Open Geospatial Consortium

Approval Date: 2013-xx-xx

Publication Date: 2013-xx-xx

External OGC identifier: **http://www.opengis.net/doc/is/openmi/2.0**

Reference number of this document: 11-014r1

Version:2.0

Category: Candidate OpenGIS® Standard

Editors: Stanislav Vanecek

Current Document Date: January 11, 2013

 OpenMI 2.0 Implementation Standard

**Copyright notice**

Copyright © 2013 Open Geospatial Consortium.

To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>.

**Warning**

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

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® Publicly Available Standard

Document subtype: if applicable

Document stage: Draft

Document language:  English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications

**Contents**

[(i) Abstract vi](#_Toc343701548)

[(ii) Keywords vi](#_Toc343701549)

[(iii) Preface vi](#_Toc343701550)

[(iv) Submitting organizations vi](#_Toc343701551)

[(v) Submitters vi](#_Toc343701552)

[1 Scope 1](#_Toc343701553)

[1.1 Background 1](#_Toc343701554)

[1.2 What is it for? 1](#_Toc343701555)

[1.3 What is it? 1](#_Toc343701556)

[1.4 What does it do? 1](#_Toc343701557)

[1.5 How is the OpenMI implemented? 2](#_Toc343701558)

[1.6 Further reading 3](#_Toc343701559)

[1.7 Document structure 3](#_Toc343701560)

[2 Conformance 4](#_Toc343701561)

[3 Normative References 5](#_Toc343701562)

[4 Terms and Definitions 6](#_Toc343701563)

[5 Conventions 8](#_Toc343701564)

[5.1 Symbols (and abbreviated terms) 8](#_Toc343701565)

[5.2 Unified modelling language (UML) 8](#_Toc343701566)

[5.3 Extensible Markup Language (XML) 8](#_Toc343701567)

[6 OpenMI Requirements classes 9](#_Toc343701568)

[6.1 Component Instantiation 9](#_Toc343701569)

[6.2 Described and Identifiable Entries 10](#_Toc343701570)

[6.3 Value Definition 12](#_Toc343701571)

[6.4 Spatial Definition 18](#_Toc343701572)

[6.5 Temporal Definition 24](#_Toc343701573)

[6.6 Value Sets 27](#_Toc343701574)

[6.7 Argument 32](#_Toc343701575)

[6.8 Component status 34](#_Toc343701576)

[6.8.1 Events 38](#_Toc343701577)

[6.9 Exchange item 40](#_Toc343701578)

[6.9.1 ITimeSpaceExchangeItem 43](#_Toc343701579)

[6.9.2 IBaseInput 44](#_Toc343701580)

[6.9.3 ITimeSpaceInput 44](#_Toc343701581)

[6.9.4 IBaseOutput 44](#_Toc343701582)

[6.9.5 ITimeSpaceOutput 46](#_Toc343701583)

[6.10 Adapted Outputs 48](#_Toc343701584)

[6.10.1 IBaseAdaptedOutput 50](#_Toc343701585)

[6.10.2 ItimeSpaceAdaptedOutput 51](#_Toc343701586)

[6.10.3 AdaptedOutputFactory 51](#_Toc343701587)

[6.11 Manage State 53](#_Toc343701588)

[6.11.1 IManageState 54](#_Toc343701589)

[6.11.2 IByteStateConverter 55](#_Toc343701590)

[6.12 LinkableComponent 56](#_Toc343701591)

[6.12.1 ILinkableComponent 58](#_Toc343701592)

[6.12.2 ITimeSpaceComponent 63](#_Toc343701593)

[6.12.3 ITimeSpace 63](#_Toc343701594)

[Annex A Conformance Class Abstract Test Suite 65](#_Toc343701595)

[Annex B XSD schema for OMI File 78](#_Toc343701596)

[Bibliography 80](#_Toc343701597)

[Revision history 81](#_Toc343701598)

**List of Figures**

[Figure 1 Linkages between components and the use of adaptors 2](file:///W%3A%5CAdmin%5CTyping%5CSharratt%5CRogers%20Document%201.docx#_Toc343602719)

[Figure 2 A graphical view of the OMI file structure 9](#_Toc343602720)

[Figure 3 UML for IDescribable and IIdentifiable 11](file:///W%3A%5CAdmin%5CTyping%5CSharratt%5CRogers%20Document%201.docx#_Toc343602721)

[Figure 4 UML diagram for Value Definition 13](#_Toc343602722)

[Figure 5 UML for Spatial Definition 20](#_Toc343602723)

[Figure 6 UML Diagram for Temporal Definition 25](#_Toc343602724)

[Figure 7 UML Diagram for Value Set 28](#_Toc343602725)

[Figure 8 Illustration of directions to interpret positive values of fluxes, levels and depths 29](#_Toc343602726)

[Figure 9 ML Diagram for Argument 32](#_Toc343602727)

[Figure 10 Component Status Change Diagram 35](#_Toc343602728)

[Figure 11 UML Diagram for Component Status 36](#_Toc343602729)

[Figure 12 UML Diagram for Exchange Item 42](#_Toc343602730)

[Figure 13 UML Diagram for Adapted Output 49](#_Toc343602731)

[Figure 14 UML Diagram for Adapted Output Factory 49](#_Toc343602732)

[Figure 15 UML Diagram for Manage State 54](#_Toc343602733)

[Figure 16 UML Diagram for LinkableComponent 58](file:///W%3A%5CAdmin%5CTyping%5CSharratt%5CRogers%20Document%201.docx#_Toc343602734)

**List of Tables**

[Table 1 Operations for IDescribable 11](#_Toc343602735)

[Table 2 Operations for IIdentifiable 11](#_Toc343602736)

[Table 3 Base units and dimension base in the OpenMI (derived from SI) 14](#_Toc343602737)

[Table 4 Operations of IValueDefinition 15](#_Toc343602738)

[Table 5 Operations of IUnit 15](#_Toc343602739)

[Table 6 Operations for IQuantity 15](#_Toc343602740)

[Table 7 Operations of IQuality 15](#_Toc343602741)

[Table 8 Operations of ICategory 15](#_Toc343602742)

[Table 9 Operations of IDimension 16](#_Toc343602743)

[Table 10 Attributes of DimensionBase 17](#_Toc343602744)

[Table 11 Dependencies for ISpatialDefinition 20](#_Toc343602745)

[Table 12 Operations of ISpatialDefinition 21](#_Toc343602746)

[Table 13 Operations of IElementSet 22](#_Toc343602747)

[Table 14 Enumerator ElementType 24](#_Toc343602748)

[Table 15 Operations of ITimeSet 26](#_Toc343602749)

[Table 16 Operation for IbaseValue Sets 30](#_Toc343602750)

[Table 17 Operations of ITimespaceValueSets 31](#_Toc343602751)

[Table 18 Operations of IArgument 33](#_Toc343602752)

[Table 19 Enumeration ComponentStatus 37](#_Toc343602753)

[Table 20 Operations for ExchangeItemChangeEventArgs 38](#_Toc343602754)

[Table 21 Operations of LinkableComponentStatusChangeEventArgs 39](#_Toc343602755)

[Table 22 Operation for IBaseExchangeItem 43](#_Toc343602756)

[Table 23 Operations for ITimeSpaceExchangeItem 43](#_Toc343602757)

[Table 24 Operations for IBaseInput 44](#_Toc343602758)

[Table 25 Operations for ItimeSpaceInput 44](#_Toc343602759)

[Table 26 Operations for IBasedOutput 45](#_Toc343602760)

[Table 27 Operations for ITimeSpaceOutput 47](#_Toc343602761)

[Table 28 Operations for IBaseAdaptedOutput 50](#_Toc343602762)

[Table 29 Operations for IAdaptedOutputFactory 52](#_Toc343602763)

[Table 30 Operations for IManageState 55](#_Toc343602764)

[Table 31 Operations for IByteStateConverter 55](#_Toc343602765)

[Table 32 Operations for IBaseLinkableComponent 59](#_Toc343602766)

[Table 33 Operation of ITimeSpace 64](#_Toc343602767)

# 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. The creation of OpenMI was 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 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 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.

