De COINS-systematiek 1.0
Reference manual - July 2010

“De systematiek rondom gebruik bouwinformatie, opgesteld door de projectgroep COINS”
De COINS-systematiek 1.0

COINS is a standard for applying Building Information Models (BIM). This standard is used by partners in building construction projects for exchanging information and building information management.

The COINS standard contains the agreements of a general nature and validity. For specific applications, additional agreements in the form of COINS Reference frameworks are available. The COINS-systematiek 1.0 is established by the COINS project group in July 2010. Criticism, comments, discussion and questions regarding the COINS standard are welcome, please use our forum: http://www.coinsweb.nl/modules/newbb/. The COINS standard is published as an open standard.

Colofon

The COINS-systematiek 1.0 Reference manual, published – July 2010

The standard for applying Building Information Models (BIM), established by the COINS project group.

Authors:
P.H. Willems
H.A. Schaap

De COINS standard is developed with contributions from among others:

The COINS standard is published by the COINS consortium, represented by CUR Bouw & Infra, Gouda. Information: CUR Bouw & Infra, Tel +31 182 540 600, e-mail info@cur.nl

The COINS standard is een open standaard. The content of the standard is freely available. There are no restrictions on reuse of the standard.

DISCLAIMER: The publication has been compiled for use by persons with expertise in various disciplines presented. The use of the publication is at your own risk. CUR Bouw & Infra can in no way be held responsible for errors in the publication and / or errors resulting from the use of the publication.
Contents

Introduction .................................................................................................................. 7
Normative References ................................................................................................. 7
Terms and definitions ................................................................................................. 8
Schema architecture .................................................................................................... 9
  Flexible architecture ............................................................................................... 9
  Object libraries ....................................................................................................... 10
Special Topics ............................................................................................................ 11
Identification of CBIM information objects .............................................................. 11
  Contents .................................................................................................................. 11
  Introduction .......................................................................................................... 11
  GUID's or UUID's .................................................................................................. 11
  Object versions ...................................................................................................... 12
  Document references ............................................................................................ 13
Object Tree ................................................................................................................ 14
  Introduction .......................................................................................................... 14
  Object tree classes ............................................................................................... 14
Systems Engineering .................................................................................................. 16
  Contents .................................................................................................................. 16
  Layering .................................................................................................................. 16
  Baselines ................................................................................................................ 18
  Requirements and Performances ......................................................................... 18
  Verifications .......................................................................................................... 19
Topological relations .................................................................................................. 20
  Introduction .......................................................................................................... 20
  Topological classes .............................................................................................. 20
  Spatial topology ................................................................................................... 22
Version management .................................................................................................. 23
  Contents .................................................................................................................. 23
  Introduction .......................................................................................................... 23
  Representing deleted and modified information objects ..................................... 23
  Representing a deleted object ............................................................................. 24
  Representing a modified object .......................................................................... 25
  Linking policy with deleted and modified information objects ......................... 25
  Linking policy with deleted information objects ............................................... 26
  Linking policy with modified information objects ............................................. 26
  Bi-directional linking (inverse relation) policy ................................................... 27
  Merging .................................................................................................................. 28
States .......................................................................................................................... 29
Exchange .................................................................................................................... 30
Coins-container .......................................................................................................... 30
  Export procedure .................................................................................................. 30
  Import procedure ................................................................................................. 31
  COINS-IO ............................................................................................................. 31
Window of Authorization ............................................................................................ 32
  Contents .................................................................................................................. 32
  Window of Authorization (WoA) .......................................................................... 32
Abstract

This specification defines the 1.0 (2010) release of the COINS systematics including the COINS Building Information Model schema, the COINS Schema Architecture, the COINS semantics, the expected behavior of COINS compatible software and the COINS Container exchange format.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document.

If you wish to make comments or report errors regarding this document, please send them to mailto:peter.willems@tno.nl and/or mailto:h.schaap@gobar.nl.
Introduction

Normative References

Business Process Model and Notation (BPMN)
Extensible Markup Language (XML)
   Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3.org/TR/2008/REC-xml-20081126/
Industry Foundation Classes (IFC)
International Framework for Dictionaries (IFD)
Information Delivery Manual (IDM)
Resource Description Framework (RDF)
   RDF Vocabulary Description Language (RDFS)
STEP file format
Unified Modeling Language™ (UML®)
   UML 1.4.2 is available as ISO/IEC 19501, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML
Uniform Resource Identifier (URI)
VISI
   VISI-systematiek 1.2, http://www.crow.nl/visi/_t21_p38_m7_i6488.htm
Web Ontology Language (OWL)
Terms and definitions

BPMN

class
Classes provide an abstraction mechanism for grouping resources with similar characteristics.

COINS Systematics schema
The core-schema that contains the backbone structure (an object tree of function fulfillers) and directly associated information objects (functions, requirements, performances, documents and catalogue parts).

datatype properties
link objects to data values.

IDM
Information Delivery Manual: [http://www.iai.no/idm/](http://www.iai.no/idm/)

IFC

IFD
International Framework for Dictionaries: [http://www.buildingsmart.com/content/ifd](http://www.buildingsmart.com/content/ifd)

object
instantiation of a class.

object properties
link objects to objects.

OWL
Web Ontology Language: [http://www.w3.org/2004/OWL/](http://www.w3.org/2004/OWL/)

RDF
Resource Description Framework: [http://www.w3.org/RDF/](http://www.w3.org/RDF/)

RDFS
RDF Vocabulary Description Language: [http://www.w3.org/TR/rdf-schema/](http://www.w3.org/TR/rdf-schema/)

Reference frame
sub-schema that extends the COINS systematics core schema for specific application areas.

STEP

UML

URI

VISI
Voorwaarden scheppen voor invoeren van standaardisatie ICT in de bouw (Establish conditions to introduce standardization of ICT in the building industry): [http://www.visi.nl](http://www.visi.nl)

XML
Extended Mark-up Language: [http://www.w3.org/XML/](http://www.w3.org/XML/)
Schema architecture

Flexible architecture

COINS is characterized by flexible schema architecture. The core-schema contains the backbone structure (an object tree of function fulfillers) and directly associated information objects (functions, requirements, performances, documents and catalogue parts).

The core schema can be extended by a set of so-called reference frames: sub-schemas that extend the COINS systematics core schema for specific application areas. At this moment two reference frames are published:

- Reference frame Functional specification
- Reference frame Quantity estimation

The reference frames listed above and in the figure are sub-schemas that are recognized by the COINS Consortium. However, there are no objections to third parties that introduce other reference frames probably with a limited scope (within a company or during a specific project). Possibly such an external reference frame could be successful and migrate later to an "official" reference frame. COINS compatible applications may decide to support zero or more reference frames depending their specific functionality.
Object libraries

The COINS systematics core schema is designed as a minimal schema, i.e. the semantics of the schema is not particularly rich. To compensate for this shortcoming the COINS systematics explicitly supports linking with object libraries. For example the COINS systematics schema defines only a physical object class to specify the various tangible parts of a building. There are no more specific classes like column, wall or door. However, the physical object "door set" may link to an object library that specifies a sub/supertype class hierarchy. For example the buildingSMART IFD library contains a class description "door-set" [ID=3vHRQ8oT0Hsm00051Mm008]. Referencing this particular class definition from a COINS model is an efficient approach to add the desired semantics. Object libraries may have a (inter-)national scope but a temporary object library during the execution of a specific project may also be of service for example to hold the set of standard catalogue items for that project. For this last application a COINS model with catalogue part items is very convenient.
Special Topics

Identification of CBIM information objects

Guideline identification of CBIM information objects

Technical note

Contents

• 1 Introduction
• 2 GUID's or UUID's
• 3 Object versions
• 4 Document references

Introduction

CBIM objects are normally presented using their name and user ID attribute values. Although this identification method will be satisfactory for most users it is simply not reliable enough to count on. The unique identification of CBIM objects is a feature that usually is kept invisible to most users. Tools will suppress this bit of information because it is semantically irrelevant. Unique identification of information objects is a major concern for ICT applications to guarantee a reliable operation: they need to possess an attribute value they can count on to be different for each object instance always and everywhere.

Until recently the available COINS tools (COINS Navigator and API library) employed a simple method for assigning identification strings. For each model the object's local name was based on a serial number starting at 1 and incremented for each new object instance. In combination with the Uniform Resource Identifier (URI) of the model this produces a global unique identification following the format <model-uri>#<local-name> for example http://www.coinsweb.nl/pilots/strukton/carpark.owl#Function34. This strategy works fine when the object creation is centralized. However, CBIM objects will be created simultaneously at various locations in a, for the time being, semi-off-line way. This brought about the need for a globally unique naming convention which should be independent from the model URI although the designation of CBIM objects will still use the format <model-uri>#<local-name>. This policy offers the possibility to track CBIM objects over the Internet.

GUID's or UUID's

Creating globally unique identifications is a well described topic in the ICT literature. The term GUID usually refers to Microsoft's implementation of the Universally Unique Identifier (UUID) standard, however the term is common in applications not written by Microsoft, or for their...
The COINS-systematiek 1.0 - Reference Manual

operating systems. The GUID is a 16-byte (128-bit) number. GUID's are most commonly written in text as a sequence of hexadecimal digits such as: 3f2504e0-4f89-11d3-9a0c-0305e82c3301.

The guideline for the identification of CBIM information objects prescribes to exclusively use the string form of GUID's as the value for the local name attribute (rdf:ID) for all CBIM object classes. Because the syntax of a fragment identifier is defined as a NCName (XML "non-colonized" name) it should start with a letter or an underscore character. The guideline recommends the use of the underscore character.

For example: http://www.coinsweb.nl/pilots/strukton/carpark.owl#_3f2504e0-4f89-11d3-9a0c-0305e82c3301.

**Object versions**

In a COINS Building Information System information objects are never removed or modified. If the information content of a CBIM object need to be changed the original object instance will be marked expired while a new object instance will be created representing the new state of the original object (see also: http://www.coinsweb.nl/wiki/index.php/Version_management).

New object version after a modification.

The new version of the object is primarily a clone of the original object, i.e. the local name attribute value (= the globally unique ID) will be copied, too. Yet, having two objects with identical ID's in the same name space cannot be allowed. For this reason the local name is extended with the version serial number separated by a dot character. By convention the initial version number (0) can be omitted.

This leads to the final form of a CBIM object identification (in BNF notation):

- `<CBIM-object-ID> ::= <model-ID> # <local-name>`
- `<model-ID> ::= <model-name-space> / <model-file-name>`
- `<local-name> ::= _<guid> | _<guid> . <version-number>`
- `<version-number> ::= <integer>`

For example: http://www.coinsweb.nl/pilots/strukton/carpark.owl#_3f2504e0-4f89-11d3-9a0c-
Document references

CBIM objects can point to other CBIM objects using the guideline discussed above. Referring to information objects outside the CBIM name space is handled by objects of the CBIM Document class or descendant classes (e.g. Explicit 3D Representation and Visi Message). A document stored on a local file system can be addressed by its absolute file path. If the document is shipped in a COINS container the file path is re-routed to point to the doc folder of the container. Importing the container will result in unwrapping the document in the assigned local folder. A document need not to be physically transported if it can be reached over the Internet (this could be good practice for very large documents). The document is then addressed by its URI. Employing both the file path and URI address options could be handy to prevent downloading over and over again or to be able to operate off-line.

Usually the document will be addressed as a whole but sometimes the use of a fragment identifier could be convenient to link to a small portion in a large HTML document or to refer to a specific object in a shape representation model file. For example in an IFC model file the global ID attribute can be used to point to a specific object in the IFC file.
Object Tree

Introduction

The COINS Building information model (CBIM) is primarily structured around an object tree. Function Fulfillers are organized according the decomposition principle, i.e. part-of relationships between an assembly object (parent) and its constituting part objects (children). A part object in its turn may act as an assembly object for even smaller part objects, etcetera until atomic part objects (the leaves of the decomposition tree) are reached. Leaf objects are the basic building stones of this structure: we know either how to produce them or how to obtain from a third party (supply company).

Object tree classes

Figure 1 shows the UML representation of the object tree data structure. Since Function Fulfiller is an abstract class (no instances) the actual object tree contains physical objects or spaces.

These Function Fulfiller types cannot be mixed, i.e. a physical object cannot decompose into spaces and vice versa. This is explicitly specified in the supproperties physicalChild/physicalParent and spatialChild/spatialParent of the child/parent relations.

The interconnection between a physical object tree and a space tree can be specified with the isSituatedIn/situates relation.
Figure 2: A parent object and its immediate child objects should reside in adjacent layers. The parent layer index and the child layer index differ exactly one
Systems Engineering

Systems Engineering is an important guiding modelling principle underlying the COINS Systematics core schema. This influence is expressed in various ways and will be discussed in the following sections.

