

Document id	Title	Organisation / Author (mvdXML1.2)	Date	Status
P9-002 ###	mvdXML specification 1.2	Matthias Weise, Andreas Geiger, Sergej Muhic	2021-07-02	Draft

mvdXML

Specification of a standardized format to define and exchange
Model View Definitions with Exchange Requirements and Validation Rules

Developed by
Technical Services of buildingSMART International Ltd.

Authors
Chipman, Tim; Liebich, Thomas; Weise, Matthias;
Geiger, Andreas; Muhic, Sergej

Version 1.2 (Draft)

23. 09. 2021

Document history

Version	Change description	Date
1.2 (Draft)	<p>Schema changes (a more detailed change log is provided in the XSD)</p> <ul style="list-style-type: none"> ▪ update <i>namespace</i> for mvdXML to match with new buildingSMART URL for technical specifications ▪ update IFC <i>schemaName</i> for available and future IFC releases following the new schema name pattern ▪ <i>RuleMessages</i> (user defined messages) added to <i>TemplateRule</i>, <i>TemplateRules</i>, <i>Constraint</i> and <i>Applicability</i> ▪ <i>ModelView</i>: Views added to enable nesting of model views ▪ <i>Precision</i> setting added for comparison of floating-point numbers (globally on <i>DefaultSettings</i> and locally on <i>Constraints</i> and <i>TemplateRule</i>) ▪ <i>TemplateRule</i>: <i>negation</i> attribute added, <i>TemplateRules</i>: NOT removed from <i>operator</i> attribute (more consistent use of negation) ▪ <i>TemplateRule/TemplateRules:Applicability</i> added (enable to add conditions to expressions) ▪ <i>DefaultSettings</i>: new element (default settings for precision and units) ▪ <i>ConceptRoot</i>: <i>Specializations</i>, <i>Specialization</i> added (for controlling reuse and inheritance) ▪ <i>Applicability</i>: <i>minOccurrence</i>, <i>maxOccurrence</i> attribute added (for checking existence and conditional statements), ▪ various changes to min/maxOccurs, e.g. to be able to use multiple <i>ConceptTemplates</i> in a <i>Concept</i> configuration ▪ <i>DefaultSettings</i> added (enable to integrate ifcXML-based unit settings) ▪ use of <i>GenericReference</i> harmonized and extended <p>A few features from an early draft are still included in this draft, but may not be included in the final version. This includes <i>TemplateRule.Requirements</i>, <i>TemplateRule.Order</i> and <i>TemplateRule.Usage</i> added.</p> <p>Grammar changes</p> <ul style="list-style-type: none"> ▪ introduce revised version based on adjustments for better compatibility for ANTLR parser ▪ add negation (NOT) for boolean terms ▪ add parameter SELF <p>Documentation</p> <ul style="list-style-type: none"> ▪ add introduction with overview about motivation and changes between mvdXML 1.1 and 1.2 ▪ extended examples sections for explaining new features and changes ▪ fully updated documentation section 	2021-07-02
1.1 Add 1	<p>Schema changes</p> <ul style="list-style-type: none"> ▪ <i>concept.TemplateRules</i> changed from mandatory to optional - It is allowed to have concepts without further configuration via template rules. In that case it is expected that the referenced concept template validates to true. <p>Documentation</p> <ul style="list-style-type: none"> ▪ documentation updated and improved 	2016-07-15
1.1 Final	<p>Schema extensions</p> <ul style="list-style-type: none"> ▪ <i>namespace</i> updated to: http://buildingsmart-tech.org/mvd/XML/1.1 ▪ <i>RuleId</i> - new simple type to restrict <i>EntityRule.Reference</i>, <i>EntityRule.@IdPrefix</i>, <i>EntityRule.@RuleId</i> and <i>AttributeRule.@RuleId</i> 	2016-02-15

Version	Change description	Date
	<ul style="list-style-type: none"> ▪ <i>EntityRule.Reference. @IdPrefix</i> - changed to <i>RuleId</i> (was <i>normalizedString</i>) ▪ <i>EntityRule. @RuleId</i> - changed to <i>RuleID</i> (was <i>normalizedString</i>) ▪ <i>AttributeRule. @RuleId</i> - changed to <i>RuleID</i> (was <i>normalizedString</i>) ▪ <i>Copyright</i> – changed to <i>normalizedString</i> (was <i>anyURI</i>) <p>Documentation</p> <ul style="list-style-type: none"> ▪ examples updated to the latest grammar ▪ documentation updated and improved 	
1.1 RC	<p>Schema extensions</p> <ul style="list-style-type: none"> ▪ <i>EntityRule.References.Template</i> - new element that allows to reference other templates as partial templates, it allows to reuse common, smaller <i>ConceptTemplate</i> definitions ▪ <i>EntityRule.References.Template. @ref</i> - reference to the partial template by uuid ▪ <i>EntityRule.References. @IdPrefix</i> - an optional prefix for the <i>RuleId</i> name, used to prevent ambiguous <i>RuleId</i>, if the same partial template is referenced twice in a concept template tree ▪ <i>Concept.TemplateRules</i> - new element and tree structure to define a logical tree (with Boolean operators) to combine several template rules ▪ <i>ConceptRoot.Applicability</i> - new element to check, whether the instance of the <i>applicableRootEntity</i> is applicable, allows for more conditions (like certain property values) ▪ <i>ConceptTemplate. @applicableSchema</i> - defined as a list of extensible enumeration of standard IFC schema identifiers, or any other schema name. ▪ <i>ModelView. @applicableSchema</i> – defined as a single string, being an extensible enumeration of standard IFC schema identifiers, or any other schema name ▪ <i>TemplateRules</i> – new element that is declared in a recursive way, allowing other <i>TemplateRules</i>, or individual <i>TemplateRule</i> as child elements. It allows to establish a Boolean tree, where at each <i>TemplateRules</i> a logical operator is defined, ▪ <i>TemplateRules. @operator</i> – new attribute that defines the logical operator to combine the logic results of its children, ▪ <i>Requirement. @requirement</i> – enhancement of the enumerators to include recommended, not-relevant (was “not relevant” and “optional”) and not-recommended. <p>Schema changes - strict version: removed, transitional version: deprecated</p> <ul style="list-style-type: none"> ▪ <i>AbstractRule</i> - abstract element and <i>complexType</i> removed, attributes moved to <i>AttributeRule</i> and <i>EntityRule</i> ▪ <i>ConceptTemplate.Rules</i> - restricted to <i>AttributeRule</i>, was an agreement in V1.0, now enforced by schema ▪ <i>AttributeRule. @Cardinality</i> - removed: this attribute shall not be used to impose a restriction on the cardinality, restrictions are all handled by template rules ▪ <i>EntityRule. @Cardinality</i> - removed: this attribute shall not be used to impose a restriction on the cardinality, restrictions are all handled by template rules ▪ <i>EntityRule.EntityRules</i> - removed: There is no usage for an <i>EntityRule</i> to directly contain other <i>EntityRules</i>, without an intermediate <i>AttributeRules</i> ▪ <i>ConceptRoot.Requirements</i> - removed: requirements are only valid for concepts, not for a root concept ▪ <i>Concept.Definition</i> – unified the whole schema to have <i>Definition</i> being always the first element in a sequence ▪ <i>Concept.Rules</i> - removed: <i>AttributeRules</i> and <i>EntityRules</i> at this level are not legal, replaces by <i>TemplateRules</i> that only allow <i>TemplateRule</i>, and Boolean logic, the old <i>Rules</i> structure did not allow for logical combinations of individual rules, other than by the implied AND combination 	2015-08-18