# Keywords

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

# 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.*

# Submitting organizations

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

* Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia
* OpenMI Association (OA), International

# Submitters

All questions regarding this submission should be directed to the editor or the submitters:

| Name | Representing | OGC member |
| --- | --- | --- |
| Standa Vanecek | OA and DHI | No |
| [Stef Hummel](https://publicwiki.deltares.nl/display/~hummel) | OA and [Deltares](http://www.deltares.nl/) | Yes |
| [Adrian Harper](https://publicwiki.deltares.nl/display/~harper) | OA and [Innovyze](http://www.innovyze.com/) | No |
| [Gennadii Donchyts (Gena)](https://publicwiki.deltares.nl/display/~donchyt) | OA and [Deltares](http://www.deltares.nl/) | Yes |
| [Peter Gijsbers](https://publicwiki.deltares.nl/display/~gijsber) | OA and [Deltares](http://www.deltares.nl/) | Yes |
| [Jesper Grooss](https://publicwiki.deltares.nl/display/~jgr%40dhigroup.com) | OA and [DHI](http://www.dhigroup.com/) | No |
| Johan Hartnack | OA and [DHI](http://www.dhigroup.com/) | No |
| [Rob Knapen](https://publicwiki.deltares.nl/display/~rob.knapen%40wur.nl) | OA and [Alterra](http://www.alterra.wur.nl/UK/) | No |
| Onno Roosenschoon | OA and [Alterra](http://www.alterra.wur.nl/UK/) | No |
| [Peter Schade](https://publicwiki.deltares.nl/display/~peter_schade) | OA and [Bundesanstalt fuer Wasserbau](http://www.baw.de/vip/abteilungen/wbk) | No |
| Jon Goodall | OA and [University of South Carolina](http://www.sc.edu/) |  |
| Andrea Antonello | OA and Universita` di Trento/ HydroloGIS |  |
| [Jan Gregersen](https://publicwiki.deltares.nl/display/~gregersen%40lictek.dk) | OA and [HydroInform](http://www.hydroinform.com/) | No |
| Simon Cox | OA and CSIRO | Yes |
| Paul Cleverley  | OA and HR Wallingford  | Yes |
| Robert Szczepanek | OA and Cracow University of Technology / NT |  |
| Roger Moore | OA | No |
| David Fortune | OA and XPSolutions | No |
| Quillon Harpham | OA and HR Wallingford | Yes |
| Robert Millington | OA and Innovyze | No |
| David Lemon | OA and CSIRO | Yes |
| Andrew Hughes | OA and NERC/BGS | Yes |
| Bert Jagers | OA and Deltares | Yes |
| Michiel Blind | OA and Deltares | Yes |
|  |  |  |

# Scope

## Background

The Open Modelling Interface standard (more usually referred to as the OpenMI) makes possible run time data exchange between independently developed modelling components. OpenMI is an enabling technology which 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 HRWallingford (UK), the last with the assistance of its former subsidiary Wallingford Software (now Innovyse (USA). Future development and the OpenMI’s publication as an Open Geospatial Consortium international standard are now the responsibility of the OpenMI Association, which has taken ownership of the IPR of the OpenMI.

## 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 functions, models, databases, visualizations, analytical tools and measurement devices, to a full complex 3D time variant modelling application. In more practical terms, components can be anything necessary to build simulation models or decision support systems (DSS) enabling policy-makers to improve the outcomes of their work, scientists to understand or predict the earth as a system, designers to develop and test new products and academics to teach and research.

## What is it?

The OpenMI is an interface standard and consists of a core group of requirements and a set of 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, iteration, extrapolation, saving state and unit transformations.

## What does it do?

The key feature of the standard is that it enables the creation of links between components, where a link matches a variable in one with its equivalent in another. Related to the links are the GetValues and SetValues calls. These enable components to obtain (‘get’)[[1]](#footnote-1) the values of a variable from one component or change (‘set’) them in another (at a 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 mismatches in model temporal and spatial resolutions and representations (e.g. vector/raster/non-spatial) - see Figure 1.

More details are available in [What’s New in OpenMI 2.0](https://4310b1a9-a-cb397f23-s-sites.googlegroups.com/a/openmi.org/home/learning-more/WhatsNewinOpenMI2.pdf?attachauth=ANoY7cqARmpXj5Ny5GDXAf3MMiXxEyYmB-y6n5-wGQNJ9JlklKhwVkzpbubP3v149uQVw3Fuygn6FRctCURUmi2VDgiO9MONDCjiN4xYW_HqA0WJbztq_ZtasYh6EGhL-_bnhZdoivmwW4HJr9lcYbUZN2xwaHm9vBSFSZzLdT45HtXwhQtksalk7cnu9KtiOlWGqJblYDq8Sy8GGCue96Sz4jrX51SACL5chRTd4BKZadInbnfyHUQ%3D&attredirects=0).



Figure 1 Linkages between components and the use of adaptors

## How is the OpenMI implemented?

The process begins by identifying the variables[[2]](#footnote-2) whose values are to be obtained or ‘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 will also need to be made as to which of the locations are to be able exchange (accept or provide) 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, 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. If the component does not exist, then it should be designed to have that structure. In nearly all cases, this will be found to improve the components design and lead to more understandable and maintainable code.

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 from the OpenMI Association web site and use the third party tools which simplify this task. These are available from the web at no charge under an open source license. The other is to write your own code following the specification and the guidance in the user documentation available on the OpenMI Association web site.

Third party open source tools are also available for building and running compositions, i.e. a set of linked modelling components. The OA undertakes, subject to resources, to make a set of aids available should there be no third party software.

## 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. They are all available through the OpenMI Association website at [www.openmi.org](http://www.openmi.org) .

## Document structure

The document that follows defines the Open Modelling Interface. Section 2 sets out the rules for conformance. Sections 3, 4 and 5 cover normative references[[3]](#footnote-3) , terms and definitions and conventions. Section 6 specifies in detail the classes for creating an Open Modelling Interface.

The creation of the components together with the entities used is described in Sections 6.1, and 6.2. The spatial and temporal reference of the variables exchanged need defining and this is described in Section 6.3 and 6.4. The arguments passed between the components are defined (see Section 6.5) as well as the state of component itself (See Section 6.6). The exchange items themselves are described (see Section 6.7). The implementation of adaptors can be found in Section 6.8. The definition of the linkable component itself can be found in Section 6.9. Testing is described in the Appendices.

# Conformance

To conform to this standard 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 and its specializations (e.g. ITimeSpaceComponentfor time and space dependent components). These interfaces are described in the Section , 'OpenMI Requirements classes'.

The OpenMI compliance definition is stated in the comments associated with the source code of the IBaseLinkableComponent. The requirements for compliance are as follows:

* An OpenMI-compliant component ***shall*** implement the IBaseLinkableComponent interface according to specifications provided in this document and as comments in the OpenMI.Standard2 source code.
* An OpenMI compliant component can also comply to one or more extensions, by implementing the IBaseLinkableComponent interface and the extension interfaces with which it wishes to comply to, according to the specifications provided in this document and as comments in the OpenMI.Standard2 source code.
* An OpenMI-compliant component including its extensions ***shall***, when compiled, reference the OpenMI.Standard2\*.dlls/jars, which are compiled and released by the OpenMI Association.
* An OpenMI-compliant component ***shall*** be associated with an XML file, referred to as the OMI file, which conforms to (can be validated with) the LinkableComponent.xsd schema.
* An OpenMI-compliant component ***shall*** be associated with an XML file, referred to as the compliancy info file, which conforms to (can be validated with) the OpenMICompliancyInfo.xsd schema. This file shall be submitted to the OpenMI Association.
* The OpenMI Association provides two additional 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 property ***shall*** be implemented according to the instructions given in this document and in the OpenMI.Standard2 source code.
* The OpenMI Association’s downloadable standard zip file provides the only recognized version of source files, XML schemas and assembly files.

The interfaces are available in C# and Java. The C# or Java compiler will both ensure that the client code correctly calls the methods, and will both 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.