Contents

- 1 Layering
- 2 Baselines
- 3 Requirements and Performances
- 4 Verifications

Layering

Layering is an organizing principle to collect information objects with their related neighboring pieces of information and on a corresponding level of detail. Point of departure is always a function fulfills. The top node of a decomposition tree of function fulfillers resides normally in the first layer: layer 0. The child function function fulfillers are located in the following layer, i.e. the number of parent function fulfiller plus one, etc.

Neighboring objects of a function fulfiller are for example the function(s) that must be fulfilled. Those function(s) should lie in the same layer. Similarly, a requirement (partly) describing a function lies in the same layer or the performance (partly) describing a function fulfiller. Not all object classes follow this principle, for example a document may be referred to by objects in different layers. The Document class is therefore layer independent. As a result layering is only meaningful for classes that are closely related to systems engineer: Connection, Function, FunctionFulfiller, Performance, Requirement, Terminal and Verification.

As a rule all objects in the same layer represent the total building at a specific level of detail. For this reason all layers should always be specified completely even if certain branches typically need more decomposition layers then others. In that case the shorter branches must be extended until the occupied layer with the highest number. For the same reason a branch can never skip a layer.

To assign an information object to a certain layer its layerIndex attribute should be specified. Since this property has CbimObject as its domain type virtually all COINS classes may specify a layer index. As observed above only systems engineering classes may use this attribute value meaningfully. This leads to the following rules:

- the fullfills or isFulfilledBy object properties only allow domain and range objects with the same layerIndex value.
- the requirement or requirementOf object properties only allow domain and range objects with the same layerIndex value.
- the performance or performanceOf object properties only allow domain and range
objects with the same `layerIndex` value.

- the `verificationRequirement` and `verificationFunctionFulfiler` object properties only allow domain and range objects with the same `layerIndex` value.
- the `terminal` or `terminalOf` object properties only allow domain and range objects with the same `layerIndex` value.
- the `maleTerminal` and `femaleTerminal` object properties only allow domain and range objects that have adjacent `layerIndex` values.

Above rules force the information objects in this instance diagram to specify the same layer index value.

- the `parent` or `child` object properties only allow domain and range objects that have adjacent `layerIndex` values.

The last rule forces the information objects in this instance diagram to specify adjacent layer index values.

Copyright © 2010 COINS consortium
**Baselines**

*Baselines* form another organizing principle, which often (but not necessarily) coincides with one or more layers. Where layers are not information objects themselves (only an integer attribute value) a *baseline* is a class with explicit links to the participating information objects of that particular *baseline*.

An information object can only form part of one *baseline*.

A *baseline* has a *baselineStatus* property of type boolean, which determines the status open ( = *true*) or closed ( = *false*). An open *baseline* is open for changes (addition, deletion, updating), while a closed *baseline* is not. COINS compatible software should protect information objects in a closed *baseline* from unintentional changes. Only users of a recognized authorization level should be allowed to unlock the *baseline* (temporarily) and make the necessary modifications.

![Warning message of the COINS Navigator.](image)

**Requirements and Performances**

Functional *requirements* specify the *function* that contains them, i.e. a functional *requirement* belongs to exactly one *function*. *Performances* characterize the *function fulfiller* that contains them, i.e. a *performance* belongs to exactly one *function fulfiller*. The COINS systematics only supports a textual (human interpretable) specification of *requirements* and *performances*. As a result it is not possible to verify automatically *requirements* with the corresponding *performances*. The reference framework "Functional Specification" extends the *Requirement* and *Performance* classes with associated *PropertyTypes* and quantitative *PropertyValue* restrictions and actual values. These extensions support in principle the automatic checking of *requirements* and *performances*.

![Diagram showing relationships between function, function fulfiller, requirement, and performance.](image)

*A function is characterized by a set of (functional) requirements, while a function fulfiller is characterized by a set of performances.*

Copyright © 2010 COINS consortium
Verifications

A verification asserts explicitly the conformance of a function fulfiller to a specific requirement. This assertion is the outcome of formal test procedure performed by a designated person at a certain date.

A verification will generally be associated to a certain baseline, which offers the opportunity to specify several verifications at various stages during the life cycle.

Verifications panel in the COINS Navigator.
Topological relations

Introduction

The COINS Building information model (CBIM) is primarily structured around an object tree. Function fulfillers are organized according the decomposition principle, i.e. part-of relationships between an assembly object (parent) and its constituting part objects (children). A part object in its turn may act as an assembly object for even smaller part objects, etcetera until atomic part objects (the leaves of the decomposition tree) are reached. Leaf objects are the basic building stones of this structure: we know either how to produce them or how to obtain from a third party (supply company). It is perfectly acceptable to restrict the backbone structure to this object tree. However, certain applications need more information then just decomposition. E.g. structural analysis or energy transmission calculations need also access to the topological structure: which part is connected to which other parts. The primary connections are defined between leaf objects on the lowest decomposition layer, although in a more abstract sense assembly objects in more higher layers could also specify connections. Because of this topological relationships can be viewed as the horizontal structuring in contrast with the vertical structuring of decomposition.

Topological classes

Figure 1: UML diagram for topological relationships

Figure 1 shows a simplified UML diagram of the topological classes and relationships of CBIM (It should be emphasized that the formal CBIM definition is in OWL. UML diagrams cannot show all
particulars of the OWL specification and should only be used to get a general idea of the model architecture). A physical object (or more general a function_fulfiller) may have zero or more terminals. A terminal is a surface part that potentially may link to another terminal of another physical object. If the link is deliberately established this fact is acknowledged by a connection. Connections are only allowed between physical_objects in the same decomposition layer, though they need not share the same parent object.

![Figure 2: Example of a column-beam connection](image)

Of course, topological relationships typically have also a geometrical aspect. As the owner physical_object may refer to location and shape data its terminals may also refer to locations and shapes. Yet certain restrictions should be obeyed.

- The locator of a terminal should be treated as relative to the locator of the owning physical_object.
- The locations and primary orientations of two terminals should be the same and in line to be able to be connected. The primary orientation should be read as the normal vector on the surface. Under certain circumstances the secondary orientation (rotation around the normal vector) could be of interest. In that case the secondary orientations should also match.
- The (surface) shape representation of a terminal should lie on the surface contour of the owning physical_object.
- The shape representations of two connected terminals should match, i.e. they are co-planar or typically refer to the same shape representation.
- Connections situated on higher decomposition levels should match with the topological structure on the lowest decomposition layer. E.g. if a higher level connection is specified
between two assembly objects but the child objects of those assemblies do not appear to have any branch crossing connection then this is a certainly a contradicting situation (although this may easily occur during the construction of the BIM: the requirement should therefore only be enforced on a stable state of the model). In principle higher level connections can be derived from lower level connections.

### Spatial topology

Though most applications will be interested in physical (tangible) connections, spatial topology (topological relations between two spaces and even between a physical object and a space) are also possible. Early architectural design may already specify which spaces will be adjacent to a specific space. Energy calculations may be interested to know which walls delimit a certain space and finally configuration management may need knowledge about which physical objects are situated in which space. This last topological relation is established with the situates/isSituatedIn relation between spaces and physical objects.
Version management

CBIS Version Management
Technical description

Contents
- 1 Introduction
- 2 Representing deleted and modified information objects
  - 2.1 Representing a deleted object
  - 2.2 Representing a modified object
- 3 Linking policy with deleted and modified information objects
  - 3.1 Linking policy with deleted information objects
  - 3.2 Linking policy with modified information objects
  - 3.3 Bi-directional linking (inverse relation) policy
- 4 Merging

Introduction

Although many database systems support version management in one way or another there are good reasons to store version management data also in the information model itself. In this way one standard way of representing the history of the model can be preserved over various database implementations. Conservation of the history of model modifications is also core business for building information modeling. It is simply of utmost importance to keep track of which modifications were when introduced by whom operating in which role. Apart from that the native version management functions of the applied database system can still (and therefore should be) used for daily operation and where the COINS Building Information System (CBIS) version management data is handled in the same way as the actual content data.

The starting point for the CBIS Version Management design was the concept of preserving every model state. This means that the information model can in principle only grow because never an information object will be physically deleted. Of course the actual implementation of this principle could be quite an undertaking. Here we describe the design features to make such an implementation feasible, not the actual implementation itself. A reference implementation is available as part of the COINS Testing tool that can be downloaded from the COINS web site: http://www.coinsweb.nl/wiki/index.php/COINS_Testgereedschap.

Representing deleted and modified information objects

All information objects in a COINS Building Information Model (CBIM) are instances of a class that inherits directly or indirectly of the abstract super type CbimObject. CbimObject defines primarily a set of attributes for identification, meta data and management data. It also defines two
attributes which serve the implementation of version management: `expired` and `nextVersion`. Version management is typically implemented as part of a CBIS, i.e. the shared project CBIM that is under control of a so-called building director (in Dutch: bouwwerkregisseur). A CBIM as part of a COINS Container (a cabinet file containing the CBIM and all referenced document files) will typically only contain an up-to-date model without any previous versions of information objects.

These different uses of CBIM (as the central shared project model or as part of a information transaction using COINS containers) also affects some basic handling of the model objects. In a CBIM without version management information objects can explicitly be modified or deleted. However, in a CBIS CBIM an information object will never be updated or deleted. A modification of an object will always result in a new object containing the new state of the original object. The deletion of an object is effected by simply setting the expired flag. Only the creation of a new object is handled the same.

**Representing a deleted object**

As stated before under version management control an information object is never physically deleted from the model. The state of the deleted object is unaffectedly preserved. The only difference in state before and after the deletion is the expired flag which is set to true after the deletion.
Representing a modified object

![Diagram showing object before and after modification]

Figure 2: A CBIM representation of an object before and after the value of its attribute “a” is changed from “0” to “1”.

After an information object has been changed the original object instance points with its nextVersion property to the newly created object instance. The ID's of both objects will be the same. However, because two or more object instances cannot be represented with identical ID's in the same model the version number is added to the identical logical ID's to create different physical ID's. Logical ID and version number are separated by a dot character. Therefore the physical ID of a CBIM object is put together from the logical ID followed by a dot character followed by the version number: `<physical ID> ::= <logical ID>.<version number>`. By convention the first version (with version number = 0) may be skipped, so a physical ID without a “trailing dot plus integer number” shall be treated as `<logical ID>.0`.

Linking policy with deleted and modified information objects

Until here the discussion restricts itself to the implications for the data properties of an object, i.e. properties which refer to a literal value: a character string or a boolean, integer or real value. These values are unaffected (in case of deletion) or simply copied to the new object instance (in case of modification). In this section will be discussed how to handle the object properties or links, i.e. properties that refer to one or more other objects. Since object properties have a direction a distinction can be made between ingoing links and outgoing links.
Linking policy with deleted information objects

Deleting an information object, i.e. setting its expired flag, does not affect the ingoing and outgoing links.

However, there are implications in the semantics. In the example of figure 3 object A refers to object B through property linkA. After the deletion it still refers to object B, which is now expired. In a historical view the semantics are that up to a certain moment in time (the last modification date of the expired object) object A referred to object B. After that moment object B ceases to exist, which implicates that linkA from that instant effectively is reduced to a void link. As a result the actual view (which only displays existing or non-expired objects) should reveal that linkA is void or empty. In the same example object B refers through property linkB to object C. The historical view shows that once object B existed and referred to object C. In the actual view this historical fact has completely disappeared.

Linking policy with modified information objects

As discussed before modifying an information object results into a new object instance with identical logical ID and an incremented version number. Except for the expired flag all property values of the original object instance are cloned into the new instance. This holds for both data properties as well as object properties (outgoing links).
In the example of figure 4 object \( A \) is referring to object \( B \) through object property \( \text{linkA} \). After the modification of object \( B \) this link still points to object \( B \). But unlike the deleted object case now there is a non-void \( \text{nextVersion} \) link available which effectively points to the next version of object \( B \). If this version should also be expired and again a non-void \( \text{nextVersion} \) link is available this procedure could be repeated until a real expired object \( B \) version is encountered or a non-expired i.e. the current version of object \( B \).

The pre-modified version of object \( B \) refers to object \( C \) through object property \( \text{linkB} \). The new version of object \( B \) also copies this link to object \( C \) so that there are now two link instances \( \text{linkB} \) to object \( C \). The reason for copying the link is that the deletion of this link could be the trigger to create the new version. Copying the link shows that this is not the case here. Mark that simply deleting the new version instance (and implicitly the copied link) and resetting the expired flag in the previous version instance is sufficient to roll back to the original state.

**Bi-directional linking (inverse relation) policy**

Certain object property are defined bi-directional, i.e. there exists an inverse relationship. As a rule in a CBIM only one direction is explicitly specified (while for performance reasons the inverse direction could be inferred and temporarily added to the live model). The policy here is that adding a bi-directional link will always trigger the creation of a new object instance no matter which actual direction was chosen. So adding a bi-directional link between two existing objects will force both sides to create new versions.
Merging

The CBIS CBIM could be updated interactively by adding, deleting and updating objects one at a time. However, in a transaction based system (for instance using VISI) the updating takes place in a batch like manner. The building director will send out an assignment to a partner role in the project (specifier, designer, planner, etc.). The assignment is described in a formal message specified according to the VISI framework schema for that specific project. Part of this message is a COINS container which holds a relevant part of the actual CBIS CBIM including referenced documents. The person who fulfills the partner role at that moment will execute the assignment which results in an updated CBIM, i.e. some new objects are added, a few obsolete objects are removed and a number of existing objects are modified. This result is returned to the building director using again a COINS container. After acceptance this result must be added the the shared CBIS CBIM in a so called merge operation.