Version	Change description	Date
	<ul style="list-style-type: none"> ▪ <i>Concept.SubConcepts</i> - removed/deprecated: the inclusion of <i>SubConcepts</i> has no functionality so far, in order to reduce complexity it should not be used, concepts should be flat ▪ <i>Cardinality</i> - simple type removed/deprecated, not used any more in <i>AttributeRule</i> or <i>EntityRule</i> ▪ <i>ModelView.Definition</i> – unified the whole schema to have <i>Definition</i> being always the first element in a sequence 	
1.1 beta	<p>First revision of mvdXML with following corrections, changes and clarifications:</p> <p>Schema extensions:</p> <ul style="list-style-type: none"> ▪ cardinality attribute of <i>AttributeRule</i> and <i>EntityRule</i> extended to support definition of any min and max settings ▪ <i>BaseConcept</i> and <i>Override</i> attribute added to <i>Concept</i> ▪ tags attribute added to <i>Definitions</i> <p>Rule grammar:</p> <ul style="list-style-type: none"> ▪ mvdXML 1.1 beta provides a grammar for defining constraints to simplify rule parsing and to enable logical “or” combination of rules <p>Schema improvements:</p> <ul style="list-style-type: none"> ▪ new complex type <i>GenericReference</i> (used by <i>ModelView</i> and <i>Concept</i>) ▪ simplified definition of <i>EntityRule</i> and <i>TemplateRule</i> ▪ definition of applicability attribute changed for <i>ExchangeRequirement</i> and <i>Requirement</i> ▪ <i>minOccurs</i> changed from 1 to 0 for <i>ModelView.Roots</i> and <i>mvdXML.Templates</i> ▪ <i>maxOccurs</i> added to several definitions, mainly for clarification ▪ definition and use of applicability (was <i>xs:attribute</i> is now <i>xs:simpleType</i>) ▪ <i>ConceptTemplate.applicableSchema</i> changed to a list of String types ▪ <i>ConceptRoot.applicableRootEntity</i> now mandatory <p>Improved and extended documentation:</p> <ul style="list-style-type: none"> ▪ Use of sub-templates and sub-concepts clarified ▪ Several improvements and corrections 	2013-11-01
1.0	<p>Final release of mvdXML. Accepted by bSI ITM committee as the official buildingSMART specification for publishing Model View Definitions</p> <ul style="list-style-type: none"> ▪ NOTE This release does not yet focus on model validation 	2012-05-14
0.9.4	<p>The following changes were made in this draft:</p> <ul style="list-style-type: none"> ▪ <i>EntityRule.EntityRules</i> added for indicating subtype rules. 	2012-05-11
0.9.3	<p>The following changes were made in this draft:</p> <ul style="list-style-type: none"> ▪ <i>ModelView.BaseView</i> added for indicating add-on views. ▪ <i>ExchangeRequirement.applicability</i> attribute added. ▪ <i>ConceptTemplate.ApplicableEntities</i> renamed to <i>ApplicableEntity</i>. 	2012-05-07
0.9.2	<p>The following changes were made in this draft:</p> <ul style="list-style-type: none"> ▪ <i>ConceptLeafNode</i> was renamed to <i>Concept</i>, with <i>SubConcepts</i> added. ▪ <i>ApplicableSchema</i> attributes use string instead of enumeration for version flexibility. ▪ <i>Cardinality</i> includes “_asSchema” to indicate default cardinality. 	2012-04-20
0.9.1	<p>Combined mvdXML schema proposal incorporating the original mvdXML 0.5 with the proposed phase 2 extension, several simplifications:</p> <ul style="list-style-type: none"> ▪ The <i>ConceptNode</i> entity was deleted. ▪ The <i>Concept</i> abstract entity was deleted, since <i>ConceptNode</i> was deleted, where the only attribute was moved to the subtype <i>ConceptLeafNode</i>. 	2012-03-27

Version	Change description	Date
	<ul style="list-style-type: none"> The <i>ConceptRootNode</i> Category attribute was deleted 	
0.8	Proposal for phase 2 of a formal mvdXML format: <i>Mapping of MVD concepts to IFC definitions</i> as appendix to mvdXML 0.5	2011-05-20
0.5	First buildingSMART release, no other changes of content	2011-06-19
0.4	Public release, first release after acceptance of mvdXML by buildingSMART ITM group, following changes have been made: Incorporation of the formally defined (IFC) schema, that describes the formal subschema corresponding to the Model View Definition. Minor changes as result of first prototype developments.	2011-05-05
0.3	Public release, incorporating feedback from buildingSMART MSG Restructuring of document content, adding MVD history. Adding general objectives, motivation and relation to MVD methodology. Minor corrections in XSD Version 0.3 (key/keyref and href for Definition).	2011-03-04
0.2	Restricted release to buildingSMART MSG and TechCom, XSD Version 0.2	2011-02-16
0.1	Internal release, XSD version 0.1	2011-02-07