# 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.

OGC 04-046r3, The OpenGIS Abstract Specification, Topic 2: Spatial Referencing by Coordinates, August 2004

Documents associated with Unified Modeling Language™ (UML®), August 2011. http://www.omg.org/spec/UML/2.4.1/

Extensible Markup Language (XML) 1.0 (Fifth Edition), 26 November 2008. http://www.w3.org/TR/xml/

# 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 or interpolate between 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*.

Composition

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

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*.

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.

Model

A *model component* which has read in the data that describes the situation to be simulated, analysed, visualised or otherwise processed e.g. a generic *model component* for simulating the flow of water down an open channel which 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 computer.

Model component

A distinct part of a *model application* where computation takes place - where the ‘computation’ might be simulating a process, analysing or visualising 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 November 17, 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 November 17, 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](http://www.itu.int/dms_pubrec/itu-r/rec/tf/R-REC-TF.457-2-199710-I%21%21PDF-E.pdf) )

# Conventions

## Symbols (and abbreviated terms)

**1D** One Dimensional

**2D** Two Dimensional

**3D** Three Dimensional

**API** Application Programming Interface

**GUI** Graphical user interface

**OpenMI** Open Modelling Interface

**SDK** Software development kit

**UML** Unified Modelling Language

**UTC** Coordinated Universal Time (for most purposes this is the same as Greenwich Mean Time (GMT))

**XML** Extensible Markup Language

**XSD** XML Schema Definition

## 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.

## Extensible Markup Language (XML)

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

# OpenMI Requirements classes

In the following text the requirement class descriptions are grouped according to the aspect of the OpenMI to which they relate. Each description begins with a general **introduction** followed by a UML diagram presenting the dependencies between the classes. If the classes define an interface, the methods and the input and output parameters for each method are then described.

An OpenMI-compliant component including its extensions shall, when compiled, reference the OpenMI.Standard2\*.dlls/jars, which are compiled and released by the OpenMI Association. The OpenMI Association’s downloadable standard zip file provides the only recognized version of source files, XML schemas and assembly files.

## 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 A graphical view of the OMI file structure

|  |
| --- |
| **Requirements Class 1** |
| /req/componentinstantiation |
| **Target type** | OpenMI Component |

|  |
| --- |
| **Requirement 1.1** |
| /req/componentinstantiation/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. |

## Described and Identifiable Entries

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.

|  |
| --- |
| **Requirements Class 2** |
| req/DescribableIDentifiable |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 2.1** | /req/DescribableIDentifiable/IDescribable |
| **Requirement 2.2** | /req/DescribableIDentifiable/IIdentifiable |

To ensure consistency in the identification and description of all kinds of entity instances, two interfaces are provided: IDescribable and, derived from that, IIdentifiable. The majority of the OpenMI Standard Version2.0 interfaces are derived from one of these interfaces.



Figure 3 UML for IDescribable and IIdentifiable

Table  Operations for IDescribable

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Caption()** stringPublic | Sets and returns the Caption string (not to be used as an Id). |  |
| **Description()** stringPublic | Sets and returns additional descriptive information about the entity. |  |

Table Operations for IIdentifiable

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Id()** stringPublic | Returns the Id 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 ILinkableComponent, but a similar Id might be used by an exchange item of another ILinkableComponent. |  |

|  |
| --- |
| **Requirement 2.1** |
| /req/DescribableIDentifiable/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** |
| /req/DescribableIDentifiable/IIdentifiable |
| An OpenMI class that represents an identifiable entity ***shall*** implement the IIdentifiable interface based on the definition in Figure 2 and Table 2 |

## Value Definition

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** |
| /req/ValueDefinition |
| **Target type** | OpenMI component |
| **Dependency**  | [/req/DescribableIDentifiable/IDescribable](http://www.opengis.net/spec/OpenMI/2.0/req/IDescribable) |
| **Requirement 3.1** | [/req/valuedefinition/IValueDefinition](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/IValueDefinition) |
| **Requirement 3.2** | [/req/valuedefinition/Iunit](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/IUnit) |
| **Requirement 3.3** | [/req/valuedefinition/Iquantity](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/IQuantity) |
| **Requirement 3.4** | [/req/valuedefinition/Iquality](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/IQuality) |
| **Requirement 3.5** | [/req/valuedefinition/Icategory](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/ICategory/implements) |
| **Requirement 3.6** | [/req/valuedefinition/IDimension](http://www.opengis.net/spec/OpenMI/2.0/req/valuedefinition/IDimension) |



Figure UML diagram for Value Definition

IValueDefinition defines the interface for establishing 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 property 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.

One of the inheritors of the IValueDefinition is the IQuantity. This interface applies to exchange items whose values are numerical. It 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.GetConversionFactorToSI() \* v + Unit.GetOffsetToSI()

To enable physical dimension[[4]](#footnote-4) checks between quantities, the IDimension interface has been defined. A dimension is expressed as a combination of the base dimensions, derived from the SI system[[5]](#footnote-5), with a minor extension for currencies.

Table 3 illustrates the base quantities and the associated SI units. The IUnit interface provides a method to obtain the power for each dimension base of a specific exchange item For example, a discharge expressed in units of m3/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 Base units and dimension base in the OpenMI (derived from SI)

|  |  |  |
| --- | --- | --- |
| **Dimension base** | **SI base unit** | **symbol used** |
| Length | meter | m |
| Mass | kilogram | kg |
| Time | second | s |
| ElectricCurrent | ampere | A |
| Temperature | kelvin | K |
| AmountOfSubstance | mole | mol |
| LuminousIntensity | candela | cd |
| Currency[[6]](#footnote-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 is the IQuality interface and it applies to exchange items whose values are qualitative. The IQuality 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 IsOrdered property indicates whether an ordering can be recognized in the categories or not. If IsOrdered is True, one category represents a ‘higher’ value than another category. The sequence ‘very light’, ‘light’, ‘heavy’, ‘very heavy’ would be an example of an ordered categorization.

Table Operations of IValueDefinition

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **MissingDataValue()** ObjectPublic | Returns the value representing data that are missing |  |
| **ValueType()** TypePublic | Returns the object types of value that will be available in the "IBaseValueSet" that is returned by the Values property and the GetValues function of the "IBaseExchangeItem”. |  |

Table Operations of IUnit

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ConversionFactorToSI()** doublePublic | Returns the conversion factor to SI units 'A' in the expression: SI-value = **A** \* quantity-value + B |  |
| **Dimension()** IDimensionPublic | Returns the unit's dimensions. |  |
| **OffSetToSI()** doublePublic | Returns the offset to SI units 'B' in the expression: SI-value = A \* quantity-value + **B** |  |

Table Operations for IQuantity

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Unit()** IUnitPublic | Returns the unit of measurement in which the quantity’s values are expressed. |  |

Table Operations of IQuality

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Categories()** IList<ICategory>Public | Returns a list of the possible "ICategory" allowed for this IQuality. If the categories are not ordered, the list contains the ICategory's in an unspecified order. When it is ordered the list contains the ICategory's in sequence. |  |
| **IsOrdered()** boolPublic | Returns a boolean indicating whether or not the IQuality is defined by an ordered set of ICategory or not. |  |

Table Operations of ICategory

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Value()** objectPublic | Returns the value for this category.For example "blue" from the allowed set of values "red", "green" and "blue". |  |

Table Operations of IDimension

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **GetPower()** doublePublic | Returns the power for the requested dimension.For example, for a quantity such as flow, which may have the unit m3/s, the GetPower method must work as follows:myDimension.GetPower(DimensionBase.AmountOfSubstance) --> returns 0myDimension.GetPower(DimensionBase.Currency) --> returns 0myDimension.GetPower(DimensionBase.ElectricCurrent) --> returns 0myDimension.GetPower(DimensionBase.Length) --> returns 3myDimension.GetPower(DimensionBase.LuminousIntensity) --> returns 0myDimension.GetPower(DimensionBase.Mass) --> returns 0myDimension.GetPower(DimensionBase.Temperature) --> returns 0myDimension.GetPower(DimensionBase.Time) --> returns -1 | DimensionBase[in] dimensionBase |

Table Attributes of DimensionBase

| **Attribute** | **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. |

|  |
| --- |
| **Requirement 3.1** |
| /req/valuedefinition/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** |
| /req/valuedefinition/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** |
| /req/valuedefinition/Iquantity |
| An OpenMI class that represents a unit of a quantity ***shall*** implement the IUnit interface based on the definition in Figure 4 and Table 6. |

|  |
| --- |
| **Requirement 3.4** |
| /req/valuedefinition/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** |
| /req/valuedefinition/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** |
| /req/valuedefinition/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 base dimensions derived from the SI system and defined in the DimensionBase enumeration. |

## Spatial Definition

Data exchange between components in the OpenMI is nearly always related to one or more of a set of elements in a space which 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. 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.

|  |
| --- |
| **Requirements Class 4** |
| req/spatialdefinition |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 4.1** | /req/spatialdefinition/ispatialdefinition |
| **Requirement 4.2** | /req/spatialdefinition/ielementset |
| **Requirement 4.3** | /req/spatialdefinition/ElementType |

The ISpatialDefinition is the general spatial construct of which all other spatial constructions are extensions. 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.