Merging could be interpreted as a sequence of separate modifications, however only the result is exchanged not the way this result has been achieved. In order to merge the CBIS CBIM and the received container CBIM must be compared to detect newly created objects, deleted objects and updated objects. To prevent the creation of superfluous new versions it is necessary to collect all changes on each object before the actual merging operation.

To realize an optimal control on the life cycle of each information object every affected object of the merge operation is attached a document reference to the VISI message that accompanied the container. This VISI message as a result of a specific transaction explains the changes in this version of the object.
States

A State represents a life-cycle stage of a function fulfiller. It bundles all information that is relevant for that particular life-cycle stage. A State object cannot exist independently but is always uniquely associated with its owner function fulfiller.

UML Class diagram of the State class and its direct relationships.

A State has a narrow relationship with Baseline. In fact States represent the various baseline aspects of one and the same Function Fulfiller. For this reason it is recommended practice to give the State object the same name as its associated Baseline object.

A Function Fulfiller refers to its most recent State through the currentState relation. A State refers to its direct predecessor through the previousState relation. All States refer to their owner Function Fulfiller through the stateOf relation.

UML Instance diagram of a function fulfiller and its associated states.
Exchange

Coins-container

The COINS Container is the central means of communication for object related content and associated documents between the various users/contributors of BIM data. Physically the container is a common Zip file with a predefined folder structure. At this stage three folders are recognized:

- **bim**
  In the *bim* folder the CBIM (sub)model can be found.

- **doc**
  The *doc* folder will contain all associated documents referenced from the CBIM in the *bim* folder. Here we may find 3D representation files, text documents, spreadsheets, etc.

- **woa**
  The *woa* folder is reserved for the window of authorization description file called *woa.xml*.

Export procedure

A COINS Container is composed during an export procedure by a COINS compatible application or a COINS Building Information System. During the export procedure a COINS model is serialized using RDF/XML formatting into a physical file. This model file is put into the *bim* folder of the COINS Container file.

As part of this transfer an inventory is drawn up of all the documents that are referenced by objects in this particular model. In principle, these documents are added to the *doc* folder of the COINS Container file. However, if a document is only referred to by a web address then this document reference is preserved as a web link. If copies of the document are both available on the web and on the local file system of the exporting party, a choice has to be made to either reference the document with a link to a copy in the *doc* folder or to reference it with a web link.

If the document file is physically into the *doc* folder the document object which contains the original file path to the document needs to be updated because the absolute file path is not relevant anymore. By convention the file path is reduced to just the file name. In case of documents with the same name an extra digit or letter should be added to prevent unintentional overwriting.

Exporting from a COINS Building Information System is little bit more complicated. Before the model can be serialized several filter operations are necessary:

- The export model will only contain last object versions that are non-expired. Previous object versions will only confuse the COINS compatible applications that are probably less sophisticated in this respect. The information of previous object versions is also not relevant in order to do their assignment correctly.

- The export model will not contain VISI messages (or other document references that contain information about how a particular object has turned up in the central model). Again, this is not relevant for COINS compatible applications.
• As a guideline it is cleaner to strip all version extensions from the export model. Version numbers are only meaningful within a COINS Building Information System. So for example an object with ID "http://www.example.com/COINS/testmodel.owl#_0d94476b-3b4b-4f4b-beb6-0f6973c6650a.6" is exported as "http://www.example.com/COINS/testmodel.owl#_0d94476b-3b4b-4f4b-beb6-0f6973c6650a" without the version delimiting dot and the version number itself.

• In case of a WoA (Window of Authorization) specification the objects that are marked non-accessible should be stripped from the export model. The WoA specification itself is stored in the woa folder of the COINS Container file.

**Import procedure**

Importing a COINS Container is relatively simple for a COINS compatible application:

• Firstly the COINS Container is unpacked in a working directory or folder. This is rather straight forward since the COINS Container is an ordinary ZIP file. The result will be a set of sub-folders: at least a bim folder and possibly also a doc and/or woa folder.

• Then the model file in the bim folder is loaded. After loading the document objects are updated to point with a correct absolute file path to the documents in the doc folder.

Importing a COINS Container in a COINS Building Information System is in principle the same procedure. However, normally the BIM manager will merge the contents of a COINS Container with the content of the shared project model. This means that the standard import procedure will be followed by the much more complex merge procedure. Merging is described in more detail in the Merging section of Version Management.

**COINS-IO**

The COINS-IO software module has built-in functions for both importing and exporting a COINS Container.
Window of Authorization

Contents

[Niet tonen]

• 1 Window of Authorization (WoA)
• 2 WoA Specification
• 3 WoA specification propagation rules
• 4 WoA during the transaction
  • 4.1 XML Schema
  • 4.2 WoA checking
• 5 WoA specification in the COINS Navigator
  • 5.1 Colour coding
  • 5.2 WoA violation messages

Window of Authorization (WoA)

An important aspect of the COINS information transaction process using COINS Containers is the control over the data access rights of the person who executes an assignment. A BIM contributor typically needs access to a subset of the central BIM and will generally add, update or delete information objects to this sub-model. It should be completely unambiguous if a certain information object is accessible (read access) or can be updated (write access).

The Window of Authorization is able to specify the access status for all information objects of the central CBIM in a very compact way. The main principle is to mark certain nodes of the object tree with explicit access rights (write access, read access or no access) and to propagate these access rights along the tree branches and along the information objects that can be reached from there.

The Window of Authorization specification forms no part of the CBIM itself. A central BIM may have many parallel WoA's in several concurrent transactions. A WoA is therefore saved in a separate XML file and shipped in a separate folder of the COINS Container.
The WoA specification uses the object tree as its base structure. The procedure is to specify the access rights for a set of function fillers explicitly while afterwards an algorithm uses some rules to decide what the access rights will be of all the other information objects (including object instances of other classes).

A function filler object may express three different access rights:

- **write access**
  Write access means that in principle all attribute and relation types of the function filler are open for updating (adding, removing, modifying). The 'in principle' addition means that certain restrictions may apply.
  The function filler that explicitly receives write access rights (in contrast to getting these rights as a derivation from an ancestor) are beforehand limited (limited write access) in updating its relations (especially in the same layer and above).

- **read access**
  Read access means that in principle all attribute and relation types of the function filler

- **limited access rights**
  Limited access rights mean that certain restrictions apply.
  Limited access rights are used to define the access rights for all other information objects (including object instances of other classes).
are open for reading (inspecting, traversing). The 'in principle' addition means that certain restrictions may apply.

The function fulfiller that explicitly receives read access rights (in contract to getting these rights as a derivation from an ancestor) are beforehand limited (limited read access) in inspecting and traversing its relations (especially in the same layer and above).

- **no access**

No access means that such an object can in no way be accessed. Even its very existence is hidden, which is practically achieved by leaving them out during the export procedure to a COINS Container.

The WoA specification can be serialized into an XML file. For example the WoA of the COINS BIM shown above could be expressed as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="nl/coinsweb/cbim/woa/WindowOfAuthorization.xsd">
  <woa:WriteAccess>
    <woa:RootObject layerDepth="1"
      objectID="http://www.coinsweb.nl/woa-example.owl#_b6f6ac82-295e-11b2-80a1-840ad48ff048">
      <woa:Name>B1.1</woa:Name>
      <woa:UserID/>
    </woa:RootObject>
  </woa:WriteAccess>
  <woa:ReadAccess>
    <woa:RootObject layerDepth="2"
      objectID="http://www.coinsweb.nl/woa-example.owl#_b6f6ac82-295e-11b2-80a1-840ad48ff048">
      <woa:Name>B1</woa:Name>
      <woa:UserID/>
    </woa:RootObject>
  </woa:ReadAccess>
</woa:WindowOfAuthorization>
```

The WoA specification may contain up to three sections: to specify the write access area, read access area and no access area respectively. In this example the no-access part is left out which means that it is implicitly defined: any object that has no write access or read access is automatically regarded to be a no-access object.

The write access area specification mentions one function fulfiller (B1.1) as a write access area root object that extends one layer deep and thus includes its children (B1.1.1, B1.1.2, B1.1.3). The link access element specifies that only child relations of the root object are open for updating (besides
the attribute values), which facilitates, in this particular case, adding or removing child objects to this root object.

```xml
<woa:ReadAccess>
  <woa:RootObject layerDepth="2"
    objectID="http://www.coinsweb.nl/woa-example.owl#_b6f6ac80-295e-11b2-80a1-840ad48ff048">
    <woa:Name>B1</woa:Name>
    <woa:UserID/>
  </woa:RootObject>
</woa:ReadAccess>
```

The read access area specification mentions one function fulfiller (B1) as a read access area root object that extends two layers deep and thus includes its children and grandchildren (B1.1, B1.2, B1.2.1). The write access area has precedence over the read access area so function fulfiller B1.1 is only upward read-only and its children are entirely not affected though they are located in the two-layer read access area.

**WoA specification propagation rules**

A WoA specification only specifies access rights for a subset of the function fulfillers. A set of rules dictates how to determine the access rights of the rest of the function fulfillers and all information objects that are no function fulfillers at all. Those rules are:

- If a so called WoA root object (i.e. a function fulfiller that has explicitly defined access rights) has a layer depth greater than 0 descendant function fulfillers inherit the access rights of this root object.
- Within the object tree a write access area overrules a read access area and a read access area overrules a no-access area.
- Function fulfillers outside any access area belong implicitly to the no-access area.
- Other (no function fulfiller) information objects that are referenced by a function fulfiller inherit the same access rights.
- If such an information object is referenced by more than one function fulfiller the most restricted rights take precedence, i.e. a read access right takes precedence of a write access right and a no-access rights takes always precedence other access rights. There is one exception with regard to baseline objects. The reason is that baselines often will contain a mix function fulfillers (write access, read access or no-access) and would end therefore as a no-access information object itself. That would be undesirable, besides baselines have a protection mechanism of its own: a baseline can be open or closed for updating. Of course only information objects with write access can be updated in an open baseline. Normally you won't expect information objects with write access in a closed baseline. When this situation occurs the closed baseline status should take precedence.
- The previous rule can be applied recurrently between the rest of the model objects. The algorithm should take care that all function fulfillers are assigned their access rights first. Other information objects cannot overrule this status afterwards.
- Finally, all (non-function fulfiller) information objects that cannot be reached using above procedure will receive by default write-access rights.
WoA during the transaction

The WoA specification XML file should be part of a COINS container. By convention, it is stored in a folder named woa.

XML Schema

The serialization of the WoA specification is in XML format. The XML Schema is shown below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:woa="http://www.coinsweb.nl"
    targetNamespace="http://www.coinsweb.nl"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="0.1">
  <xs:element
    name="WindowOfAuthorization"
    type="woa:WindowOfAuthorizationType">
  </xs:element>
</xs:schema>
```

Copyright © 2010 COINS consortium
WoA checking

There are two ways to check on violations of the WoA specification:

- during the actual updating by a COINS compatible application
- during a merge operation of a COINS container with the central BIM

The first option is primarily to assist the person charged to update the model. He/she will be informed immediately that a certain modification is not allowed by the prevailing WoA. Various (configurable) implementation forms can be used. The COINS Navigator for example warns but enables the user to neglect this warning. The idea behind it is the possibility that the WoA is too restrictively defined for the job in question. Informal communication could agree about broadening the write access area.

It could of course also happen that an application add-on cannot prevent (read: has insufficient access) WoA violations. In that case a separate WoA checker could do the job.

Before the actual merge operation a final check is done. The COINS Building Information System should offer sufficient control how to deal with encountered violations.

