

OpenMI Association. (2010). *The OpenMI 'in a Nutshell' for the OpenMI (Version 2.0).* OpenMI Association, OpenMI Association Technical Committee. Delft: OpenMI Association.

OpenMI Association. (2010). *What's New in OpenMI 2.0.* OpenMI Association, OpenMI Association Technical Committee. Delft: OpenMI Association.

Revision History

Date	Release	Author	Paragraph modified	Description
	1.0.0	OATC		1st Release