Note that 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 (e.g. 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]](#footnote-7). The IElementSet interface has been defined to describe, in a finite element sense, the location where each exchange item value applies. Note that to correctly interpret the co-ordinates, it will be necessary to obtain the spatial reference system to which they relate from the ISpatialDefinition.

Note that IElementSet 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 therefore are 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 which 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 (elementIndex, faceIndex and 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 UML for Spatial Definition

Table Dependencies for ISpatialDefinition

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| GeneralizationSource -> Destination  | PublicIElementSet  | PublicISpatialDefinition  |   |
| GeneralizationSource -> Destination  | PublicISpatialDefinition  | PublicIDescribable  |   |

Table 12 Operations of ISpatialDefinition

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ElementCount()** intPublic | Returns the number of data elements on the spatial axis, e.g. the number of points, the number of polygons, etc. |  |
| **SpatialReferenceSystemWkt()** stringPublic | Returns the SpatialReferenceSystemWkt which specifies the OGC Well Known Text representation of the spatial reference system to be used in association with the coordinates in the "ISpatialDefinition".For the list of WKT strings see: " <http://spatialreference.org/> ". |  |
| **Version()** intPublic | Returns the current version number for the spatial axis.The version must be incremented if any of the spatial data are changed. |  |

Table Operations of IElementSet

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ElementType()** ElementTypePublic | Returns the "ElementType" of the elementset. All elements in the set are of his type. |  |
| **GetElementId()** IIdentifiablePublic | Gets the Id of the ' "index" -th' element in the ElementSet. Indexes start from zero. If the ElementType of the ElementSet is not IdBased, a null or an empty string may be returned.Returns the index of the element with the specified Id or -1 if the Id was not found. | int [in] indexThe element index for which the element Caption is requested. If the element index is outside the range [0, number of elements -1], an exception must be thrown. |
| **GetElementIndex()** intPublic | Gets the index of element with Id "elementId" in the elementset. Indexes start from zero. There are no restrictions on how elements are ordered. | IIdentifiable [in] elementIdIdentification string for the element for which the element index is requested. If no element in the ElementSet has the specified elementId, -1 must be returned. |
| **GetFaceCount()** intPublic | Gets the number of faces in a 3D element. For 2D elements this returns 0.Returns the number of faces. | int [in] elementIndexIndex for the elementIf the element index is outside the range [0, number of elements -1], an exception must be thrown. |
| **GetFaceVertexIndices()** intPublic | Gets the array of the vertex indices for a faceRemark: the vertex indices for a face must be locally numbered for the element (containing numbers in the range [0;"GetVertexCount" (elementIndex)-1]). | int [in] elementIndexElement index.int [in] faceIndexFace index. |
| **GetVertexCount()** intPublic | Gets the number of vertices for the element specified by the elementIndex. If the GetVertexCount() method is invoked for element sets of type TimeSpace.ElementType.IdBased, an exception must be thrown.Gets the number of vertices in element defined by the elementIndex. | int [in] elementIndexThe element index for the element for which the number of vertices is requested. If the element index is outside the range [0, number of elements -1], an exception must be thrown. |
| **GetVertexMCoordinate()** doublePublic | Gets the M co-ordinate for the vertex with VertexIndex of the element with elementIndex. | int [in] elementIndexElement index.int [in] vertexIndexVertex index in the element with index elementIndex. |
| **GetVertexXCoordinate()** doublePublic | Gets the X co-ordinate for the vertex with vertexIndex of the element with elementIndex. | int [in] elementIndexElement index.int [in] vertexIndexVertex index in the element with index elementIndex. |
| **GetVertexYCoordinate()** doublePublic | Gets the Y co-ordinate for the vertex with vertexIndex of the element with elementIndex. | int [in] elementIndexElement index.int [in] vertexIndexVertex index in the element with index elementIndex. |
| **GetVertexZCoordinate()** doublePublic | Gets the Z co-ordinate for the vertex with vertexIndex of the element with elementIndex. | int [in] elementIndexElement index.int [in] vertexIndexVertex index in the element with index elementIndex. |
| **HasM()** boolPublic | Returns a boolean which is true if the element set contains M co-ordinates. |  |
| **HasZ()** boolPublic | Gets a boolean which is true if the element set contains Z co-ordinates. |  |

Table Enumerator ElementType

| **ElementType** | **Convention** |
| --- | --- |
| IDBased | ID-based (string comparison). |
| Point | Geo-referenced point in the horizontal (XY)-plane or in the 3-dimensional (XYZ)-space. |
| PolyLine | Geo-referenced polyline connecting at least two vertices in the horizontal (XY)-plane or in the 3-dimensional (XYZ)-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 (XY)-plane or in the 3-dimensional (XYZ)-space. Vertices defined anti-clockwise. Closed entity with one face, begin- and end-vertices being identical. |
| Polyhedron | Geo-referenced polyhedra in 3-dimensional (XYZ)-space. Vertices defined anti-clockwise for each face. Closed entity with many faces, begin- and end-vertices being identical |

|  |
| --- |
| **Requirement 4.1** |
| /req/spatialdefinition/ispatialdefinition |
| An OpenMI component ***shall*** implement the ISpatialDefinition interface based on the definition in Figure 5 and Table 12. |

|  |
| --- |
| **Requirement 4.2** |
| /req/spatialdefinition/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** |
| /req/spatialdefinition/ElementType |
| An OpenMI component ***shall*** implement the enumeration for the known element types as specified in Table 14. |

## Temporal Definition

Time in the OpenMI is defined by the ITimeSet interface. 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** |
| /req/temporaldefinition |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 5.1** | /req/temporaldefinition/itime |
| **Requirement 5.2** | /req/temporaldefinition/itimeset |



Figure UML Diagram for Temporal Definition

Table Operations of ITimeSet

| **Method** | **Notes** |
| --- | --- |
| **HasDurations()** boolPublic | 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()** doublePublic | 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()** ITimePublic | 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 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 yetIf for an input item TimeSet.Times.Count == 0 , the input item is time dependent item, but currently there are no values required yet. |

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

The value of HasDurations specifies whether the time set’s times are time stamps (in case of False) or time spans (in case of True).

The TimeHorizon property provides information on the timeframe during which an exchange item will interact with other exchange items. For an input item, the property 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 horizons begin time, StampAsModifiedJulianDay. Also, it will never go further ahead than the time horizons 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.

|  |
| --- |
| **Requirement 5.1** |
| /req/temporaldefinition/itime |
| An OpenMI component supporting time handling ***shall*** implement the ITime interface based on the definition in Figure 6. |

|  |
| --- |
| **Requirement 5.2** |
| /req/temporaldefinition/itimeset |
| An OpenMI component supporting time handling ***shall*** implement the ITimeSet interface based on the definition in Figure 6 and Table 15. |

## Value Sets

To enable massive data exchange over a link, an interface structure has been defined that supports the exchange of multi-dimensional data. The base interface for this, the IBaseValueSet represents an ordered N-dimensional list of values. Time and space dependent computational cores often compute per time step, therefore the time and space extension OpenMI.Standard2.TimeSpace contains a more specialized version of the IBaseValueSet, the ITimeSpaceValueSet. This interface represents an ordered two-dimensional list of values. The first dimension stands for the times for which values are available, whereas in the second dimension each value belongs to precisely one element in the corresponding element set (which was specified when asking 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 element set.

|  |
| --- |
| **Requirements Class 6** |
| /req/valueset |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 6.1** | /req/valueset/ibasevalueset |
| **Requirement 6.2** | /req/valueset/itimespacevalueset |

****

Figure UML Diagram for Value Set

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)

where the exchange item value indicates the movement of something from the “source/ providing” component to the “target/requesting/accepting” component.