As a rule, more than one transaction will stand out at an arbitrary point in time. The BIM manager should have an overview of all those transactions and their corresponding WoA's. In principle, write access areas should not overlap. This will prevent update clashes. The WoA specification offers fine tuning in updating the relationships giving write access for relation type A (for example updating parent/child relations) while blocking write access for relation type B (for example attaching planning objects).
WoA specification in the COINS Navigator

A Window of Authorization can be specified using the object tree panel. Selecting a function fulfiller and right clicking shows a pop-up menu including a Window of Authorization sub-menu. The menu-items trigger the following functions:

- **Add write access**
  Declare this object a read/write access root.
- **Add read access**
  Declare this object a read-only access root.
- **Add no access**
  Declare this object inaccessible.
- **Link access**
  Sub-menu to fine-grain the link accessibility.
- **Remove access**
  Remove the access specification.
- **Increase layer depth**
  Increase the number of downward layers this access specification can reach. Initially the number of layers is set to 0 meaning: the same layer as the function fulfiller itself.
- **Decrease layer depth**
  Inverse function of increase layer depth.
- **Read WoA**
  Import a WoA description file.
- **Write WoA**
  Export a WoA description file.
- **Propagate WoA**
  Propagate the WoA root object specifications over all the information objects of the BIM.
- **Reset WoA**
  Remove the current WoA specification.
Colour coding

![Object Tree Panel Screenshot](http://www.coinsweb.nl/very-simple-case/beam.png)

**Figure 44: Example colour coding in the object tree panel.**

To have a quick clue about the access status of a certain information object the COINS Navigator uses colour coding:

- **Green**
  Full read/write access.
- **Turquoise**
  Limited read/write access for certain link types.
- **Blue**
  Full read-only access.
- **Violet**
  Limited read access for certain link types.
- **Red**
  No access.

The red colour code (no access rights) can only be observed in a CBIS environment. During the generation of a COINS Container the no-access information objects are simply skipped.

WoA violation messages

![Authorization Violation Warning](http://www.coinsweb.nl/very-simple-case/beam.png)

**Figure 45: Window of Authorization violation warning.**

The COINS Navigator actively checks all changes to a CBIM object. If a violation is detected a warning message is issued.

The COINS Navigator offers the option to ignore the WoA specification. The reason is that during the execution of the assignment it may turn out that the WoA was defined to restrictively. In that case it could be very cumbersome to generate a new COINS Container and restart the transaction. Then it could be handy to circumvent the authorization. Still there is a final check during the merge operation with the central BIM. At that point in the transaction this violation will be detected and must be approved or rejected.
The COINS Building Information Model (CBIM) is of course object based. However, until the ultimate goal that all available information is object based we have to deal with document oriented information. CBIM has a Document class to link to external documents which enables any CBIM object to attach a document even Document objects itself.

![UML Class diagram of the Document class and its subclasses.](image)

The Document class may specify a file path or a file URI or both to address the indicated document. There is no prescribed precedence rule if both addressing methods are used. If a document file URI is specified the user has to decide when the model is exported into a COINS Container if the document should be physically stored or only virtually as a hyperlink.

The Document class has several specializations:

- **Explicit 3D Representation**
  Used for 3D shape related documents.
- **Visi Message**
  Used to attach meta-data how this particular object showed up in the model.
- **Library Reference**
  Used to link to an object definition in an object library.
Explicit 3D Representation

Explicit 3D representation objects are typically addressed by a subset of the CBIM classes:

- FunctionFulfiller
- State
- Terminal
- CataloguePart

To distinguish these specific shape related documents they are specified using the `shape` relation.

If the shape document contains object based information where objects have unique addressable ID's an `Explicit 3D Representation` object may use the `documentUri` attribute to deep link to an object inside the shape document using a fragment delimiter.

If a `function_fulfiller` specifies `state` `shape` links can be discriminated to a certain state (e.g. early design, detail design, as-built, etc.). More generally a `shape` relation may address more than one `Explicit 3D Representation`. The recommended semantics is that in that case various representations of the same shape are intended. However, in case of deep linking in the same shape document the various shape objects are supposed to form one representation.

If the shape document/object publishes parameters that can be valued from outside an `Explicit 3D Representation` may specify `parameter` objects to specify specific values for specific parameters.
The COINS-systematiek 1.0 - Reference Manual

**UML Class diagram of the Parameter class and its subclasses.**

**UML Instance diagram of an Explicit 3D representation object that specifies parameter values for a linked parametrized shape object. By convention the parameter object name is identical to the parameter name of the shape object.**

**Visi Message**

**UML Class diagram of the VISI Message class and its subclasses.**

A **Visi Message** object is a document link that attaches a meta-data document to the object that specifies the link. In spite of the class name the referenced document could be any document and not necessarily a VISI message. The link is typically created during a merge operation of a COINS Container into a COINS Building Information System. New, updated or expired COINS objects will reference a meta-date object that clarifies the origin of their current state. In case of a VISI message...
the meta-data could contain:

- the transaction involved
- the executioner
- the persons in role
- message specific data

**Library Reference**

*UML Class diagram of the Library Reference class and its subclasses.*

A Library Reference object is a document link that addresses a definition in an external object library. Typical examples are:

- IFD Library
- Cheobs
- ETIM/2BA

However, a library reference could also link to company or project object library.
COINS-IO API

Contents

• 1 Introduction
• 2 API definition
• 3 Software

Introduction

The COINS-IO API is a platform independent interface to facilitate application programmers with functions to read and write CBIM model files and importing and exporting COINS containers. To fulfill the platform independence requirement the implementation of the IO functions is realized in Java. The API actually publishes two different interfaces:

• a tightly coupling

which is gained by direct calling of the Java functions (by a Java application or using a generated DLL (eg. IKVM)).

• a loosely coupling

which uses a SOAP protocol in a service oriented architecture.

COINS-IO Architecture

The SOAP protocol offers the opportunity to install COINS-IO as a client-server architecture where the server may or may not run on the same platform. This choice has some influence on the behaviour of some of the API functions. For example if the server does not run on the same platform local files need to be uploaded before the load or importContainer functions can be invoked. However, if these files can be retrieved from the web the location of the server is not relevant anymore.
The COINS-systematiek 1.0 - Reference Manual

Software to implement a COINS-IO interface yourself is available, see [#Software](#).

## API definition

<table>
<thead>
<tr>
<th>Function call</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addAttributeFloatValue(String, int, Float)</td>
<td>void</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td>addAttributeIntValue(String, int, Integer)</td>
<td>void</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td>addAttributeStringValue(String, int, String)</td>
<td>void</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td>addObject(int, String, String)</td>
<td>String</td>
<td>Create and add a new object to the model.</td>
</tr>
<tr>
<td>addObjectWithID(int, String, String, String)</td>
<td>void</td>
<td>Create and add object with ID to the model.</td>
</tr>
<tr>
<td>addOntology(int, String)</td>
<td>void</td>
<td>Add an imported ontology to the model.</td>
</tr>
<tr>
<td>addRelationTargetObject(String, String, String)</td>
<td>void</td>
<td>Add the relation target object.</td>
</tr>
<tr>
<td>cloneAttributeValues(String, String)</td>
<td>void</td>
<td>Clone the attribute values.</td>
</tr>
<tr>
<td>cloneRelations(String, String)</td>
<td>void</td>
<td>Clone the relations.</td>
</tr>
<tr>
<td>compareAttributeValues(String, String)</td>
<td>String[]</td>
<td>Compare the attribute values.</td>
</tr>
<tr>
<td>compareRelations(String, String)</td>
<td>String[]</td>
<td>Compare the relations.</td>
</tr>
<tr>
<td>create(String)</td>
<td>int</td>
<td>Create a new Coins model.</td>
</tr>
<tr>
<td>export(String, String, int)</td>
<td>void</td>
<td>obsolete see exportContainer(int, String)</td>
</tr>
<tr>
<td>exportContainer(int, String)</td>
<td>void</td>
<td>Exports a COINS container from a given model.</td>
</tr>
<tr>
<td>getAttributeBooleanValue(String, String)</td>
<td>boolean</td>
<td>Return the boolean typed attribute value.</td>
</tr>
<tr>
<td>getAttributeDateValue(String, String)</td>
<td>long</td>
<td>Return the date typed attribute value.</td>
</tr>
<tr>
<td>getAttributeDomain(int, String)</td>
<td>String</td>
<td>Get the domain class of an attribute type.</td>
</tr>
</tbody>
</table>

Copyright © 2010 COINS consortium
<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getAttributeFloatValue(String, String)</code></td>
<td>float</td>
<td>Return the float typed attribute value.</td>
</tr>
<tr>
<td><code>getAttributeFloatValues(String, String)</code></td>
<td>float[]</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td><code>getAttributeIntegerValue(String, String)</code></td>
<td>int</td>
<td>Return the integer typed attribute value.</td>
</tr>
<tr>
<td><code>getAttributeIntegerValues(String, String)</code></td>
<td>int[]</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td><code>getAttributeName(int)</code></td>
<td>String</td>
<td>obsolete see <code>getAttributes</code></td>
</tr>
<tr>
<td><code>getAttributesRange(int, String)</code></td>
<td>String</td>
<td>Return the literal range of an attribute type.</td>
</tr>
<tr>
<td><code>getAttributes(int)</code></td>
<td>String[]</td>
<td>Return the attribute type ID's of this model (plus imported models).</td>
</tr>
<tr>
<td><code>getAttributeStringValue(String, String)</code></td>
<td>String</td>
<td>Return the string typed attribute value.</td>
</tr>
<tr>
<td><code>getAttributeStringValues(String, String)</code></td>
<td>String[]</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td><code>getAttributeURI(int)</code></td>
<td>String</td>
<td>obsolete see <code>getAttributes</code></td>
</tr>
<tr>
<td><code>getAttributeValuesCount(String, int)</code></td>
<td>int</td>
<td>postponed Since there are no multiple valued attributes in CBIM</td>
</tr>
<tr>
<td><code>getClassAttributeCount(int)</code></td>
<td>int</td>
<td>obsolete see <code>getClassAttributes</code></td>
</tr>
<tr>
<td><code>getClassAttributeID(int, int)</code></td>
<td>int</td>
<td>obsolete see <code>getClassAttributes</code></td>
</tr>
<tr>
<td><code>getClassAttributes(int, String)</code></td>
<td>String[]</td>
<td>Return the attributeID's of a class.</td>
</tr>
<tr>
<td><code>getClassName(int)</code></td>
<td>String</td>
<td>obsolete see <code>getClasses</code></td>
</tr>
<tr>
<td><code>getClassRelationCount(int)</code></td>
<td>int</td>
<td>obsolete see <code>getClassRelations</code></td>
</tr>
<tr>
<td><code>getClassRelationID(int, int)</code></td>
<td>int</td>
<td>obsolete see <code>getClassRelations</code></td>
</tr>
<tr>
<td><code>getClassRelations(int, String)</code></td>
<td>String[]</td>
<td>Return the relationID's of a class.</td>
</tr>
<tr>
<td><code>getInverse(String)</code></td>
<td>String</td>
<td>Get the URI of the inverse of the specified relation or null.</td>
</tr>
<tr>
<td><code>getInverseName(int)</code></td>
<td>String</td>
<td>obsolete see <code>getInverse(String)</code></td>
</tr>
<tr>
<td><code>getModelURI(int)</code></td>
<td>String</td>
<td>Get the URI of the model.</td>
</tr>
<tr>
<td>Method</td>
<td>Return Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getObjectClass(String)</code></td>
<td>String</td>
<td>Return the class of an object.</td>
</tr>
<tr>
<td><code>getObjectID(int, int)</code></td>
<td>String</td>
<td>obsolete see <code>getObjectIDs</code></td>
</tr>
<tr>
<td><code>getObjectIDs(int)</code></td>
<td>String[]</td>
<td>Return the objectID's in this model.</td>
</tr>
<tr>
<td><code>getObjectsCount(int)</code></td>
<td>int</td>
<td>Return the number of objects in the model.</td>
</tr>
<tr>
<td><code>getObjectUri(String)</code></td>
<td>String</td>
<td>obsolete see <code>getObjectIDs</code></td>
</tr>
<tr>
<td><code>getOntologies(int)</code></td>
<td>String[]</td>
<td>List all imported ontologies.</td>
</tr>
<tr>
<td><code>getRelationName(int)</code></td>
<td>String[]</td>
<td>obsolete see <code>getRelations</code></td>
</tr>
<tr>
<td><code>getRelations(int)</code></td>
<td>String[]</td>
<td>Get the relation type ID's of this model (plus imported models).</td>
</tr>
<tr>
<td><code>getRelationsCount(String, String)</code></td>
<td>int</td>
<td>Return the number of instances of this relation.</td>
</tr>
<tr>
<td><code>getRelationTargetObject(String, String)</code></td>
<td>String</td>
<td>Return the ID of the target object of the relation.</td>
</tr>
<tr>
<td><code>getRelationTargetObjects(String, String)</code></td>
<td>String[]</td>
<td>Return the ID's of the target objects of the relation.</td>
</tr>
<tr>
<td><code>getRelationUri(int)</code></td>
<td>String</td>
<td>obsolete see <code>getRelations</code></td>
</tr>
<tr>
<td><code>getValueType(int)</code></td>
<td>int</td>
<td>obsolete see <code>getAttributeRange</code></td>
</tr>
<tr>
<td><code>import(String, String)</code></td>
<td>int</td>
<td>renamed (because of Java reserved word clash) see <code>importContainer</code></td>
</tr>
<tr>
<td><code>importContainer(String, String)</code></td>
<td>int</td>
<td>Imports a model from a given COINS container.</td>
</tr>
<tr>
<td><code>isAttributeNull(String, String)</code></td>
<td>boolean</td>
<td>Test if the attribute value for the specified object is set?</td>
</tr>
<tr>
<td><code>isMultiple(String, String)</code></td>
<td>boolean</td>
<td>Has the specified relation type used in the specified class a max cardinality of more than one?</td>
</tr>
<tr>
<td><code>isRelation(int)</code></td>
<td>boolean</td>
<td>obsolete Relations (object properties) and attributes (datatype properties) are discriminated through out the whole API.</td>
</tr>
<tr>
<td><code>load(String)</code></td>
<td>int</td>
<td>Load a Coins model. The uri may refer to a web file (uri: &quot;http:// ... /[Coins model file name].owl&quot;) or a local file in case client and server run on the same platform.</td>
</tr>
<tr>
<td><code>removeAttributeValue(String, String)</code></td>
<td>void</td>
<td>Remove (or unset) the attribute value of</td>
</tr>
</tbody>
</table>
### Software

Software is available to build a COINS-IO interface:

• A test function to see if communication is really stateful: http://www.e-bouw.org:8080/CoinsIO/services/CoinsIO/echo?string=Hello%20World (the "callCount" parameter must be incremented by 1 after each call.