Table of Content

0	Foreword.....	9
1	Overview	11
1.1	Purpose	11
1.2	Methodology	11
1.3	Usage	12
2	Schema.....	13
2.1	Quick overview	13
2.2	Simple example.....	18
3	Description of mvdXML schema elements and types.....	23
3.1	mvdXML	23
3.2	Concept Template.....	23
3.2.1	Attribute Rule	24
3.2.2	Entity Rule.....	25
3.2.3	Constraint	26
3.3	Model View	26
3.3.1	Exchange Requirement	27
3.3.2	Concept Root	28
3.3.3	Applicability	29
3.3.4	Concept.....	29
3.3.5	Requirement	30
3.3.6	TemplateRules	31
3.3.7	TemplateRule.....	32
3.3.8	RuleMessage	34
3.4	Common type and attribute definitions	34
3.4.1	Identity.....	34
3.4.2	Definition	35
3.4.3	Body	35
3.4.4	Link.....	36
3.4.5	GenericReference	36
4	Rule Grammar	38
5	mvdXML Use Cases	44
5.1	MVD Documentation	44
5.2	Specification of subset schemas	44
5.3	Data Filtering	45
5.4	Data Validation	45
6	Glossary	46
6.1	ER	46
6.2	ERM.....	46
6.3	MVD	46
7	Examples	47

Page no.	Authors
6	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

7.1	Example for MVD documentation	47
7.2	Examples for MVD validation.....	53
7.2.1	Checking IFC Properties on load bearing, external walls	54
7.2.2	Checking existance of objects with error messages	61
7.2.3	Use of SELF expression for defining exception from list of subtypes	61
7.2.4	Combination of ConceptTemplate.....	62
7.2.5	Check existence of a particular information for all material layers	63
7.2.6	Adding information about precision and units	65
8	XSD Listing.....	67

Page no.	Authors
7	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

Table of Figures

Figure 1: Basic structure of the mvdXML schema	13
Figure 2: main mvdXML elements for <i>ConceptTemplate</i>	14
Figure 3: main mvdXML elements for <i>ModelView</i>	17
Figure 4: HTML documentation of IFC4	44
Figure 5: Graphical representation of the concept template “Nesting”	47
Figure 6: graphical representation of the concept template “port nesting”	48
Figure 7: rendering of HTML tables to document the exchange requirements for the different ports	48
Figure 8: graphical representation of the usage of concept “port-nesting” at the root concept	
IfcHeatExchanger.....	49

Table of Tables

Table 1: Common element references defined in the element <i>mvdXML</i>	23
Table 2: Common attributes and element references defined in the element <i>ConceptTemplate</i>	24
Table 3: Common attributes and element references defined in the element <i>AttributeRule</i>	25
Table 4: Common attributes and element references defined in the element <i>EntityRule</i>	26
Table 5: Common attributes defined in the element <i>Constraint</i>	26
Table 6: Common attributes and element references defined in the element <i>ModelView</i>	27
Table 7: Common attributes defined in the element <i>ExchangeRequirement</i>	28
Table 8: Common attributes and element references defined in the element <i>ConceptRoot</i>	28
Table 9: Common element references defined in the element <i>Concept</i>	30
Table 10: Common element references defined in the element <i>Requirement</i>	31
Table 11: Common attributes defined in the element <i>TemplateRules</i>	32
Table 12: Truth table for operator attribute	32
Table 13: Common attributes defined in the element <i>TemplateRule</i>	33
Table 14: Common attributes defined in the attributeGroup <i>identity</i>	35
Table 15: Common element references defined in the element <i>Definition</i>	35
Table 16: Common attributes defined in the element <i>Body</i>	36
Table 17: Common attributes defined in the element <i>Link</i>	36
Table 18: Common attributes defined in the complex type <i>GenericReference</i>	37
Table 19: Description of <i>metric</i> values	43
Table 20: Description of <i>operators</i>	43
Table 21: Description of <i>operators</i> that can be applied to different data types.	43

0 Foreword

This document describes an update of mvdXML 1.1 based on experiences coming from the use of mvdXML for the IFC documentation, software certification and check of exchange requirements. It is a minor update with few changes to the schema and the rule grammar. Improvements are addressing the following points:

- Improve the check of floating-point numbers and measurements by allowing to define precision settings and to specify the unit of the given value. An example is given in chapter 7.2.6.
- Enable to check existence of objects and not only the data that is attached to objects. This feature enables to state that a particular instance or object type must be included once or more in the IFC data set. An example is given in chapter 7.2.2.
- An important design principle of mvdXML is the use of templates describing a particular fundamental IFC functionality. Such templates are later configured to reflect implementer agreements or to encode specific model checks. It is an essential part of the IFC documentation but also of the model checking. So far, the use was limited to the configuration of one template only so that any new combination of fundamental IFC functionalities was leading to the definition of a new template. With the new version it is now possible to combine different templates to avoid the need to create new templates. This enables to remove all “overriding” templates, which were necessary to check alternatives. It also offers the possibility to further standardize basic IFC features to be published as configurable ConceptTemplates. An example is given in chapter 7.2.4.
- Enable to add error messages to checking rules. Such error messages are multilingual and will improve feedback in case of failed model checks. An example is given in chapter 7.2.2.
- Check of exchange requirements are mainly testing that some information exists for a given object type. For instance, a property called “FireRating” must exist within the set of properties. Such global existence check does not allow to express existence for each element within a set of objects. Check of material properties for each layer of a wall is a practical example where such global existence check will fail. An example for the extended functionality is given in chapter 7.2.5.

Checking rules are partially encoded in a string data type. Since mvdXML 1.1 those definitions have been formalized by using a grammar to simplify implementation with tools like ANTLR. Changes to the rule grammar include the following improvements:

- Updated of the definition to be compatible with newer ANTLR parser toolboxes. More information is given in chapter 4.
- More consistent use of negation by adding the unary operator NOT for boolean terms.
- SELF parameter added that enables to check the type of the selected instance. An example is given in chapter 7.2.3

This update of mvdXML is not in contrast to the official roadmap and new standards published by buildingSMART. It is the continuation of work that is necessary to finalize specification work on IFC 4.3 and to prepare later software certification.

Page no.	Authors
9	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

Acknowledgement

This work is a collaborate effort from different authors who have been supported by the following projects and organisations:

- The software certification team from buildingSMART being a main driver for change requests.
- BIM4REN project also dealing with model quality issues, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 820773.
- "Infrastructure Extensions Deployment" project from buildingSMART (part of Infra Room) working on the new IFC4.3 specification based on mvdXML 1.2 – in collaboration with the IfcRail phase 2 project from the Rail Room.
- Karlsruhe Institute of Technology (KIT) within the projects IBPSA Project1 - an international project conducted under the umbrella of the International Building Performance Simulation Association (IBPSA) and the SDaC research project (Smart Design and Construction) funded by the Federal Ministry of Economics and Technology (BMWf).
- Siemens Real Estate work associated with the Twingine project aiming at assuring the model quality for CAPEX (capital expenditure) and OPEX (operating expense) use cases.