To prevent misunderstanding of positive and negative values, the following conventions are applied:

* Values are positive if the matter leaves the source component and enters the target component (this also is the case for volumes).
* The 'right-hand rule' applies for fluxes through a plane or polygon[[8]](#footnote-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]](#footnote-9).



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

Software developers that do not comply with those conventions should make software users aware of this risk.

Table Operation for IbaseValue Sets

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **GetIndexCount()** intPublic | Gets the length (max index count) of the dimension specified by the given indices. To get 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"Returns the length of the specified dimension. | int[] [in] indicesIndices specifying the dimension whose length is to be obtained |
| **GetValue()** ObjectPublic | Gets 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.Returns the value object for the given indices. | int[] [in] indicesIndex value for each dimension |
| **NumberOfIndices()** intPublic | Gets the number of possible indices (dimensions) for the value set.Returns the number of indices. |  |
| **SetValue()** voidPublic | 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] indicesValue for each dimensionObject [in] valueThe value object for the given indices |
| **ValueType()** TypePublic | Returns the object type of the values that will be available in the value set that is returned by the Values property and the GetValues function. |  |

Table Operations of ITimespaceValueSets

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **GetElementValuesForTime()** IListPublic | Gets values from "Values2D", for all elements, for the specified “timeIndex". If the data are time independent, timeIndex must be specified as 0. | int [in] timeIndex  |
| **GetTimeSeriesValuesForElement()** IListPublic | Gets values from "Values2D", 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()** objectPublic | Gets 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] timeIndexint [in] elementIndex  |
| **SetElementValuesForTime()** voidPublic | Sets values in "Values2D", for all elements, for the specified “timeIndex". If the data are time independent, timeIndex must be specified as 0. | int [in] timeIndexIList [in] values  |
| **SetTimeSeriesValuesForElement()** voidPublic | Sets values in "Values2D", for all times, for the specified “elementIndex". If the data are not related to a location, elementIndex must be specified as 0. | int [in] elementIndexIList [in] values  |
| **SetValue()** voidPublic | Sets the value in "Values2D”, 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] timeIndexint [in] elementIndexobject [in] value  |
| **Values2D()** IList<IList>Public | Two-dimensional list of values. The first IList represents time, and the contained Ilist represents the elements in the IElementSet. |  |

|  |
| --- |
| **Requirement 6.1** |
| /req/valueset/ibasevalueset |
| An OpenMI component ***shall*** implement the IBaseValueSet interface based on the definition in Figure 7 and Table 16. |

|  |
| --- |
| **Requirement 6.2** |
| /req/valueset/itimespacevalueset |
| An OpenMI component exchanging time based data ***shall*** implement the ITimeSpaceValueSet interface based on the definition in Figure 7 and Table 17. |

## Argument

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 done by means of the IArgument interface.

|  |
| --- |
| **Requirements Class 7** |
| /req/argument |
| **Target type** | OpenMI component |
| **Dependency**  | /req/DescribableIDentifiable/IIdentifiable |
| **Requirement 7.1** | /req/argument/iargument |



Figure UML Diagram for Argument

Table Operations of Iargument

| **Method** | **Notes** |
| --- | --- |
| **DefaultValue()** objectPublic | Returns the default value of the argument. |
| **IsOptional()** boolPublic | Specifies whether the argument is optional or not. If the Values property returns null and IsOptional == false, a value has to be set before the argument can be used. |
| **IsReadOnly()** boolPublic | Defines whether the Values property 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()** objectPublic | 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()** stringPublic | 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 function offers the possibility to treat it as a string, e.g. to let the GUI persist the value in the composition file. |
| **ValueType()** TypePublic | Returns the type of the value of the argument, e.g. a integral type like string, integer or double, or a non-integral type, such as a time series object. |

|  |
| --- |
| **Requirement 7.1** |
| /req/argument/iargument |
| An OpenMI component exchanging time based data ***shall*** implement the IArgument interface based on the definition in Figure 9 and Table 18. |

## Component status

The OpenMI explicitly specifies the possible states in which a linkable component can be found.

The status of a component is notified to the outside world by raising a change of status event whenever the component’s state or exchange item values alter.

|  |
| --- |
| **Requirements Class 8** |
| /req/componentstatus |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 8.1** | /req/componentstatus/statusenum |
| **Requirement 8.2** | /req/ componentstatus/statusphases |
| **Requirement 8.3** | /req/componentstatus/eventstatuschanged |
| **Requirement 8.4** | /req/componentstatus/eventexchangeitemchanged |

The linkable component status has two purposes; an informative purpose (what is the component currently doing) and a control flow decision purpose. The possible states in which a component can exist, and the condition and sequence in which they can arise, are illustrated in Figure 10 and listed in Table 19.

Two types of events are distinguished: the ‘component status changed’ event and the ‘exchange item value changed’ event.



Figure Component Status Change Diagram

****

Figure UML Diagram for Component Status

Table Enumeration ComponentStatus

| **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 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 its required input will be available, and it it will 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 ILinkableComponent was requested to perform the actions to be performed before it will either be disposed or re-intialized 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. |
| **Finished**Public | The ILinkableComponent 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. |

### Events

When a component moves from one status to another, the component must raise an ILinkableComponentStatusChanged event. Similarly, when, during computation, the value of an exchange item alters, the component must raise an IBaseExchangeItemChanged event.

Sending exchange item events is optional, so it should not be used as a mechanism upon which to build critical functionality.

The LinkableComponentStatusChangeEventArgs contains the information that will be passed when the "IBaseLinkableComponent" raises a LinkableComponentStatusChange event.

Table Operations for ExchangeItemChangeEventArgs

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ExchangeItem()** IBaseExchangeItemPublic | Returns and sets the exchange item whose the 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" property. | IBaseExchangeItem [in] exchangeItemstring [in] message  |
| **Message()** stringPublic | Returns and sets a message that describes the way in which the status of the exchange item has been changed. |  |

The ExchangeItemChangeEventArgs contains the information that will be passed when the"IBaseExchangeItem" raises the ExchangeItemValueChanged event. Sending exchange item events is optional, so they should not be used as a mechanism upon which to build critical functionality.

Table 21 Operations of LinkableComponentStatusChangeEventArgs

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **LinkableComponent()** IBaseLinkableComponentPublic | Returns and sets the linkable component that raised the status change event. |  |
| **LinkableComponentStatusChangeEventArgs()**Public | Constructor. |  |
| **Message()** stringPublic | Returns and sets the message providing additional information on the status change. If there is no message, an empty string is returned. |  |
| **NewStatus()** LinkableComponentStatusPublic | Returns and sets the linkable component's status after the status changes. |  |
| **OldStatus()** LinkableComponentStatusPublic | Returns and sets the linkable component's status before the status change. |  |

|  |
| --- |
| **Requirement 8.1** |
| /req/componentstatus/statusenum |
| An OpenMI component ***shall*** provide its status using one of the status conditions enumerated in Figure 10 and Table 19. |

|  |
| --- |
| **Requirement 8.2** |
| /req/ componentstatus/statusphases |
| An OpenMI component ***shall*** change the status according to the rules given in Figure 10 and Table 19. |

|  |
| --- |
| **Requirement 8.3** |
| /req/componentstatus/eventstatuschanged |
| If an OpenMI component implements a status change event handling mechanism, then, when moving from one status to another, the component ***shall*** raise a LinkableComponentStatusChange. Event pass information in parameter of the type LinkableComponentStatusChangeEventArgs based on that presented in Figure 11 and Table 20. |

|  |
| --- |
| **Requirement 8.4** |
| /req/componentstatus/eventexchangeitemchanged |
| If an OpenMI component implements event handling mechanism andif during computation the content of an exchange item changes, the component ***shall*** raise an ExchangeItemChange event. Event pass information in parameter of the type ExchangeItemChangeEventArgs based on that presented in Figure 11 and Table 21. |

## Exchange item

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 is information is obtained through the IBaseExchangeItem interface and the various interfaces that derived from it.