• An example Java CoinsIO client: http://www.e-bouw.org:8080/CoinsIO/CoinsIOClient.java (with sample test utilities to demonstrate the operation of the API, but not intended to actually execute.)
WSDL script

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:ns="http://api.cbim.coinsweb.nl"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:documentation>Please Type your service description here</wsdl:documentation>
  <wsdl:types>
    <xs:schema attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://api.cbim.coinsweb.nl">
      <xs:element name="upload">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="filename" nillable="true"
              type="xs:string"/>
            <xs:element minOccurs="0" name="binary" nillable="true"
              type="xs:base64Binary"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="setRelationTargetObject">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="sourceObjectID" nillable="true"
              type="xs:string"/>
            <xs:element minOccurs="0" name="relationID" nillable="true"
              type="xs:string"/>
            <xs:element minOccurs="0" name="targetObjectID" nillable="true"
              type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="setAttributeStringValue">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="objectID" nillable="true"
              type="xs:string"/>
            <xs:element minOccurs="0" name="attributeID" nillable="true"
              type="xs:string"/>
            <xs:element minOccurs="0" name="value" nillable="true"
              type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
</wsdl:definitions>
```

Copyright © 2010 COINS consortium
<xs:element name="setAttributeIntegerValue">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="value" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="setAttributeFloatValue">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="value" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="setAttributeDateValue">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="value" type="xs:long"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="setAttributeBooleanValue">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="value" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="save">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="file" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="modelID" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="load">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="uri" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
The COINS-systematiek 1.0 - Reference Manual

```xml
<xs:element name="loadResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationsCount">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="sourceObjectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="relationID" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationsCountResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationTargetObjects">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="sourceObjectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="relationID" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationTargetObjectsResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationTargetObject">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="sourceObjectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="relationID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="index" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getRelationTargetObjectResponse">
  <xs:complexType>
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Copyright © 2010 COINS consortium
<xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectsCount">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="modelID" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectsCountResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectIDs">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="modelID" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectIDsResponse">
<xs:complexType>
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectClass">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getObjectClassResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributes">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="modelID" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributesResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeStringValue">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectId" nillable="true" type="xs:string"/>
<xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeStringValueResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeIntegerValue">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectId" nillable="true" type="xs:string"/>
<xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeIntegerValueResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeFloatValue">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectId" nillable="true" type="xs:string"/>
<xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeFloatValueResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:float"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeDateValue">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="objectId" nillable="true" type="xs:string"/>
<xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getAttributeDateValueResponse">
<xs:complexType>
<xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:dateTime"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType>
  <xs:sequence>
    <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="getAttributeDateValueResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" type="xs:long"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getAttributeBooleanValue">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="objectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="attributeID" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getAttributeBooleanValueResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="echo">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="string" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="echoResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="create">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="uri" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="createResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="uri" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType>
  <xs:sequence>
    <xs:element minOccurs="0" name="return" type="xs:int"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="addRelationTargetObject">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="sourceObjectID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="relationID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="targetObjectID" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="addObject">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="modelID" type="xs:int"/>
      <xs:element minOccurs="0" name="classID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="objectName" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="addObjectResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<wsdl:schema>
The COINS-systematiek 1.0 - Reference Manual

</wsdl:message>
<wsdl:message name="getObjectClassRequest">
  <wsdl:part name="parameters" element="ns:getObjectClass"/>
</wsdl:message>
<wsdl:message name="getObjectClassResponse">
  <wsdl:part name="parameters" element="ns:getObjectClassResponse"/>
</wsdl:message>
<wsdl:message name="saveRequest">
  <wsdl:part name="parameters" element="ns:save"/>
</wsdl:message>
<wsdl:message name="getAttributesRequest">
  <wsdl:part name="parameters" element="ns:getAttributes"/>
</wsdl:message>
<wsdl:message name="getAttributesResponse">
  <wsdl:part name="parameters" element="ns:getAttributesResponse"/>
</wsdl:message>
<wsdl:message name="getAttributeIntegerValueRequest">
  <wsdl:part name="parameters" element="ns:getAttributeIntegerValue"/>
</wsdl:message>
<wsdl:message name="getAttributeIntegerValueResponse">
  <wsdl:part name="parameters" element="ns:getAttributeIntegerValueResponse"/>
</wsdl:message>
<wsdl:message name="getAttributeFloatValueRequest">
  <wsdl:part name="parameters" element="ns:getAttributeFloatValue"/>
</wsdl:message>
<wsdl:message name="getAttributeFloatValueResponse">
  <wsdl:part name="parameters" element="ns:getAttributeFloatValueResponse"/>
</wsdl:message>
<wsdl:message name="setAttributeBooleanValueRequest">
  <wsdl:part name="parameters" element="ns:setAttributeBooleanValue"/>
</wsdl:message>
<wsdl:message name="setRelationTargetObjectRequest">
  <wsdl:part name="parameters" element="ns:setRelationTargetObject"/>
</wsdl:message>
<wsdl:message name="getRelationsCountRequest">
  <wsdl:part name="parameters" element="ns:getRelationsCount"/>
</wsdl:message>
<wsdl:message name="getRelationsCountResponse">
  <wsdl:part name="parameters" element="ns:getRelationsCountResponse"/>
</wsdl:message>
<wsdl:message name="setAttributeIntegerValueRequest">
  <wsdl:part name="parameters" element="ns:setAttributeIntegerValue"/>
</wsdl:message>
<wsdl:message name="getAttributeDateValueRequest">
  <wsdl:part name="parameters" element="ns:getAttributeDateValue"/>
</wsdl:message>
<wsdl:message name="getAttributeDateValueResponse">
  <wsdl:part name="parameters" element="ns:getAttributeDateValueResponse"/>
</wsdl:message>
<wsdl:message name="getAttributeStringValueRequest">
  <wsdl:part name="parameters" element="ns:getAttributeStringValue"/>
</wsdl:message>
<wsdl:message name="getAttributeStringValueResponse">
  <wsdl:part name="parameters" element="ns:getAttributeStringValueResponse"/>
</wsdl:message>
<wsdl:message name="getAttributeBooleanValueRequest">
  <wsdl:part name="parameters" element="ns:getAttributeBooleanValue"/>
</wsdl:message>
<wsdl:part name="parameters" element="ns:getAttributeBooleanValue"/>
</wsdl:message>
<wsdl:message name="getAttributeBooleanValueResponse">
<wsdl:part name="parameters" element="ns:getAttributeBooleanValueResponse"/>
</wsdl:message>
<wsdl:message name="getRelationTargetObjectRequest">
<wsdl:part name="parameters" element="ns:getRelationTargetObject"/>
</wsdl:message>
<wsdl:message name="getRelationTargetObjectResponse">
<wsdl:part name="parameters" element="ns:getRelationTargetObjectResponse"/>
</wsdl:message>
<wsdl:message name="getRelationTargetObjectsRequest">
<wsdl:part name="parameters" element="ns:getRelationTargetObjects"/>
</wsdl:message>
<wsdl:message name="getRelationTargetObjectsResponse">
<wsdl:part name="parameters" element="ns:getRelationTargetObjectsResponse"/>
</wsdl:message>
<wsdl:message name="setAttributeStringValueRequest">
<wsdl:part name="parameters" element="ns:setAttributeStringValue"/>
</wsdl:message>
<wsdl:message name="addRelationTargetObjectRequest">
<wsdl:part name="parameters" element="ns:addRelationTargetObject"/>
</wsdl:message>
<wsdl:message name="echoRequest">
<wsdl:part name="parameters" element="ns:echo"/>
</wsdl:message>
<wsdl:message name="echoResponse">
<wsdl:part name="parameters" element="ns:echoResponse"/>
</wsdl:message>
<wsdl:message name="uploadRequest">
<wsdl:part name="parameters" element="ns:upload"/>
</wsdl:message>
<wsdl:message name="setAttributeDateValueRequest">
<wsdl:part name="parameters" element="ns:setAttributeDateValue"/>
</wsdl:message>
<wsdl:message name="getObjectsCountRequest">
<wsdl:part name="parameters" element="ns:getObjectsCount"/>
</wsdl:message>
<wsdl:message name="getObjectsCountResponse">
<wsdl:part name="parameters" element="ns:getObjectsCountResponse"/>
</wsdl:message>
<wsdl:message name="loadRequest">
<wsdl:part name="parameters" element="ns:load"/>
</wsdl:message>
<wsdl:message name="loadResponse">
<wsdl:part name="parameters" element="ns:loadResponse"/>
</wsdl:message>
<wsdl:portType name="CoinsIOPortType">
<wsdl:operation name="create">
<wsdl:input message="ns:createRequest" wsaw:Action="urn:create"/>
<wsdl:output message="ns:createResponse" wsaw:Action="urn:createResponse"/>
</wsdl:operation>
<wsdl:operation name="setAttributeFloatValue">
<wsdl:input message="ns:setAttributeFloatValueRequest" wsaw:Action="urn:setAttributeFloatValue"/>
</wsdl:operation>
</wsdl:portType>
The COINS-systematiek 1.0 - Reference Manual

```xml
<wsdl:operation name="getObjectIDs">
    <wsdl:input message="ns:getObjectIDsRequest" wsaw:Action="urn:getObjectIDsResponse" />
    <wsdl:output message="ns:getObjectIDsResponse" wsaw:Action="urn:getObjectIDsResponse" />
</wsdl:operation>

<wsdl:operation name="addObject">
    <wsdl:input message="ns:addObjectRequest" wsaw:Action="urn:addObjectResponse" />
    <wsdl:output message="ns:addObjectResponse" wsaw:Action="urn:addObjectResponse" />
</wsdl:operation>

<wsdl:operation name="getObjectClass">
    <wsdl:input message="ns:getObjectClassRequest" wsaw:Action="urn:getObjectClassResponse" />
    <wsdl:output message="ns:getObjectClassResponse" wsaw:Action="urn:getObjectClassResponse" />
</wsdl:operation>

<wsdl:operation name="save">
    <wsdl:input message="ns:saveRequest" wsaw:Action="urn:save" />
</wsdl:operation>

<wsdl:operation name="getAttributes">
    <wsdl:input message="ns:getAttributesRequest" wsaw:Action="urn:getAttributesResponse" />
    <wsdl:output message="ns:getAttributesResponse" wsaw:Action="urn:getAttributesResponse" />
</wsdl:operation>

<wsdl:operation name="getAttributeIntegerValue">
    <wsdl:input message="ns:getAttributeIntegerValueRequest" wsaw:Action="urn:getAttributeIntegerValueResponse" />
    <wsdl:output message="ns:getAttributeIntegerValueResponse" wsaw:Action="urn:getAttributeIntegerValueResponse" />
</wsdl:operation>

<wsdl:operation name="getAttributeFloatValue">
    <wsdl:input message="ns:getAttributeFloatValueRequest" wsaw:Action="urn:getAttributeFloatValueResponse" />
    <wsdl:output message="ns:getAttributeFloatValueResponse" wsaw:Action="urn:getAttributeFloatValueResponse" />
</wsdl:operation>

<wsdl:operation name="setAttributeBooleanValue">
    <wsdl:input message="ns:setAttributeBooleanValueRequest" wsaw:Action="urn:setAttributeBooleanValueResponse" />
    <wsdl:output message="ns:setAttributeBooleanValueResponse" wsaw:Action="urn:setAttributeBooleanValueResponse" />
</wsdl:operation>

<wsdl:operation name="setRelationTargetObject">
    <wsdl:input message="ns:setRelationTargetObjectRequest" wsaw:Action="urn:setRelationTargetObjectResponse" />
</wsdl:operation>

<wsdl:operation name="getRelationsCount">
    <wsdl:input message="ns:getRelationsCountRequest" wsaw:Action="urn:getRelationsCountResponse" />
    <wsdl:output message="ns:getRelationsCountResponse" wsaw:Action="urn:getRelationsCountResponse" />
</wsdl:operation>

<wsdl:operation name="setAttributeIntegerValue">
    <wsdl:input message="ns:setAttributeIntegerValueRequest" wsaw:Action="urn:setAttributeIntegerValueResponse" />
    <wsdl:output message="ns:setAttributeIntegerValueResponse" wsaw:Action="urn:setAttributeIntegerValueResponse" />
</wsdl:operation>

<wsdl:operation name="getAttributeDateValue">
    <wsdl:input message="ns:getAttributeDateValueRequest" wsaw:Action="urn:getAttributeDateValueResponse" />
    <wsdl:output message="ns:getAttributeDateValueResponse" wsaw:Action="urn:getAttributeDateValueResponse" />
</wsdl:operation>
```