We kindly thank for supporting and partially funding this work.

1 Overview

A Model View Definition (MVD) describes the subset of a data schema that is required to exchange the data required in specific data exchange scenarios. An Exchange Requirement (ER) defines the required population (the data that is provided in an exchange file) of such a sub schema.

The buildingSMART standard mvdXML refers to an electronic format for representing such Model View Definitions and associated Exchange requirements. The purpose of this document is to describe the structure and usage of mvdXML.

While a data schema describes available data structures and type information, a model view definition describes graphs of such data structures to be used in scenarios with particular constraints. While mvdXML is a generic structure that could be applied to any data schema, the primary intention and documentation herein describes its use relative to the Industry Foundation Classes (IFC) data schema, the ISO16739 standard.

1.1 Purpose

The mvdXML format serves several purposes:

- To define the sub schema for the MVD, based on the base schema of IFC
- To support automated validation of IFC data sets for quality assurance and software certification.
- To generate documentation for specific model views and the IFC specification itself.
- To support software vendors providing filtering of IFC data based on model views.
- To limit the scope of IFC to well-defined subsets applicable for specific applications.

NOTE: If mvdXML shall be used for one specific purpose only not all features of this specification might be of interest. For instance, data filtering and data validation not necessarily require detailed end-user documentation or any meta-data like status, owner etc. Accordingly, depending on the main use case a subset of mvdXML might be sufficient to cover required functionality. More details about suggested use cases and evaluation of mvdXML are discussed in chapter 5.

1.2 Methodology

The underlying methodology of mvdXML is the definition of concept templates and concepts.

- A Concept Template is a graph that starts with a root entity and consists of attribute and other entity definition, all are required to represent a functional unit required to exchange specific data
 - An example is the concept template “property sets for objects”, that describes the graph, starting at the applicable supertype *IfcObject*, and describing the graph down to the assigned *IfcPropertySet* and further to the individual properties, such as *IfcPropertySingleValue*.
 - The official IFC specification lists within its chapter 4 “fundamental concepts and assumptions” those concept templates already defined. Developers of Model View Definitions are encouraged to use these concept templates but may enhance the existing or define new ones.
- A Concept is the reference to such a concept template for each entity (as subtypes of the applicable root entity of the concept template) and describes the constraints and usages within the scope of the entity.
 - An example is the definition of all applicable property sets, such as *Pset_BeamCommon* for the entity *IfcBeam* as a particular usage of the concept template “property sets for objects”.

A model view indirectly describes the set of IFC entities within scope based on entities referenced from each root concept, and entities or types (and subtypes as defined) used within instance graphs. All such IFC entities may be combined and published as the IFC schema subset corresponding to the model view. The IFC schema subset must be a valid schema by itself.

1.3 Usage

While it is still being possible to write an mvdXML based Model View Definition by using any text editor, it is anticipated that specific software applications are used to read and write mvdXML data sets. Software for working with mvdXML may include the following.

- The IFC Documentation Generator (ifcDoc) is a free tool issued by buildingSMART that reads and writes mvdXML and provides a graphical user interface for defining all content within mvdXML. It can be preloaded with a particular IFC release specification and allows access to all parts of the IFC specification when developing the mvdXML concepts and constraints. This tool may also auto-generate instantiation diagrams, output HTML documentation for model views, and is also used for generating the IFC4 documentation.
- XML/XSD editors such as Microsoft Visual Studio and Eclipse may edit mvdXML in raw format, just as any other XSD-based schema.
- Testing servers may read mvdXML and use such information to validate submitted IFC files for conformance.
- IFC-based software applications may read mvdXML for automatically filtering and validating data to conform to the specified constraints. It is also possible for IFC-based software applications to write mvdXML to enable users to define custom exchange scenarios.
- Requirement management tools may support configuration of data exchange requirements that, if based on existing mvdXML snippets, could be exported as an mvdXML document to be used for filtering and validating IFC data.

2 Schema

The overall mvdXML Schema is introduced with a quick overview and by a simple example first. The next chapter 3 is the reference for all XSD elements and types.

2.1 Quick overview

An mvdXML document contains an instance of *mvd:mvdXML* as the only single valid root element. The *mvdXML* element defines three main sub elements:

- *mvd:DefaultSettings* (new since mvdXML 1.2): Global settings about precision and units that apply to the whole mvdXML file unless overridden by rule specific settings.
- *mvd:Templates*: a list of reusable concept templates, *mvd:ConceptTemplate*, that define the graph within the base IFC schema representing the entities and attributes needed to support the functional unit addressed by the concept
- *mdv:Views*: a list of model view definitions, *mvd:ModelView* that contains the necessary entities and associated concepts to define the sub schema of the base schema to support the exchange requirements.