For a linkable component, an exchange item either is an input item (IBaseInput) or an output item (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, the output item’s GetValues() method is invoked with a query specification. 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 visualisation.

|  |
| --- |
| **Requirements Class 9** |
| /req/exchangeitem |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 9.1** | /req/exchangeitem/ibaseexchangeitem |
| **Requirement 9.2** | /req/exchangeitem/ibaseinput |
| **Requirement 9.3** | /req/exchangeitem/ibaseoutput |
| **Requirement 9.4** | /req/exchangeitem/itimespaceexchangeitem |
| **Requirement 9.5** | /req/exchangeitem/itimespaceoutput |
| **Requirement 9.6** | /req/exchangeitem/itimespaceinput |



Figure UML Diagram for Exchange Item

Table Operation for IBaseExchangeItem

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Component()** IBaseLinkableComponentPublic | 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()** IValueDefinitionPublic | 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. |  |

### ITimeSpaceExchangeItem

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

Table Operations for ITimeSpaceExchangeItem

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **SpatialDefinition()** ISpatialDefinitionPublic | Spatial information (usually an element set) about the values that are available in an output exchange item or required by an input exchange item. |  |
| **TimeSet()** ITimeSetPublic | Temporal information about the values that are available in an output exchange item or required by an input exchange item. |  |

### IBaseInput

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

Table Operations for IBaseInput

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Provider()** IBaseOutputPublic | Returns and sets the provider of this IBaseInput |  |
| **Values()** IBaseValueSetPublic | The exchange item's values. |  |

### ITimeSpaceInput

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 Operations for ItimeSpaceInput

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **Values()** ITimeSpaceValueSetPublic | Sets the exchange item's values, as a specialized "ITimeSpaceValueSet"  |  |

### IBaseOutput

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 Operations for IBasedOutput

| Method | 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 readonly. Add and remove from the list by using "AddAdaptedOutput” and "RemoveAdaptedOutput”. |  |
| **AddAdaptedOutput()** voidPublic | Adds an “IBaseAdaptedOutput" as adaptee to the current output item. Every adapted output that uses data from this output item 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."adaptedOutput” consumer that has to be added  | IBaseAdaptedOutput [in] adaptedOutput |
| **AddConsumer()** voidPublic | 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 GetValues, it should remove itself by calling the RemoveConsumer method.The list is readonly. Add and remove from the list by using "AddConsumer” and "RemoveConsumer".Remark: please be aware that the "raw" values in the output item, provided by the read only Values property, may be called anyway, even if there are no values available. |  |
| **GetValues()** IBaseValueSetPublic | 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 Iinput, 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()** voidPublic | 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] adaptedOutputAdaptee that has to be removed. |
| **RemoveConsumer()** voidPublic | Removes a consumer. If an input item is no longer interested in calling the GetValues method, it should remove itself by calling RemoveConsumer.Consumer that has to be removed  | IBaseInput [in] consumerConsumer that has to be removed. |
| **Values()** IBaseValueSetPublic | The exchange item's values. |  |

### ITimeSpaceOutput

An output exchange item that can deliver values from a time / space dependent ILinkableComponent. The output is a combination of an "IValueDefinition”, an "IElementSet” and an "ITimeSet". This combination specifies which type of data can be provided where and when by the ILinkableComponent.

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, unit transformations, etc.).

Table Operations for ITimeSpaceOutput

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

|  |
| --- |
| **Requirement 9.1** |
| /req/exchangeitem/ibaseexchangeitem |
| An OpenMI component ***shall*** implement the IBaseExchangeItem interface based on the definition in Figure 12 and Table 22. |

|  |
| --- |
| **Requirement 9.2** |
| /req/exchangeitem/ibaseinput |
| If an OpenMI component needs to accept input it ***shall*** implement the IBaseInput interface based on the definition in Figure 12 and Table 24. |

|  |
| --- |
|  **Requirement 9.3** |
| /req/exchangeitem/ibaseoutput |
| If an OpenMI component needs to provide output it ***shall*** implement the IBaseOutput interface based on the definition in Figure 12 and Table 26. |

|  |
| --- |
| **Requirement 9.4** |
| /req/exchangeitem/itimespaceexchangeitem |
| A time dependent OpenMI component ***shall*** implement the ITimeSpaceExchangeItem interface based on the definition in Figure 12 and Table 23 |

|  |
| --- |
| **Requirement 9.5** |
| /req/exchangeitem/itimespaceoutput |
| If an OpenMI component provides time dependent output it ***shall*** implement the ITimeSpaceOutput interface based on the definition in Figure 12 and Table 27. |

|  |
| --- |
| **Requirement 9.6** |
| /req/exchangeitem/itimespaceinput |
| If an OpenMI component accepts time dependent inputs it ***shall*** implement the ITimeSpaceInput interface based on the definition in Figure 12 and Table 25.  |

## Adapted Outputs

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 section 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.

To create adapted outputs, the IAdaptedOutputFactory has been defined.

|  |
| --- |
| **Requirements Class 10** |
| /req/adaptedoutput |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 10.1** | /req/adaptedoutput /ibaseadaptedoutput |
| **Requirement 10.2** | /req/adaptedoutput /itimespaceadaptedoutput |
| **Requirement 10.3** | /req/adaptedoutput /iadaptedoutputfactory |



Figure UML Diagram for Adapted Output



Figure 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 Refesh() method of all the output item's AdaptedOutputs. IBaseAdaptedOutput instances are created by means of an "IAdaptedOutputFactory".

### IBaseAdaptedOutput

An "IBaseAdaptedOutput" adds one or more data operations on top of an output item. It is in itself an "IBaseOutput”. The AdaptedOutput 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 Operations for IBaseAdaptedOutput

|  |  |
| --- | --- |
| **Method** | **Notes** |
| **Adaptee()** IBaseOutputPublic | Output item that this adaptedOutput extracts content from. In the adapter design pattern, it 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()** voidPublic | 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()** voidPublic | 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. |

### ItimeSpaceAdaptedOutput

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 property return an ITimeSpaceValueSet instead of a IBaseValueSet.

### AdaptedOutputFactory

AdaptedOutputFactory 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 Operations for IAdaptedOutputFactory

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **CreateAdaptedOutput()** IBaseAdaptedOutputPublic | 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] adaptedOutputIdThe identifier of the adaptedOutput to create.IBaseOutput [in] adapteeThe "IBaseOutput"to adapt.IBaseInput [in] target"IBaseInput" to adapt the adaptee to, can be null . |
| **GetAvailableAdaptedOutputIds()** IIdentifiablePublic | 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"IBaseInput" to adapt the adaptee to, can be null . |

|  |
| --- |
| **Requirement 10.1** |
| /req/adaptedoutput /ibaseadaptedoutput |
| An OpenMI component which supports the adaptation of its output values ***shall*** implement the IBaseAdaptedOutput interface according to the definition in Figure 13 and Table 28.  |

|  |
| --- |
| **Requirement 10.2** |
| /req/adaptedoutput /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 13 and Table 28. |

|  |
| --- |
| **Requirement 10.3** |
| /req/adaptedoutput /iadaptedoutputfactory |
| To be able create instances of adaptors, an OpenMI component ***shall*** implement the IAdaptedOutputFactory interface based on the definition in Figure 14 and Table 29. |

## Manage State

An OpenMI linkable component may implement two optional interfaces. The "IManageState" interface handles feed back loops by storing and restoring states. The"IByteStateConverter" interface provides the ability to makes these states persistent.



Figure UML Diagram for Manage State

|  |
| --- |
| **Requirements Class 11** |
| /req/managestate |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 11.1** | /req/managestate/imanagestate |
| **Requirement 11.2** | /req/exchangeitem/ibytestateconverter |

### IManageState

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 Operations for IManageState

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ClearState()** voidPublic | 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] stateIdIdentifier of the state to be cleared. |
| **KeepCurrentState()** IIdentifiablePublic | Stores the linkable component's current State.Returns the identifier of the stored state. |  |
| **RestoreState()** voidPublic | 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] stateIdIdentifier of the state to be restored. |

### IByteStateConverter

This interface is an rextension of the IManageState interface mentioned above (both are extensions to IBaseLinkableComponent). 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 get or set a model's state as a simple bytestream that can be made persistent by writing it to file.