Copyright © 2010 COINS consortium
<wsdl:input message="ns:getAttributeDateValueRequest" wsaw:Action="urn:getAttributeDateValue"/>
<wsdl:output message="ns:getAttributeDateValueResponse" wsaw:Action="urn:getAttributeDateValueResponse"/>
</wsdl:operation>
<wsdl:operation name="getAttributeStringValue">
<wsdl:input message="ns:getAttributeStringValueRequest" wsaw:Action="urn:getAttributeStringValue"/>
<wsdl:output message="ns:getAttributeStringValueResponse" wsaw:Action="urn:getAttributeStringValueResponse"/>
</wsdl:operation>
<wsdl:operation name="getAttributeBooleanValue">
<wsdl:input message="ns:getAttributeBooleanValueRequest" wsaw:Action="urn:getAttributeBooleanValue"/>
<wsdl:output message="ns:getAttributeBooleanValueResponse" wsaw:Action="urn:getAttributeBooleanValueResponse"/>
</wsdl:operation>
<wsdl:operation name="getRelationTargetObject">
<wsdl:input message="ns:getRelationTargetObjectRequest" wsaw:Action="urn:getRelationTargetObject"/>
<wsdl:output message="ns:getRelationTargetObjectResponse" wsaw:Action="urn:getRelationTargetObjectResponse"/>
</wsdl:operation>
<wsdl:operation name="getRelationTargetObjects">
<wsdl:input message="ns:getRelationTargetObjectsRequest" wsaw:Action="urn:getRelationTargetObjects"/>
<wsdl:output message="ns:getRelationTargetObjectsResponse" wsaw:Action="urn:getRelationTargetObjectsResponse"/>
</wsdl:operation>
<wsdl:operation name="setAttributeStringValue">
<wsdl:input message="ns:setAttributeStringValueRequest" wsaw:Action="urn:setAttributeStringValue"/>
</wsdl:operation>
<wsdl:operation name="addRelationTargetObject">
<wsdl:input message="ns:addRelationTargetObjectRequest" wsaw:Action="urn:addRelationTargetObject"/>
<wsdl:output message="ns:addRelationTargetObjectResponse" wsaw:Action="urn:addRelationTargetObjectResponse"/>
</wsdl:operation>
<wsdl:operation name="echo">
<wsdl:input message="ns:echoRequest" wsaw:Action="urn:echo"/>
<wsdl:output message="ns:echoResponse" wsaw:Action="urn:echoResponse"/>
</wsdl:operation>
<wsdl:operation name="upload">
<wsdl:input message="ns:uploadRequest" wsaw:Action="urn:upload"/>
<wsdl:output message="ns:uploadResponse" wsaw:Action="urn:uploadResponse"/>
</wsdl:operation>
<wsdl:operation name="setAttributeDateValue">
<wsdl:input message="ns:setAttributeDateValueRequest" wsaw:Action="urn:setAttributeDateValue"/>
<wsdl:output message="ns:setAttributeDateValueResponse" wsaw:Action="urn:setAttributeDateValueResponse"/>
</wsdl:operation>
<wsdl:operation name="getObjectsCount">
<wsdl:input message="ns:getObjectsCountRequest" wsaw:Action="urn:getObjectsCount"/>
<wsdl:output message="ns:getObjectsCountResponse" wsaw:Action="urn:getObjectsCountResponse"/>
</wsdl:operation>
<wsdl:operation name="load">
<wsdl:input message="ns:loadRequest" wsaw:Action="urn:load"/>
<wsdl:output message="ns:loadResponse" wsaw:Action="urn:loadResponse"/>
</wsdl:operation>
The COINS-systematiek 1.0 - Reference Manual

<wSDL:portType>
      <wSDL:operation name="create">
            <soap:operation soapAction="urn:create" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
            <wSDL:output>
                  <soap:body use="literal"/>
            </wSDL:output>
      </wSDL:operation>
      <wSDL:operation name="getObjectClass">
            <soap:operation soapAction="urn:getObjectClass" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
            <wSDL:output>
                  <soap:body use="literal"/>
            </wSDL:output>
      </wSDL:operation>
      <wSDL:operation name="addObject">
            <soap:operation soapAction="urn:addObject" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
            <wSDL:output>
                  <soap:body use="literal"/>
            </wSDL:output>
      </wSDL:operation>
      <wSDL:operation name="setAttributeFloatValue">
            <soap:operation soapAction="urn:setAttributeFloatValue" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
      </wSDL:operation>
      <wSDL:operation name="getObjectIDs">
            <soap:operation soapAction="urn:getObjectIDs" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
            <wSDL:output>
                  <soap:body use="literal"/>
            </wSDL:output>
      </wSDL:operation>
      <wSDL:operation name="save">
            <soap:operation soapAction="urn:save" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
      </wSDL:operation>
      <wSDL:operation name="getAttributes">
            <soap:operation soapAction="urn:getAttributes" style="document"/>
            <wSDL:input>
                  <soap:body use="literal"/>
            </wSDL:input>
      </wSDL:operation>
</wSDL:portType>
<wsdl:operation name="getAttributeIntegerValue">
  <soap:operation soapAction="urn:getAttributeIntegerValue" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="setAttributeBooleanValue">
  <soap:operation soapAction="urn:setAttributeBooleanValue" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="setRelationTargetObject">
  <soap:operation soapAction="urn:setRelationTargetObject" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getAttributeFloatValue">
  <soap:operation soapAction="urn:getAttributeFloatValue" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getRelationsCount">
  <soap:operation soapAction="urn:getRelationsCount" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="setAttributeIntegerValue">
  <soap:operation soapAction="urn:setAttributeIntegerValue" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getAttributeStringValue">
  <soap:operation soapAction="urn:getAttributeStringValue" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
The COINS-systematiek 1.0 - Reference Manual

```xml
<wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAttributeDateValue">
<soap:operation soapAction="urn:getAttributeDateValue" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAttributeBooleanValue">
<soap:operation soapAction="urn:getAttributeBooleanValue" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setAttributeStringValue">
<soap:operation soapAction="urn:setAttributeStringValue" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
</wsdl:operation>
<wsdl:operation name="getRelationTargetObjects">
<soap:operation soapAction="urn:getRelationTargetObjects" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRelationTargetObject">
<soap:operation soapAction="urn:getRelationTargetObject" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addRelationTargetObject">
<soap:operation soapAction="urn:addRelationTargetObject" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
</wsdl:operation>
```

Copyright © 2010 COINS consortium
The COINS-systematiek 1.0 - Reference Manual

  <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="addObject">
  <soap12:operation soapAction="urn:addObject" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="setAttributeFloatValue">
  <soap12:operation soapAction="urn:setAttributeFloatValue" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getObjectIDs">
  <soap12:operation soapAction="urn:getObjectIDs" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="save">
  <soap12:operation soapAction="urn:save" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAttributes">
  <soap12:operation soapAction="urn:getAttributes" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAttributeIntegerValue">
  <soap12:operation soapAction="urn:getAttributeIntegerValue" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="setAttributeBooleanValue">
  <soap12:operation soapAction="urn:setAttributeBooleanValue" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

Copyright © 2010 COINS consortium  65
The COINS-systematiek 1.0 - Reference Manual

Copyright © 2010 COINS consortium
<soap12:body use="literal"/>
</wsdl:input>
</wsdl:operation>