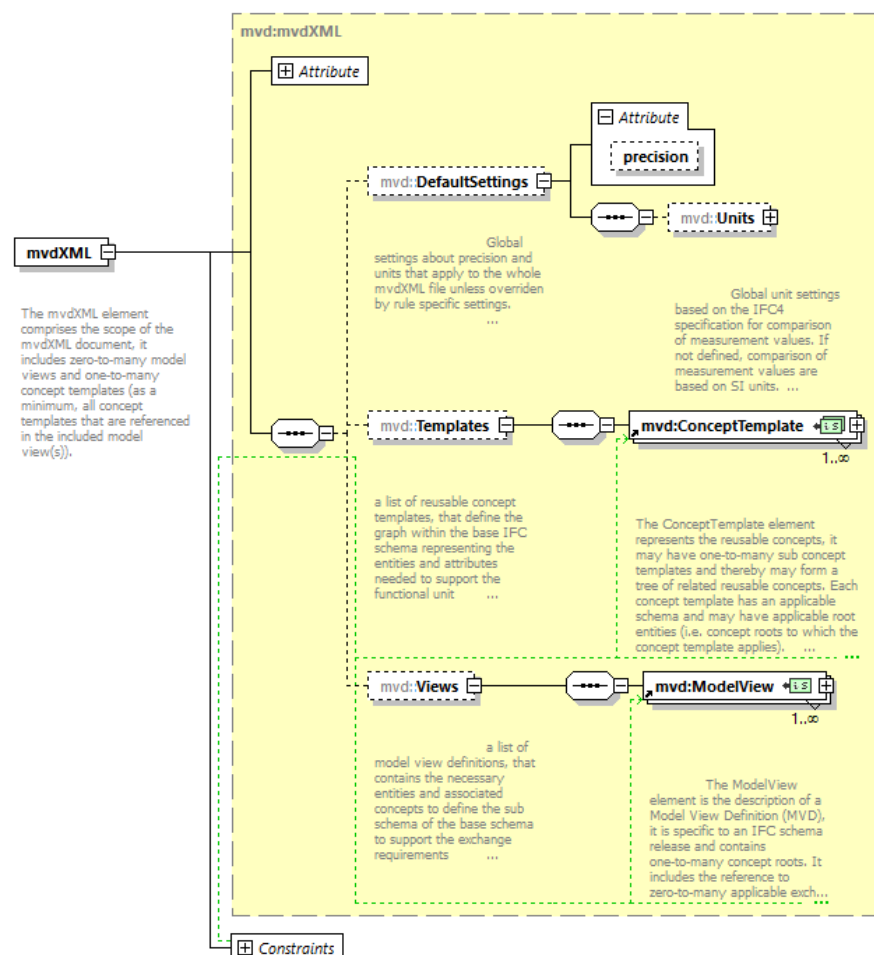


Figure 1: Basic structure of the mvdXML schema

An *mvd:ConceptTemplate* defines the graph, starting from an applicable root entity, following attribute and entity links, down to the individual attributes, which contains all schema information for a particular unit of functionality – or “concept template”, the term used within the Model View Definition methodology.

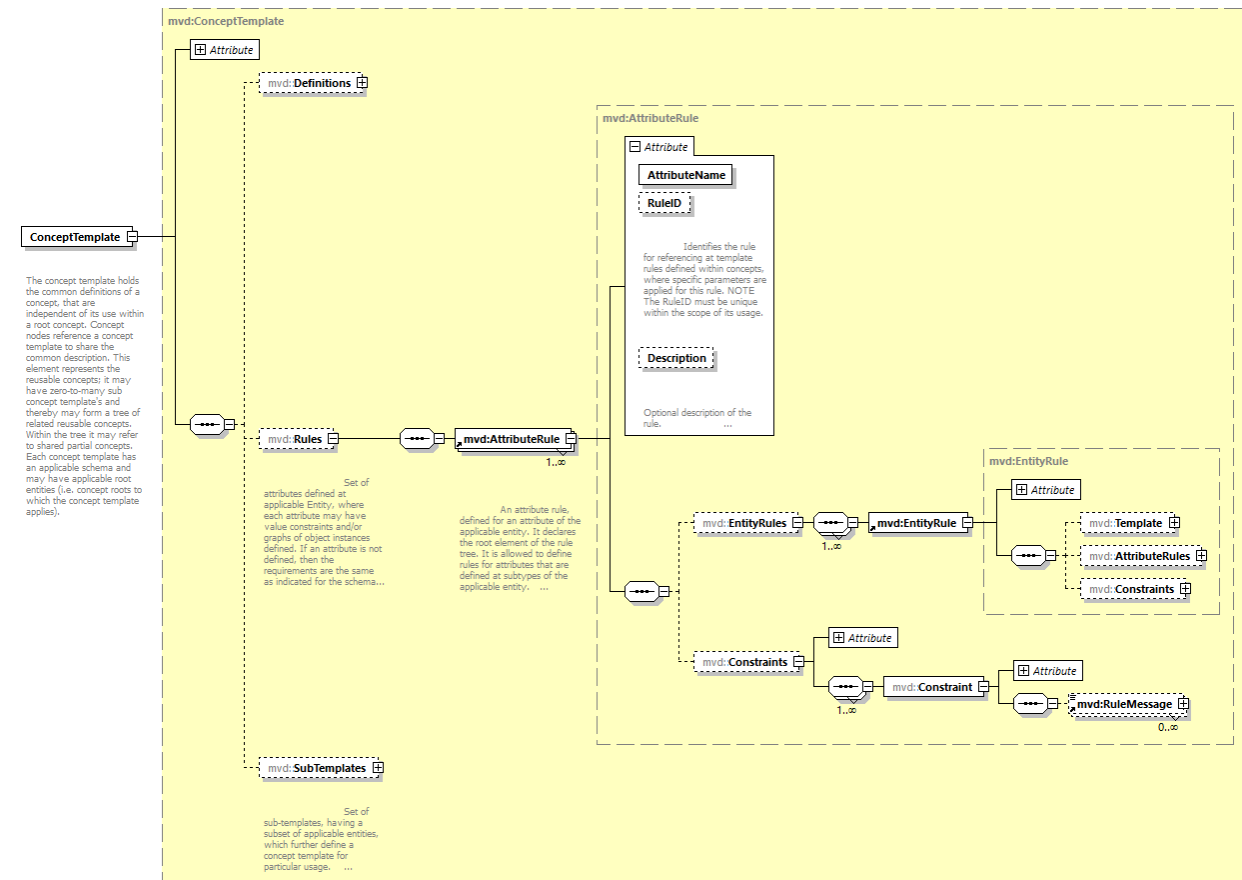


Figure 2: main mvdXML elements for ConceptTemplate

EXAMPLE: The attachment of property sets with specific properties to an element is the unit of functionality that is described by the concept template “Property Sets for Objects”¹. The definition of an assembly structure, where the assembly, such as an elemented wall has building element parts is another example that is described by “Element Decomposition”².

Each *mvd:ConceptTemplate* starts with the applicable entity, the root of this unit of functionality. In most cases, it is a subtype of *IfcObject*, being an occurrence of a model element subject to validation.

Then the attribute(s) used for expressing the unit of functionality are declared, then the type of the attribute, in case of an entity type, it can have own attribute definitions again. All together it defines a tree structure that describes the portion of the IFC schema needed for this unit of functionality. The *ConceptTemplate* element contains:

- *@applicableSchema*, such as IFC2X3 or IFC4,

¹ See: https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/schema/templates/property-sets-for-objects.htm

² See: https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/schema/templates/element-decomposition.htm