Table Operations for IByteStateConverter

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **ConvertFromByteArray()** IIdentifiablePublic | 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] byteArrayState as a byte stream. |
| **ConvertToByteArray()** bytePublic | Converts the state with the "stateId" into a byte stream.Returns the state identified by "stateId" as an array of bytes. | IIdentifiable [in] stateIdId of the state. |

|  |
| --- |
| **Requirement 11.1** |
| /req/managestate/imanagestate |
| If an OpenMI component supports the optional managing of state, it ***shall*** implement the IManageState interface based on the definition in Table 30 and Figure 15. |

|  |
| --- |
| **Requirement 11.2** |
| /req/exchangeitem/ibytestateconverter |
| If an OpenMI component supports the managing of state, it ***shall*** implement the IByteStateConverter interface based on the definition in Table 31 and Figure 15. |

## LinkableComponent

All interfaces mentioned above come together in the main interface of OpenMI, the basic interface for accessing a model component. This ILinkableComponent 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.

|  |
| --- |
| **Requirements Class 12** |
| /req/linkablecomponent |
| **Target type** | OpenMI component |
| **Dependency**  |  |
| **Requirement 12.1** | /req/linkablecomponent/ibaselinkablecomponent |
| **Requirement 12.2** | /req/linkablecomponent/itimespacecomponent |
| **Requirement 12.3** | /req/linkablecomponent/itimespace |

Since all access to a component is through this interface, generic OpenMI implementation environments (e.g. GUI's or operational systems) 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 ILinkableComponent interface. By having this one generic interface, the process of assembling components is greatly simplified. Further, component developers can make changes within their component without impacting the rest of the assembly – 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() function 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() function can be called repeatedly until the component reaches the end of its processing.

Quite often, the values produced by an output item are not the form in which 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 to one or more extensions, by implementing both the IBaseLinkableComponent interface and the extension interfaces to which it wishes to comply, e.g. the ITimeSpaceExtension.

Figure 16 UML Diagram for LinkableComponent

*IIdentifiable*

«interface»

**IBaseLinkableComponent**

+

Finish() : void

+

Initialize() : void

+

Prepare() : void

+

Update(IBaseOutput[]) : void

+

Validate() : string[]

«property»

+

AdaptedOutputFactories() : List<IAdaptedOutputFactory>

+

Arguments() : IList<IArgument>

+

Inputs() : IList<IBaseInput>

+

Outputs() : IList<IBaseOutput>

+

Status() : LinkableComponentStatus

«event»

+

StatusChanged() : EventHandler<LinkableComponentStatusChangeEventArgs>

«interface»

**ITimeSpaceExtension**

«property»

+

TimeExtent() : ITimeSet

*ITimeExtension*

«interface»

**ITimeSpaceComponent**

### ILinkableComponent

The IBaseLinkableComponent is the main interface in the OpenMI standard.

Table Operations for IBaseLinkableComponent

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **AdaptedOutputFactories()** List<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 | Arguments needed to let the component do its work. An un-modifiable 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 property 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()** voidPublic | This method is and must be invoked as the last of any methods in the ILinkableComponent 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()** voidPublic | Initializes the LinkableComponent. The "Initialize()" method must be invoked before any other method or property of the ILinkableComponent interface is invoked or accessed, except for the "Arguments" property. 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 properties Id, Caption, Description, "Inputs”, "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 with getArguments. Settings can be used to read input files, allocate memory, and organize input and output exchange items. |  |
| **Inputs()** IList<IBaseInput>Public | The list of input items for which a component can receive values.Remark: this property must be accessible after the "Initialize()" method has been invoked and until the "Validate" method has been invoked. If this property 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 "IBaseInputitems". 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. |  |
| **Outputs()** IList<IBaseOutput>Public | The list of output items for which a component can produce results.Remark: this property must be accessible after the "Initialize()" method has been invoked and until the "Validate" method has been invoked. If this property 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 of the component, not the IBaseAdaptedOutput derived from each IBaseOutput (etc.). To obtain a complete list of outputs, traverse the chain of IBaseAdaptedOutput that starts with the IOutputs returned 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()** voidPublic | Prepares the IBaseLinkableComponent for calls to the "Update" methodBefore 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 property 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 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()** LinkableComponentStatusPublic | Defines 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. In this Status, "Arguments" is the only property 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()** voidPublic | This method is called to let the component update itself, thus reaching its next state. Immediately after the method is invoked, it changes the linkable component's Status 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 output items can provide up-to-date values. If the Update method is performed successfully, the component sets its state to "LinkableComponentStatus.Updated”, unless after this Update action the component is at the end of its computation, in which case it will be set its State to "LinkableComponentStatus.Done". If during the Update method a problem arises, the component sets its state to"LinkableComponentStatus.Failed”, and throws an exception. | IBaseOutput[] [in] requiredOutputThis 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. |
| **Validate()** stringPublic | 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 property 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 present in 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 has changed 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 "LinkableComponentStatus.Invalid”, at least one of the messages indicates a fatal error. |  |

### ITimeSpaceComponent

ITimeSpaceComponent providing exchange items of type time-space.

### ITimeSpace

Methods that are specific for a time-space component.

Table Operation of ITimeSpace

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **TimeExtent()** ITimeSetPublic | The "TimeExtent" property describes in what time span the component can operate. This can be used to support the user when creating a composition. |  |

|  |
| --- |
| **Requirement 12.1** |
| /req/linkablecomponent/ibaselinkablecomponent |
| An OpenMI component ***shall*** implement the IBaseLinkableComponent interface based on the definition in Figure 16 and Table 32.  |

|  |
| --- |
| **Requirement 12.2** |
| /req/linkablecomponent/itimespacecomponent |
| Time dependent OpenMI component ***shall*** implement the ITimeSpaceLinkableComponent interface based on the definition in Figure 16 and Table 32. |

|  |
| --- |
| **Requirement 12.3** |
| /req/linkablecomponent/itimespace |
| Time dependent OpenMI component ***shall*** implement the ITimeSpace interface based on the definition in in Figure 16 and Table 33. |

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 (see the related sites for more detailed):

* 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 (OTC) [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** |
| /conf/Instantiation |
| **Requirements** | /req/componentinstantiation |
| **Test 1.1** | /conf/componentinstantiation/availableXML |
| Requirement  | [/req/componentinstantiation/validXML](#ValidXML) |
| Test purpose  | Check if an .OMI-file has been provided for the linkable component. |
| Test method  | Visual inspection if one or more .OMI files are present. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 2** |
| /conf/Describable |
| **Requirements** | req/DescribableIDentifiable |
| **Test 2.1** | /conf/DescribableIDentifiable /IDescribable |
| Requirement  | /req/DescribableIDentifiable/IDescribable |
| Test purpose  | Test if the OpenMI component implements the IDescribable interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 2.2** | /conf/describedandidentifiableentities/IIdentifiable |
| Requirement  | /req/DescribableIDentifiable/IIdentifiable |
| Test purpose  | Test if the OpenMI component implements the IIdenifiable interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 3** |
| /conf/ValueDefinition |
| **Requirements** | /req/ValueDefinition |
| **Test 3.1** | /conf/ValueDefinition/IValueDefinition |
| Requirement  | /req/valuedefinition/IValueDefinition |
| Test purpose  | Test if the OpenMI component implements the IValueDefinition interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |
| **Test 3.2** | /conf/ValueDefinition/IUnit |
| Requirement  | /req/valuedefinition/Iunit |
| Test purpose  | Test if the OpenMI component implements the IUnit interface |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |
| **Test 3.3** | /conf/ValueDefinition/IQuantity |
| Requirement  | /req/valuedefinition/IValueDefinition |
| Test purpose  | Test if the OpenMI component implements the IQuantity interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 3.4** | /conf/ValueDefinition/IQuality |
| Requirement  | /req/valuedefinition/Iquality |
| Test purpose  | Test if the OpenMI component implements the IQuality interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 3.5** | /conf/ValueDefinition/ICategory |
| Requirement  | /req/valuedefinition/Icategory |
| Test purpose  | Test if the OpenMI component implements the ICategory interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 3.6** | /conf/ValueDefinition/IDimension |
| Requirement  | /req/valuedefinition/IDimension |
| Test purpose  | Test if the OpenMI component implements the IDimension interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |

|  |
| --- |
| **Conformance Class 4** |
| /conf/spatialdefinition |
| **Requirements** | req/spatialdefinition |
| **Test 4.1** | /conf/spatialdefinition/ispatialdefinition |
| Requirement  | /req/spatialdefinition/ispatialdefinition |
| Test purpose  | Test if the OpenMI 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/spatialdefinition/ielementset |
| Requirement  | /req/spatialdefinition/ielementset |
| Test purpose  | Test if the OpenMI component implements the IElementSet interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 4.3** | /conf/enumeration/elementtype |
| Requirement  | /req/spatialdefinition/ElementType |
| Test purpose  | Test if the OpenMI component implements the element types enumeration. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 5** |
| /conf/temporaldefinition |
| **Requirements** | /req/temporaldefinition |
| **Test 5.1** | /conf/temporaldefinition/itime |
| Requirement  | /req/temporaldefinition/itime |
| Test purpose  | Test if the OpenMI component implements the itime interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 5.2** | /conf/temporaldefinition/itimeset |
| Requirement  | /req/temporaldefinition/itimeset |
| Test purpose  | Test if the OpenMI component implements the itimeset interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 6** |
| /conf/valueset |
| **Requirements** | /req/valueset |
| **Test 6.1** | /conf/valueset/ibasevalueset |
| Requirement  | /req/valueset/ibasevalueset |
| Test purpose  | Test if the OpenMI component implements the ibasevalueset interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 6.2** | /conf/valueset/itimespacevalueset |
| Requirement  | /req/valueset/itimespacevalueset |
| Test purpose  | Test if the OpenMI component implements the ItimeSpaceValueSet interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 7** |
| /conf/argument |
| **Requirements** | /req/argument |
| **Test 7.1** | /conf/argument/iargument |
| Requirement  | /req/argument/iargument |
| Test purpose  | Test if the OpenMI component implements the iargument interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 8** |
| /conf/ComponentStatus |
| **Requirements** | /req/componentstatus |
| **Test 8.1** | /conf/req/componentstatus/statusenum |
| Requirement  | /req/componentstatus/statusenum |
| Test purpose  | Test if the OpenMI component implements the statusenum interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 8.2** | /conf/req/componentstatus/statusenum |
| Requirement  | /req/componentstatus/statusphases |
| Test purpose  | Test if the OpenMI component implements the statusphases interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 8.3** | /conf/req/componentstatus/statusenum |
| Requirement  | /req/componentstatus/eventstatuschanged |
| Test purpose  | Test if the OpenMI component implements the eventstatuschanged interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 8.4** | /conf/req/componentstatus/statusenum |
| Requirement  | /req/componentstatus/eventexchangeitemchanged |
| Test purpose  | Test if the OpenMI component implements the eventexchangeitem interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Conformance Class 9** |
| /conf/ValueDefinition |
| **Requirements** | /req/exchangeitem |
| **Test 9.1** | /conf/exchangeitem/ibaseexchangeitem |
| Requirement  | /req/exchangeitem/ibaseexchangeitem |
| Test purpose  | Test if the OpenMI component implements the ibaseexchangeitem interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 9.2** | /conf/exchangeitem/ibaseinput |
| Requirement  | /req/exchangeitem/ibaseinput |
| Test purpose  | Test if the OpenMI component implements the ibaseinput interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 9.3** | /conf/exchangeitem/ibaseoutput |
| Requirement  | /req/exchangeitem/ibaseoutput |
| Test purpose  | Test if the OpenMI component implements the ibaseoutput interface |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 9.4** | /conf/exchangeitem/itimespaceexchangeitem |
| Requirement  | /req/exchangeitem/itimespaceexchangeitem |
| Test purpose  | Test if the OpenMI component implements the itimespaceexchangeitem interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |
| **Test 9.5** | /conf/exchangeitem/itimespaceoutput |
| Requirement  | /req/exchangeitem/itimespaceoutput |
| Test purpose  | Test if the OpenMI component implements the itimespaceoutput interface |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |
| **Test 9.6** | /conf/exchangeitem/itimespaceinput |
| Requirement  | /req/exchangeitem/itimespaceinput |
| Test purpose  | Test if the OpenMI component implements the itimespaceinput interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 10** |
| /conf/valueset |
| **Requirements** | /req/adaptedoutput |
| **Test 10.1** | /conf/adaptedoutput/ibaseadaptedoutput |
| Requirement  | /req/adaptedoutput /ibaseadaptedoutput |
| Test purpose  | Test if the OpenMI component implements the ibaseadaptedoutput interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 10.2** | /conf/valueset/itimespacevalueset |
| Requirement  | /req/adaptedoutput /itimespaceadaptedoutput |
| Test purpose  | Test if the OpenMI component implements the itimespaceadaptedoutput interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 10.3** | /conf/valueset/iadaptedoutputfactory |
| Requirement  | /req/adaptedoutput /iadaptedoutputfactory |
| Test purpose  | Test if the OpenMI component implements the iadaptedoutputfactory interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 11** |
| /conf/managestate |
| **Requirements** | /req/managestate |
| **Test 11.1** | /conf/managestate/imanagestate |
| Requirement  | /req/managestate/imanagestate |
| Test purpose  | Test if the OpenMI component implements the imanagestate interface |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 11.2** | /conf/managestate/ibyteconverter |
| Requirement  | /req/exchangeitem/ibytestateconverter |
| Test purpose  | Test if the OpenMI component implements the ibytestatecoverter interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |

|  |
| --- |
| **Conformance Class 12** |
| /conf/linkablecomponent |
| **Requirements** | /req/linkablecomponent |
| **Test 12.1** | /conf/linkablecomponent/ibaselinkablecomponent |
| Requirement  | /req/linkablecomponent/ibaselinkablecomponent |
| Test purpose  | Test if the OpenMI component implements the ibaselinkablecomponent interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | Basic |
| **Test 12.2** | /conf/timespacelinkablecomponent/itimespacecomponent |
| Requirement  | /req/linkablecomponent/itimespacecomponent |
| Test purpose  | Test if the OpenMI component implements the itimespacecomponent interface. |
| Test method  | Compile the code against the openmi/2.0/req interface binaries. |
| Test type  | basic |
| **Test 12.3** | /conf/timespacelinkablecomponent/itimespace |
| Requirement  | /req/linkablecomponent/itimespace |
| Test purpose  | Test if the OpenMI component implements the itimespace interface. |
| Test method  | 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" targetNamespace="http://www.openmi.org" elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.4.0.0">

 <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:choice>

 <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:choice>

 <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>

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. ‘Get’ in the context of the OpenMI covers the activities of ‘requesting’, ‘providing’ and ‘accepting’ the values of exchange items. [↑](#footnote-ref-1)
2. These are called ‘exchange items’. [↑](#footnote-ref-2)
3. Normative references are references to other standards upon which this standard builds. [↑](#footnote-ref-3)
4. A dimension describes the type of thing being measured, without specifying the magnitude. Thus the centimeter, kilometer, inch and foot all have dimensions of length. [↑](#footnote-ref-4)
5. More information on the SI system can be found at the National Institute of Standards and Technology (http://physics.nist.gov/cuu/Units/). [↑](#footnote-ref-5)
6. Currency has no base quantity in the SI system. Note that currency has conversion units that may vary over time. [↑](#footnote-ref-6)
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 is an extension of the ISpatialDefinition. [↑](#footnote-ref-7)
8. Curl your right hand in the vertex order of the plane or polygon. The thumb points in the positive direction [↑](#footnote-ref-8)
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. [↑](#footnote-ref-9)