<wsdl:operation name="getObjectsCount">
  <soap12:operation soapAction="urn:getObjectsCount" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="load">
  <soap12:operation soapAction="urn:load" style="document"/>
  <wsdl:input>
    <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="CoinsIOHttpBinding" type="ns:CoinsIOPortType">
  <http:binding verb="POST"/>
  <wsdl:operation name="create">
    <http:operation location="CoinsIO/create"/>
    <wsdl:input>
      <mime:content type="text/xml" part="create"/>
    </wsdl:input>
    <wsdl:output>
      <mime:content type="text/xml" part="create"/>
    </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="getObjectClass">
    <http:operation location="CoinsIO/getObjectClass"/>
    <wsdl:input>
      <mime:content type="text/xml" part="getObjectClass"/>
    </wsdl:input>
    <wsdl:output>
      <mime:content type="text/xml" part="getObjectClass"/>
    </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="addObject">
    <http:operation location="CoinsIO/addObject"/>
    <wsdl:input>
      <mime:content type="text/xml" part="addObject"/>
    </wsdl:input>
    <wsdl:output>
      <mime:content type="text/xml" part="addObject"/>
    </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="setAttributeFloatValue">
    <http:operation location="CoinsIO/setAttributeFloatValue"/>
    <wsdl:input>
      <mime:content type="text/xml" part="setAttributeFloatValue"/>
    </wsdl:input>
    <wsdl:output>
    </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="getObjectIDs">
    <http:operation location="CoinsIO/getObjectIDs"/>
    <wsdl:input>
    </wsdl:input>
    <wsdl:output>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:input>
  <mime:content type="text/xml" part="getObjectIDs"/>
</wsdl:input>
<wsdl:output>
  <mime:content type="text/xml" part="getObjectIDs"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="save">
  <http:operation location="CoinsIO/save"/>
  <wsdl:input>
    <mime:content type="text/xml" part="save"/>
  </wsdl:input>
</wsdl:operation>
<wsdl:operation name="getAttributes">
  <http:operation location="CoinsIO/getAttributes"/>
  <wsdl:input>
    <mime:content type="text/xml" part="getAttributes"/>
  </wsdl:input>
  <wsdl:output>
    <mime:content type="text/xml" part="getAttributes"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAttributeIntegerValue">
  <http:operation location="CoinsIO/getAttributeIntegerValue"/>
  <wsdl:input>
    <mime:content type="text/xml" part="getAttributeIntegerValue"/>
  </wsdl:input>
  <wsdl:output>
    <mime:content type="text/xml" part="getAttributeIntegerValue"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="setAttributeBooleanValue">
  <http:operation location="CoinsIO/setAttributeBooleanValue"/>
  <wsdl:input>
    <mime:content type="text/xml" part="setAttributeBooleanValue"/>
  </wsdl:input>
</wsdl:operation>
<wsdl:operation name="setRelationTargetObject">
  <http:operation location="CoinsIO/setRelationTargetObject"/>
  <wsdl:input>
    <mime:content type="text/xml" part="setRelationTargetObject"/>
  </wsdl:input>
</wsdl:operation>
<wsdl:operation name="getAttributeFloatValue">
  <http:operation location="CoinsIO/getAttributeFloatValue"/>
  <wsdl:input>
    <mime:content type="text/xml" part="getAttributeFloatValue"/>
  </wsdl:input>
  <wsdl:output>
    <mime:content type="text/xml" part="getAttributeFloatValue"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRelationsCount">
  <http:operation location="CoinsIO/getRelationsCount"/>
  <wsdl:input>
    <mime:content type="text/xml" part="getRelationsCount"/>
  </wsdl:input>
</wsdl:operation>
The COINS-systematiek 1.0 - Reference Manual

```xml
<wSDL:operation name="addRelationTargetObject">
  <http:operation location="CoinsIO/addRelationTargetObject"/>
  <wSDL:input>
    <mime:content type="text/xml" part="addRelationTargetObject"/>
  </wSDL:input>
</wSDL:operation>

<wSDL:operation name="echo">
  <http:operation location="CoinsIO/echo"/>
  <wSDL:input>
    <mime:content type="text/xml" part="echo"/>
  </wSDL:input>
  <wSDL:output>
    <mime:content type="text/xml" part="echo"/>
  </wSDL:output>
</wSDL:operation>

<wSDL:operation name="upload">
  <http:operation location="CoinsIO/upload"/>
  <wSDL:input>
    <mime:content type="text/xml" part="upload"/>
  </wSDL:input>
</wSDL:operation>

<wSDL:operation name="setAttributeDateValue">
  <http:operation location="CoinsIO/setAttributeDateValue"/>
  <wSDL:input>
    <mime:content type="text/xml" part="setAttributeDateValue"/>
  </wSDL:input>
</wSDL:operation>

<wSDL:operation name="getObjectsCount">
  <http:operation location="CoinsIO/getObjectsCount"/>
  <wSDL:input>
    <mime:content type="text/xml" part="getObjectsCount"/>
  </wSDL:input>
  <wSDL:output>
    <mime:content type="text/xml" part="getObjectsCount"/>
  </wSDL:output>
</wSDL:operation>

<wSDL:operation name="load">
  <http:operation location="CoinsIO/load"/>
  <wSDL:input>
    <mime:content type="text/xml" part="load"/>
  </wSDL:input>
  <wSDL:output>
    <mime:content type="text/xml" part="load"/>
  </wSDL:output>
</wSDL:operation>
</wSDL:binding>
<wSDL:service name="CoinsIO">
  <wSDL:port name="CoinsIOHttpSoap11Endpoint" binding="ns:CoinsIOSoap11Binding">
    <soap:address location="http://localhost:8080/CoinsIO/services/CoinsIO.CoinsIOHttpSoap11Endpoint"/>
  </wSDL:port>
  <wSDL:port name="CoinsIOHttpSoap12Endpoint" binding="ns:CoinsIOSoap12Binding">
    <soap12:address location="http://localhost:8080/CoinsIO/services/CoinsIO.CoinsIOHttpSoap12Endpoint"/>
  </wSDL:port>
</wSDL:service>

Copyright © 2010 COINS consortium  71
The COINS-systematiek 1.0 - Reference Manual

<wsdl:port name="CoinsIOHttpEndpoint" binding="ns:CoinsIOHttpBinding">
  <http:address location="http://localhost:8080/CoinsIO/services/CoinsIO.CoinsIOHttpEndpoint/"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
The COINS-systematiek 1.0 - Reference Manual

COINS Building Information System

Functional specification of a COINS-compatible Building Information System (CBIS)

Version: 4.1
Date: 10 November 2009
Prepared by: Working group 'Specification'
Members: Henk Schaap, Peter Willems, Arjen Adriaanse, Renzo van Rijswijk, Hans Jongedijk, René Dorleijn, Bram Mommers

Contents

- 1 Introduction
- 2 Global architecture
- 3 COINS BIM specification
  - 3.1 Web Ontology Language (OWL)
  - 3.2 Industry Foundation Classes (IFC)
  - 3.3 Plug-in reference frameworks
- 4 CBIS functionality
  - 4.1 Importing a COINS-container
  - 4.2 Merging the contributions of various project partners into one central COINS Building Information Model (CBIM)
  - 4.3 Tracing modifications
  - 4.4 Keeping up with the consecutive versions of information objects
  - 4.5 Keeping up with baselines
  - 4.6 Seeing to that only designated persons (using explicit authorizations) are allowed to make changes to certain information objects
  - 4.7 Browsing the CBIM from various viewpoints (systems engineering, 3D representation, verification, etc.)
  - 4.8 Extracting sections of the central BIM for export in a COINS container
  - 4.9 Recording of receipt messages (VISI)
  - 4.10 Reporting of performed modifications
  - 4.11 Application Programming Interface (API)
  - 4.12 Multi-user environment
- 5 Appendix A: Terms and definitions
- 6 Appendix B: COINS Navigator
- 7 Appendix C: A very simple case
Introduction

To maintain the building information in a building construction project a building information system dealing with object information is essential. The availability of a COINS-compatible building information system is a prerequisite for a broad implementation / deployment of the COINS standard. Therefore, in the COINS project group meeting of September 23, 2009 it is decided to send out an appeal to the software market to deliver a COINS-compatible building information system.

Initially, such a building information system should support the following functionality:

- Importing a COINS-container
- Merging the contributions of various project partners into one central COINS Building Information Model (CBIM)
- Tracing modifications
- Keeping up with the consecutive versions of information objects
- Keeping up with baselines
- Seeing to that only designated persons (using explicit authorizations) are allowed to make changes to certain information objects
- Browsing the CBIM from various viewpoints (systems engineering, 3D representation, verification, etc.)
- Extracting sections of the central CBIM for export in a COINS container
- Recording of receipt messages (VISI) [http://www.crow.nl/visi/](http://www.crow.nl/visi/)
- Reporting of performed modifications
- Application Programming Interface (API)
- Multi-user environment

Many of the functions stated above can be demonstrated in a software tool: COINS Navigator [http://www.coinsweb.nl/wiki/index.php/COINS_Navigator](http://www.coinsweb.nl/wiki/index.php/COINS_Navigator). This tool is a reference implementation (in addition to the documentation set) to gain a clear insight into how to apply the various COINS concepts.
Global architecture

The global architecture of a COINS compatible Building Information System (CBIS) is depicted in figure 1. The CBIS kernel manages exclusively the database containing the central BIM and its associated documents residing in a protected document store. Protected means documents are subject to version management control by CBIS exclusively. Primary input and output is organized around a VISI framework of formal transactions which are executed using standard messages and attached COINS containers. Although the deployment of VISI is not a prerequisite for the application of COINS the CBIS should be able to (let) interpret a VISI message and deal with the appropriate linking of affected objects with these VISI messages (See also: 4.2 Merging the contributions of various project partners into one central COINS Building Information Model.

Figure 1: Global architecture of a COINS-compatible Building Information System (CBIS). The blue area marks the actual CBIS, while the gray objects describe the interaction environment. The usage of VISI is not mandatory for the application of COINS. A less accurate solution could deploy e-mail messages instead.
The COINS-systematiek 1.0 - Reference Manual

(CBIM)). The COINS containers are the vehicles that transport content information between CBIS and COINS compatible discipline specific applications (e.g. Revit, Ticon, Allplan, Smarteam FS, MSProject). The CBIS-API should enable a set of dashboard applications to assist the BIM manager with a wide spectrum of inspection and control functions. In addition, report generators could use the same channel to output up-to-date general surveys over arbitrary periods and for various model views.

![Figure 2: Hub and spoke configuration with multiple CBIS's](image)

In a larger project it could occur that a project partner organizes his information management around a local CBIS (e.g. to communicate with sub-partners/departments). This situation will not change the communication protocol really. The BIM manager of the partner using a local CBIS must decide what part of the CBIS is public (i.e. should be communicated with the main CBIS) and what part is private and should only communicated with the sub-partners.

Besides the interaction with a discipline specific application a COINS dedicated application (modeler) could be useful. This modeler is not the same as the editor application on top of the CBIS. This last tool is meant for small repairs performed by the BIM manager. However, a COINS modeling tool is perfectly aligned with the complete CBIM specification instead of the indirect control from a discipline specific application which only observes a mapped model view.

An information object in a CBIM may reference an external object in a public or local object library. If this object library has a CBIM structure CBIS should be able to handle this reference (e.g. an existence check). However, if the object library has a foreign structure CBIS should accept the link as is.
COINS BIM specification

Web Ontology Language (OWL)


The OWL files that specify the COINS kernel schema and two sub-schema's for the established reference frameworks (functional specification and quantity estimation) are part of this specification.

On-line specification and documentation:


The serialization of a CBIM into a physical file takes the form of RDF/XML. This serialization format is also a W3C recommendation. A function library to ease the efforts for developing a COINS compatible interface is available as public domain software. It hides the serialization and deserialization peculiarities for the (.NET/C#) application programmer. Java programmers are recommended to make use of the JENA library [http://jena.sourceforge.net/](http://jena.sourceforge.net/).

Industry Foundation Classes (IFC)

The COINS BIM development team has decided not to add shape representation object classes to the specification schema. The considerations for this decision are various:

- it is a considerable effort
- it will take a long time to persuade the CAD vendors to implement yet another shape representation format
- there are already various standardized formats available

As a result COINS does not prescribe a shape representation format. In principle, any agreed upon format is permitted including proprietary formats. However, IFC is regarded as somewhat special because it is aimed for use in the building industry and because most CAD vendors that serve the construction market offer import and export facilities for IFC. In this way IFC can at least play the role of a lingua franca in a way that PDF fills a corresponding role for text documents, i.e. a read-only publishing format.

For this reason a CBIS implementation should at least be able to parse and render IFC models and additionally being capable to establish and maintain links between CBIM objects and IFC objects.
Plug-in reference frameworks

The COINS standard has a built-in structure to extend the central schema with sub-models (reference frameworks) that focus on specific topics for certain particular applications. At this moment two reference frameworks are specified and published (functional specification and quantity estimation) while others are in various stages of development. In addition to the reference frameworks that form part of the COINS standardization effort it is essentially the intention that development communities outside the COINS organization could specify customized reference frameworks in the scope of a project or a company or a particular branch or discipline of the building and construction industry.

A productive COINS compatible Building Information System should display enough flexibility to support this need for extendability.

CBIS functionality

A COINS compatible Building Information System maintains the actual central CBIM including the development history of the model. In principle all modifications lead to new versions of the information objects involved while preserving the preceding object versions (identifiable by the state value \textit{expired}). The structure offers the opportunity to undo modifications under exceptional circumstances returning effectively to a previous database state. This functionality amounts to the standard available facilities of general database management systems.

![Figure 3: Canonical COINS BIM development cycle](image-url)
Partners in a construction project will not operate directly on the central CBIM. COINS facilitates concurrent engineering using a structure of off-line information packages (COINS containers) which contain a subset of the central CBIM including an authorization prescription which part is open for modifications (additions, deletions) and which part is restricted to read-only access. How the participating applications this authorization prescription incorporate in their user-interface is left to the application supplier. However, the CBIS should contain features to check the result container of the contributing partner on authorization violations.

If the contribution is accepted the sub-model is merged with the central CBIM. After the merge operation the modified section will be incorporated in new versions of the affected information objects (in case of deleted objects this only leads to objects with an expired status).

The CBIS should maintain an administration of all issued transactions (expected contributions of partners) with the accompanying authorization prescriptions. In principle at any point in time only one partner should have modification rights in the same demarcated area of the central CBIM. However, enforcing this policy belongs to the responsibility of the BIM manager, e.g. by aligning the authorizations of the issued transactions. CBIS itself should not impose any restrictions in this field.

Below follows a more detailed description of the essential CBIS functions.

**Importing a COINS-container**

As a rule a COINS container contains a CBIM file with associated documents. The import of a COINS container will include at least the following tasks:

- Unpack the container
- Deserialize the CBIM file into the native CBIS storage format as a separate model
- Store the attached documents
- Update the document links of the CBIM

**Merging the contributions of various project partners into one central COINS Building Information Model (CBIM)**

Merging the contribution of a project partner (packed in a COINS container) with the central CBIM will include at least the following tasks:

- Add the accompanying VISI message to the document store.
- Scan the contribution on new, modified and deleted information objects.
  - New objects are added to the central BIM.
  - Deleted objects are marked expired.
  - Modified objects receive also expired status in the central BIM.
  - Create new versions of the modified objects with the actual attributes and links.
  - Let the previous version of a modified object point to the new version.
- Let all new, modified and deleted objects link to the accompanying VISI message.
- The merge operation should implicate the corresponding authorization prescription (Window of Authorization).
Tracing modifications

In principle an information object is never removed from the central BIM. Modifications are incorporated in a new version of the object. The previous (expired) version contains a link to the new version. Not affected objects that point to the modified object continue pointing to the original object version (This rule enables an undo of the applied modification, furthermore external links will stay valid: the next-version relations build up a linked list ultimately pointing to the actual object version.).

Each object may contain meta-data with among others the creation date and the last modification date. This offers the opportunity to compile a changes report over an established period of time.

Keeping up with the consecutive versions of information objects

The central CBIM is able to contain all versions of the same information object. A newly created object is automatically assigned version 0. Subsequent versions increment this number with 1. To give each object version a unique ID this version number is post-fixed to the ID.

Keeping up with baselines