- *@applicableEntity*, the root entity of the concept template, often *IfcObject*, or a subtype, like *IfcProduct*, or *IfcGroup*, *IfcProject* and other high-level entities, deriving from *IfcRoot*
 - NOTE: In case of partial concept templates, which are reused at many concept templates, may have root entities that do not derive from *IfcRoot*. An example is the partial concept template of swept solid geometry with a root element *IfcSweptAreaSolid*, which is reused in several other concept templates describing element shape representation
- *Definitions*, a general element to include potentially multilingual documentation and links to figures, diagrams, examples and other external documents. It is mainly used for the mvdXML purpose of generating MVD documentation
- *Rules*, a list of attribute definitions, being direct attribute or relationships of the root entity, or attributes defined at the level of its subtypes, that are part of the concept template tree
 - NOTE: In many cases, the inverse attribute is used here to navigate to relationship entities
 - EXAMPLE: a common example of an attribute rule that relate to an attribute that is only defined at a subtype level is *PredefinedType*.
- *SubTemplates*, a concept template that extends the definition of the main concept, it is used to group related concept templates, e.g. all concept templates that relate to element geometry may have a common parent concept template "Product Geometric Representation", and then extent to box geometry, foot print geometry and body geometry.
 - NOTE: If a template with subtemplates is used in an exchange requirement, then the applicableEntity decides which template is used for model checking.
 - EXAMPLE: A template is defined for *IfcSimpleProperty* with subtemplates for *IfcPropertySingleValue* and *IfcPropertyEnumeratedValue*. It is referenced by a Concept for checking properties. If an instance of *IfcPropertySingleValue* is to be checked, then the template with best matching applicableEntity is selected from the subtemplates.

The tree structure of the *Rules* section at *ConceptTemplate* consists of *AttributeRules*, referring to *EntityRule*, referring to *AttributeRules*, and so on. Each *AttributeRule* has:

- an *@AttributeName*, the name of the attribute, relationship or inverse relationship in the IFC schema
- an *@RuleID*, if present, it defines an ID which is used in the model view definition to document specific usage for particular entities, or to validate its values according to exchange requirements,
- a *Constraints*, a list of *Constraint* on the schema population, if used for this concept template
 - EXAMPLE: a concept template for swept solid geometry would enforce, that the value of *RepresentationType* of entity *IfcShapeRepresentation* is always "SweptSolid", independently of its usage in a model view definition later. This can be encoded as a Constraint. Similarly, the cardinality of sets or lists might be constrained.
 - NOTE: The Constraints are a way to flexibly enhance the WHERE rules within the EXPRESS definition of IFC, and to add such rules, when using ifcXML (that cannot include such WHERE rules).
- an *EntityRules* element, containing a list of *EntityRule*, relating to the underlying type of the attribute

An *EntityRule* refers to an entity, an enumeration, a derived or simple type (based on the EXPRESS definition of IFC). Each *EntityRule* has:

- an *@EntityName*, the name of the underlying type
 - NOTE: it shall not be a SELECT type, those have to be expanded to the selected types
- an *@RuleID*, see above
- a *Constraints*, see above
- an *AttributeRules* element, containing a list of *AttributeRule*, relating to the attributes, relationships or inverse relationships, if the *EntityRule* represents an entity itself

- a *Template* element, if present, it links to a partial concept template that shall be used to expand the concept template further.
 - EXAMPLE: the definition of a property set is used in different concept templates, for property sets on occurrences and for property sets on types. Hence it can be defined once as a partial concept template that is referenced from the main concept template through *References*.
 - NOTE: The underlying type of the *EntityRule*, defined by the *EntityName* attribute, and the *applicableEntity* of the referenced template should be the same.

The *mvd:ModelView* element describes how the concept templates are used in a view and contains:

- *@applicableSchema*, such as IFC2X3 or IFC4,
- *mvd:BaseView* definition if it is an add-on view that only defines restrictions or extensions on top of another model view definition,
- *mvd:ExchangeRequirements*, a list of *mvd:ExchangeRequirement*, that stipulate if the template rules imposed on concepts, declared for each *ConceptRoot*, have to be fulfilled for the individual exchange requirements,
- *mvd:Roots*, a list of *mvd:ConceptRoot*, that defined the concepts applicable to each entity instance in an IFC data set together with the template rules
- *Definitions*, see above