Information objects can be assigned to a baseline. A baseline is either in the status open or closed. Objects belonging to a closed baseline should be protected to unauthorized modifications.

Seeing to that only designated persons (using explicit authorizations) are allowed to make changes to certain information objects

An authorization prescription (Window of Authorization of WoA) describes which part of the central BIM are open for modifications (write access), which part is open for inspection (read-only access) and which parts remain invisible (no access). The COINS container that is compiled for a specific WoA will not include the no-access part. During the merge operation of the result contribution the original WoA will tell if the authorization prescription is violated or not, i.e. if modifications are encountered in the read-only section.

Browsing the CBIM from various viewpoints (systems engineering, 3D representation, verification, etc.)

Obviously the CBIS must support browsing functionality. In addition support for graphical inspection of the associated 3D representation is essential. A click-able 3D window should select the associated CBIM objects. Conversely selecting a CBIM object should single out associated 3D objects (if such a link exists of course).

Initially this functionality is required for 3D objects represented according the IFC format.
Extracting sections of the central BIM for export in a COINS container

Creating a COINS container as part of a transaction in a construction project will include at least the following tasks:

- An authorization prescription is determined (WoA) with the various sections for write access, read access and no-access.
- Under control of this WoA a sub-BIM is compiled with only actual (valid) versions of the objects in the write access and read access sections.
- The sub-BIM is places in a ZIP file (COINS container) in the top-level folder `bim`.
- Relevant documents (to be decided by the BIM manager) are placed in the top-level folder `doc` (either as file copy or as web link).
- The WoA itself (an XML file) is placed in top level folder `woa`.

Recording of receipt messages (VISI)

VISI messages that accompany the partner contributions are added to the document store of the CBIS. All information objects which during the merge operation are assigned the `expired` status or are newly created object versions will include a link that points to this accompanying message.

Reporting of performed modifications

The CBIS should be able to generate changes reports of arbitrary periods of time.

Application Programming Interface (API)

The CBIS should implement an application programming interface specification to enable third parties to interface with the functionality of CBIS.

Multi-user environment

The CBIS should facilitate a multi-user environment. More than one user should be allowed to access the information stored and managed by CBIS either by API, webservices, web page or any other means of access.

Appendix A: Terms and definitions

Baseline. Documentation/description of a product in a defined phase of development. A baseline can be frozen and serve as a stable basis for next development phases.
Building Information Modeling (BIM). Is the process of generating and managing building data during its life cycle. The process produces the Building Information Model (also abbreviated BIM), which encompasses functional and physical characteristics, building geometry, spatial relationships, geographic information, and quantities and properties of building parts. The Building Information Model forms the basis for specification, verification, construction, procurement, logistics, implementation, quality assurance, etc.

Building Part. A physical part of a building. A building part can be a assembly of other building parts. A building part is also referred to as an object.

CBIM. COINS Building Information Model (CBIM) is a building information model that complies with the COINS-standard.

CBIM schema. The schema that specifies the structure of a CBIM.

CBIS. COINS-compatible Building Information System.

CEM. COINS Engineering Method (CEM) is a collection of agreements with respect to the handling of building information. CEM is a part of the COINS-standard.

COINS-standard. A collection of agreements with respect to the structure of building information and the handling of building information which serves as a backbone for integration of the building process.

COINS-Reference framework. An add-on to the COINS-standard on the purpose of serving a certain application domain. A framework may have significance in national, organizational or project level.

COINS-implementation guideline. Guidelines which are additionally to the COINS-standard and are prepared to serve IT-implementation and to support interoperability between IT-systems.

Requirement. Description of a desired property of a function, product or service.

Function. A task, action, or activity that must be performed to achieve a desired outcome.

Function fulfiller. A building part or space that fulfills one of more functions.

Functional analysis. The purpose of this system engineering process activity is to transform the functional, performance, interface and other requirements that were identified through requirements analysis into a coherent description of system functions that can be used to guide the Design Synthesis activity that follows.

Functional requirement. The necessary task, action, or activity that must be accomplished.

Status. A building information model is populated by many information objects. Through the established status, the meaning is determined of a given information object for the establishment of a building. The status is determined by three characteristics that occur each information item, namely: Release Status, Baseline Status and Validity Status.

Systems Engineering. A collection of related methods for a controlled inter-disciplinary approach to the development of systems and products using a life cycle approach which is focused to a reach a solution that meets the customer requirements.

Stage. A building part goes through a product life cycle, from design to demolition. Where a building component is in the life-cycle is determined by the Stage (is also called life-cycle stage).

Validation. The effort during which the technical data product is tested for technical adequacy, accuracy, and compliance with the provisions of the specifications and other contractual
requirements. Validation is accomplished by comparing the data product with the actual systems or equipment for which the data product was prepared.

**Verification.** Demonstrate through research and provision of objective evidence that specified requirements have been met. [EN-ISO 9000:2005].

**Verification Method.** A method to verify or demonstrate that a design or realized building part meets each of the requirements that are established and recorded during requirements analysis and functional allocation activities. (If it cannot be verified, it cannot be a legitimate requirement.)

**Verification Matrix.** A matrix in which the link is established between the requirements, the function full-filler and verification method.

**Verification Note.** The verification note is an overview of verifications, tests and inspections throughout the project. Where appropriate selections can be made by a building part or stage.

**Verification Plan.** A plan setting out how and when the verification of a part should be executed. The core of the plan concerns the verification method.

**Verification Report.** The registration of a separate verifications. If necessary reference is made to a supporting document.

**Version** A given information object can appear more than once in a building information system. The various instances are distinguished by a version number.

**Virtual Building.** Is a synonym of Building Information Model and Building Information Modelling.

**Request Specification.** The agreement as such identified in the contract document, which is prepared by or on behalf of the client, under which the contractor has prepared and submitted his offer. [UAVGC 2005].

**Request Specification Part I.** Contract Document in which the requirements to the (partial) product are expressed (the WHAT-requirements). **Request Specification Part II.** Contract Document in which the demands on the work to be performed (= Statement of Work) are expressed (the HOW-requirements).

**Window of Authorization.** Authorization specification that is linked with a specific transaction request to enable model change management.
Appendix B: COINS Navigator

The COINS Navigator [http://www.coinsweb.nl/wiki/index.php/COINS_Navigator_User_Manual](http://www.coinsweb.nl/wiki/index.php/COINS_Navigator_User_Manual) (CN) is a reference implementation of COINS concepts and is able to demonstrate many functions of a COINS compatible building information system. CN does not employ a database yet it loads COINS BIM files or COINS containers entirely in memory. The user decides if the modifications will be saved or not.

CN contains a set of custom-made browsers to go through and link external object libraries (it would be nice if a standardized API to communicate with object libraries would come available).

An IFC engine plus 3D viewer are added to interpret IFC files. Both applications utilize DLL's so that CN in spite of its Java coding effectively only operates on a Windows platform.

Reports can be generated either as spreadsheet (Excel format) or as text document (HTML).
Appendix C: A very simple case

Figure 5: A very simple case: the specification, design and construction planning of a park bench.
To demonstrate some general principles of the COINS engineering method a very simple case has been developed which focuses on the specification, design and construction planning transactions to realize a bench (a COINS benchmark so to say) in a park.

Figure 6: Scenario with the transactions between the BIM manager and the various contributors.
The various activities and milestones of this case scenario have been worked out in a tutorial so that the necessary actions can be emulated hands-on with the COINS Navigator. This tutorial is documented at the following site: http://www.coinsweb.nl/wiki/index.php/Tutorial_en .
Object libraries

COINS Library

A COINS Library is a universal designation for a model that complies with the COINS standard but that can be utilized beyond a specific project scope. Typical applications are:

- **Catalogue part library**
  A set of catalogue part descriptions that can be reused in various projects.

- **Property types set**
  Set of standard property types.

- **Templates**
  Global information structure of a specific product type, e.g. bridge template, highway template, office template, etc.

- **All COINS models that are imported in a sub-model hierarchy**
  especially practical in large project (various sections of a railway project that are tendered to various contractors) or to organize variant elaborated designs.

The remainder of this article will focus on the various ways to set up a catalogue part library.

Contents

- 1 Catalogue Part Library
  - 1.1 Properties
    - 1.1.1 Amount property type
    - 1.1.2 Complex property
  - 1.2 Supertype hierarchy
  - 1.3 Assembly hierarchy

Catalogue Part Library

A catalogue part library contains a set of catalogue_part_objects which collectively may structure a type hierarchy and/or decomposition structure.
The CataloguePart class has a rich set of relation types that is able to support complex hierarchical information structures. The next paragraphs will give a walk through of the various modelling constructs.

Properties

Specifying properties is probably the most basic modelling construct for CatalogueParts. The CBIM schema distinguishes Property Types and Property Values, where a Property Value always references a Property Type.

Take for example a beam catalogue part with dimensional properties length, width and height.

Figure 1: A UML class diagram of the CataloguePart class and its direct relations with neighbouring classes.

Figure 2: A COINS Navigator screen grab of the specification of a beam catalogue part with three dimensional properties.
The COINS-systematiek 1.0 - Reference Manual

Figure 3: A SketchUp screen grab of the 3D representation of a beam catalogue part with three dimensional properties.

Figure 4: A UML instance diagram of a beam catalogue part with three dimensional properties.

Amount property type

A special kind of property is the amount property. An amount object keeps the amount of a certain catalogue part with regard to an assembly object. Such an assembly object could be a physical object node in the object tree. To compile a bill of quantities most catalogue parts will be counted piecewise. For this reason the `CataloguePart` class is a subclass of the `PropertyType` class. The amount property is in that case identical to the catalogue part itself. But in other cases it could be that the quantity of a catalogue part is measured by its length or area or volume or mass, etc. One of the properties of the catalogue part is then selected as the amount property.
Complex property

Simple properties are single valued. A complex property however may handle a set of properties. Complex property values are part of the reference framework Functional Specifying. A typical example of a complex property is material. A specific material will contain a set of standard properties (e.g. density, melting point) which are universal for all materials and a set of specific properties that only occur for a certain group of materials (e.g. yield strength). Complex property values are stored in catalogue part objects and will generally form a family using the supertype construct.
value and by selecting a so called template. The template will form the root of all selectable values of this property type.

Figure 7: Setting the template for all selectable complex values for this property type. A template further down the family tree will restrict the set of selectable values.

If the root of the family tree (here Material) is selected as template all descendant complex values in the family tree are qualified for value selection. If for example Steel is assigned as template only descendant values of Steel will appear in the selection box of selectable values.

Figure 8: Setting the actual value offers a choice between all selectable complex values which share the template as a common root.
Figure 9: Instance diagram showing the data structure of catalogue part Beam that specifies a material complex property value Concrete. The PropertyValue class is extended in the reference frame Functional Specifying with the complexValue functional property.

**Supertype hierarchy**

A Catalogue Part may reference another CataloguePart as its supertype. This modelling construct introduces (single) inheritance into a catalogue part library. The supertype semantics allow both property type as well as property value inheritance.
Figure 6: A COINS Navigator screen grab of the specification of a concrete beam catalogue part founded on a beam catalogue part as its supertype.

Figure 7: A UML instance diagram of the specification of a concrete beam catalogue part founded on a beam catalogue part as its supertype. To simplify the diagram the property value boxes plus connecting relation links are substituted by a mock link annotated with the property value.

Figure 6 and 7 show an example where a catalogue part Concrete Beam inherits from a catalogue part Beam both property types and associated property values. Because the inherited property values cannot be overwritten the Navigator represents these rows dimmed. The semantics are that the subtype catalogue part is also a supertype catalogue part (it belongs to the same category), overwriting a property value would violate this condition.

The material property is not inherited and its value may freely be chosen (although the naming of the catalogue part seems to restrict this freedom).

If the supertype catalogue part specifies a property type without giving it a value (value attribute of the property value object is empty) the semantics are that subtype catalogue parts are free to give it
a value (or not: delegating this degree of freedom further down the type hierarchy).

![Figure 8: COINS Navigator screen grabs of the specification of a concrete beam catalogue part founded on a beam catalogue part as its supertype. The length property is not valued by the supertype and can be specified by the subtype.](image)

![Figure 10: UML instance diagram of the specification of a concrete beam catalogue part founded on a beam catalogue part as its supertype. The length property is not valued by the supertype and can be specified by the subtype.](image)

This modelling construct facilitates the specification of catalogue part families, in this example a family of beams with different lengths.
Assembly hierarchy

The catalogue part data-structure facilitates also a modelling construct to build assembly hierarchies (decomposition). The basic approach is that a new catalogue part can be specified by combining already available catalogue (sub)parts. For example a portal can be composed from two column parts and a beam part.
Figure 12: UML instance diagram of a portal catalogue part that is composed from a column catalogue part (twice) and a beam catalogue part. Locators position the subparts at the right location and orientation.

Figure 13: SketchUp representation of multiple instances of an assembly catalogue part
CBIM Schema

http://www.coinsweb.nl/c-bim.owl

Classes

- Amount
- Baseline
- Building
- C-BIM Object
- Catalogue Part
- Connection
- Document
- Explicit 3D Representation
- Function
- Function Fulfiller
- Library Reference
- Locator
- Parameter
- Performance
- Person or Organisation
- Physical Object
- Property Type
- Property Value
- Requirement
- Space
- State
- Task
- Task Type
- Terminal
- Vector
- Verification
- VISI Message

Object Properties

- affects
- amount property type
- baseline
- baseline object
- catalogue part
- child
• contains
• creator
• current state
• document
• female terminal
• first parameter
• fulfills
• is affected by
• is fulfilled by
• is situated in
• locator
• male terminal
• max bounding box
• min bounding box
• modifier
• next parameter
• next version
• parent
• performance
• performance of
• physical child
• physical parent
• previous state
• primary orientation
• property type
• property value
• requirement
• requirement of
• secondary orientation
• shape
• situates
• spatial child
• spatial parent
• state of
• supertype
• task type
• terminal
• terminal of
• translation
• verification function fulfiller
• verification performer
• verification requirement
Datatype Properties

- baseline status
- change log
- creation date
- default value
- description
- document alias file path
- document type
- document URI
- end date
- end date actual
- end date planned
- expired
- layer index
- modification date
- name
- release date
- release status
- start date
- start date actual
- start date planned
- unit
- user ID
- value
- value domain
- verification date
- verification method
- verification result
- x coordinate
- y coordinate
- z coordinate