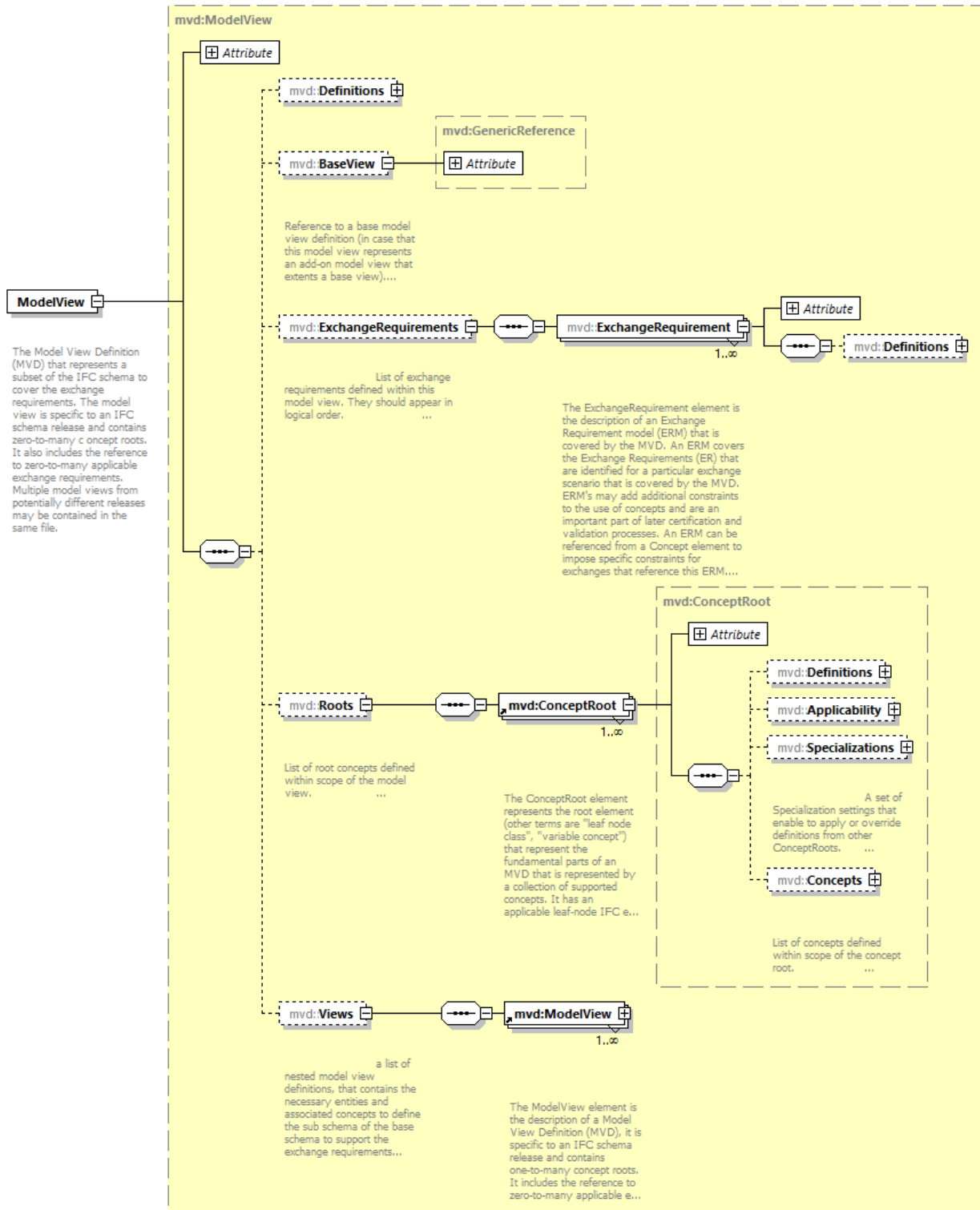


Figure 3: main mvdXML elements for ModelView

An *mvd:ConceptRoot* references a specific IFC entity, e.g. *IfcWall*, representing a major and individually testable model element³ in a MVD. Each concept root contains

- *mvd:Concepts*: a set of concepts, *mvd:Concept*, which describe template rules for common subsets of information (e.g. material usage) within the context of the particular concept root.
- *mvd:Template*: Each concept is backed by one or more templates, *mvd:ConceptTemplate*, describing a graph of object instances, relationships, and constraints, where the concept may provide a set of template rules containing the parameters that apply to the referenced rule ID at the concept template. The *mvd:Template* provides a link, based on the uuid, to that *mvd:ConceptTemplate*
- *mvd:Requirements*: a list of *mvd:Requirement*, each linking to the *mvd:ExchangeRequirement* by uuid, to declare that this Concept is stipulated for this exchange.
- *mvd:TemplateRules*: a tree of *mvd:TemplateRule* that creates a Boolean logic between individual template rules (applying and, or, and other Boolean operators). The outermost *TemplateRules* element has to validate to true for this concept to pass validation
- *mvd:Applicability*: a list of *TemplateRule(s)* with a link to the applicable *ConceptTemplate(s)* via the *Template* element. It optionally applies additional constraints on the applicable entity that needs to be fulfilled by the entity instance before the *Concepts* are validated.
- *mvd:Specializations*: a set of Specialization settings that enable to apply or override definitions from other ConceptRoots. This is needed to specify exceptions that are inherited from generic definitions. But it also enables to reuse existing definitions to another set of applicable entities.

NOTE: All definitions are applied to subtypes by default. This is defined by the IFC inheritance tree and may not always fit to requirements. The Specialization feature enables to control and adjust inheritance of such definitions.

2.2 Simple example

The following simple example shows the use of mvdXML for validation purposes. It defines a necessary concept template describing the unit of functionality of how to associate a port to a distribution element in IFC, and a hypothetical model view definition, that enforces that every sensor within submitted IFC data complying with the MVD shall have at least one port that submits signals.

Header section:

```
<?xml version="1.0" encoding="UTF-8"?>
<mvdXML
  name="example MVD for mvdXML documentation - sensor signals"
  uuid="4afb1a8b-0b61-4ff8-9863-c10690fe06f2"
  xmlns="https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=" https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/mvdXML_V1-2.xsd">
```

³ The IFC schema differentiates between root entities, all entities derived from *IfcRoot*, and resource entities, all other entity definitions. A resource entity shall always be used (referenced) by a root entity. Therefore, entities defined in an *mvd:ConceptRoot* should be an IFC root entity.

A single mvdXML element shall be the single root element within an mvdXML file. The name space of the current mvdXML version shall be <https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2>⁴. The schema location is provided locally.

```
<DefaultSettings precision="1e-4">
  <Units>
    <IfcSIUnit UnitType="lengthunit" Prefix="milli" Name="metre" />
  </Units>
</DefaultSettings>
```

A *DefaultSettings* is declared that defines a global relative precision value for comparison of floating-point values and global units for comparison of measurements. Units are declared using the existing IFC4 functionality based on the ifcXML.xsd schema. In the given example a relative precision of +/- 0.0001 and the unit for length measurements is defined to be [mm]. This global setting can be adjusted for each check.

```
<Templates>
  <ConceptTemplate uuid="baf93b7-d0e2-42d8-84cf-5da20ee1480a"
    name="Port Assignment" applicableSchema="IFC4"
    applicableEntity="IfcDistributionElement">
    <Definitions>
      <Definition>
        <Body>
<![CDATA[<p>Distribution ports are defined by <i>IfcDistributionPort</i> and attached
by the <i>IfcRelNests</i> relationship. Ports can be distinguished by the
<i>IfcDistributionPort</i> attributes <i>Name</i>, <i>PredefinedType</i>, and
<i>FlowDirection</i>:</p>]]>
        </Body>
      </Definition>
    </Definitions>
```

A single *ConceptTemplate* is declared with the name “Port Assignment”, based on the schema definition of IFC4. The applicable entity, and root entity of the concept template, is *IfcDistributionElement*. The definition of the attribute rules starts from that root entity.

```
<Rules>
  <AttributeRule AttributeName="IsNestedBy">
    <EntityRules>
      <EntityRule EntityName="IfcRelNests">
        <AttributeRules>
          <AttributeRule AttributeName="RelatedObjects">
            <EntityRules>
              <EntityRule EntityName="IfcDistributionPort">
                <AttributeRules>
                  <AttributeRule AttributeName="Name" RuleID="Name"/>
                  <AttributeRule AttributeName="PredefinedType" RuleID="Type"/>
                  <AttributeRule AttributeName="FlowDirection" RuleID="Flow"/>
                </AttributeRules>
              </EntityRule>
            </EntityRules>
          </AttributeRule>
        </AttributeRules>
```

⁴ NOTE: This namespace is a proposal and still must be approved by buildingSMART. It follows the naming convention for publishing bS standards. All content related to this draft standard is currently hosted at: <https://github.com/buildingSMART/mvdXML/tree/master/mvdXML1.2>

Page no.	Authors
19	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    </EntityRule>
  </EntityRules>
</AttributeRule>
</Rules>
</ConceptTemplate>
</Templates>

```

The concept template tree follows the IFC definition for that portion that is required to describe, how a distribution port is associated to a distribution element. It is defined by a path starting from the applicable entity.

- *IfcDistributionElement* – *IsNestedBy* → *IfcRelNests* – *RelatedObjects* → *IfcDistributionPort*

At the related *IfcDistributionPort*, the three necessary attributes *Name*, *PredefinedType* and *FlowDirection* are declared. Since the instantiation of those attributes shall be checkable, they each have a *@RuleID*. The *@RuleID* attributes shall be unique within the scope of its usage in a *ConceptTemplate*. The validation rules at the individual concepts use the *@RuleID* strings as variable names within the formal grammar.

```

<Views>
  <ModelView uuid="72dad5df-6f61-49f2-ba8c-baccf24a6ce5"
    name="Sensor signal view" applicableSchema="IFC4" code="Sensor">
    <Definitions>
      <Definition>
        <Body lang="en"><![CDATA[ModelView for mvdXML 1.2 documentation.]]></Body>
      </Definition>
    </Definitions>
  </ModelView>

```

A single Model View Definition MVD is defined within the mvdXML file, it is applicable to the IFC4 schema and has a name and code. In case of fully defined and published MVD's, the name is the full name as published, and the code is the same abbreviation, as used in the IFC HEADER Section.

EXAMPLE: name="IFC4 Reference View Version 1.2" and code="IFC4 RV 1.2" see official documentation⁵

```

<ExchangeRequirements>
  <ExchangeRequirement uuid="ae70f764-938b-4cf7-9814-c29a47f56b0e"
    name="Distribution signal" code="ERM1" applicability="export">
    <Definitions>
      <Definition>
        <Body lang="en">
<![CDATA[Simple example for checking sensor elements to always submit signals.]]>
        </Body>
      </Definition>
    </Definitions>
  </ExchangeRequirement>
</ExchangeRequirements>

```

For the MVD there is one exchange requirement defined. Each exchange requirement has an own selection of validation rules, so that data requirements and data completeness can be described specifically for an exchange.

NOTE: An example for an exchange requirement is the import requirements for a BIM usage or purpose.

```

<Roots>
  <ConceptRoot uuid="8b949664-a5df-4bfc-922c-4a486c41d756" name="Sensor"
    applicableRootEntity="IfcSensor">

```

⁵ https://standards.buildingsmart.org/MVD/RELEASE/IFC4/ADD2_TC1/RV1_2/HTML/

Page no.	Authors
20	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

<Definitions>
  <Definition>
    <Body>
<![CDATA[<p>A sensor is a device that measures a physical quantity and converts it
into a signal which can be read by an observer or by an instrument.</p>]]>
    </Body>
  </Definition>
</Definitions>

```

The MVD (or the MVD part that shall be validated) consists of a single root concept, i.e. a single IFC model element (or occurrence object), here *IfcSensor*.

```

<Concepts>
  <Concept uuid="a4fa348c-a025-4a02-abfd-c42fd0901540" name="Port Assignment">
    <Definitions>
      <Definition>
        <Body lang="en">
<![CDATA[Concept to validate that every sensor elements has a port defined that
submits signals.]]>
        </Body>
      </Definition>
    </Definitions>
    <Template ref="baafc93b7-d0e2-42d8-84cf-5da20ee1480a"/>

```

The *ConceptTemplate* “Port Assignment”, which is applicable to *IfcSensor*, since *IfcSensor* is a subtype of the *@applicableEntity IfcDistributionElement*, for which the *ConceptTemplate* is declared, is assigned as a *Concept* with the same name “Port Assignment”.

The assignment is declared using an ID/IDREF pair, based on the uuid for the *ConceptTemplate*.

NOTE: If using mvdXML for MVD definition and documentation purposes, this statement means that any implementation of *IfcSensor* has to support the functionality to assign ports to the sensor in order to comply with the requirements of that MVD. A certification process for that MVD would impose tests to make sure that ports are assigned to sensors for import and/or export.

```

<Requirements>
  <Requirement exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e"
    requirement="mandatory" applicability="export"/>
</Requirements>

```

Reference to the exchange requirement where additional constraints apply to the data provided for the *Concept* “Port Assignment”. The link is declared using an ID/IDREF pair, based on the uuid for the *ExchangeRequirement*. The logical results created by the *TemplateRule(s)* are interpreted following the *@requirement* attribute.

EXAMPLE: The requirement=“mandatory” stipulates, that the outcome of the single outermost *TemplateRule* shall be true, otherwise an error is reported.

```

<TemplateRule Parameters="Name[Value]='Output' AND Type[Value]='SIGNAL' AND
Flow[Value]='SOURCE'" Description="Transmits signal."/>

```

NOTE: In mvdXML 1.1 the first element by default is *TemplateRules*. This is no longer necessary if there is a single *TemplateRule* definition only. This change simplifies such definitions and also avoids confusion if using a binary operator like *OR*. Negation of a single *TemplateRule* is done in mvdXML 1.1 with the operator *NOR*, which since mvdXML 1.2 is defined with the optional *negation* attribute in the *TemplateRule*.

Page no.	Authors
21	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

A rule, referring to the template definition, hence the name “*TemplateRule*” is imposed, and an mvdXML compliant validator would check that each instance of *IfcSensor* in the IFC file would have an assigned *IfcDistributionPort* with the attributes and corresponding values Name=“Output”, Type=“SIGNAL”, and Flow=“SOURCE”.

NOTE: The parameter syntax of mvdXML 1.0 is using a semicolon between the parameters for defining AND combination. The use of semicolon is still supported, but it is recommended to use AND instead. The metric [Value] checks the value of the attribute. It is the default metric, therefore Name='Output' is identical to Name[Value]='Output'.

```
</Concept>
</Concepts>
</ConceptRoot>
</Roots>
</ModelView>
</Views>
</mvdXML>
```

3 Description of mvdXML schema elements and types

This section documents the individual mvdXML element, type and attribute definitions.

3.1 mvdXML

This element comprises the scope of the mvdXML document; it includes zero-to-many *mvd:ModelView* and zero-to-many *mvd:ConceptTemplate* (as a minimum, all concept templates that are referenced in the included model view(s)).

It is recommended to include all concept templates that are referenced by included model view(s), or else distribute the mvdXML file along with other mvdXML files containing such templates.

NOTE: A particular usage of mvdXML is to publish concept templates only. In this case, the *ModelView* element remains empty.

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
<i>DefaultSettings</i>	DefaultSettings [0:1]	Global settings about precision and units that apply to the whole mvdXML file unless overridden by rule specific settings. HISTORY: added in mvdXML1.2
<i>@Units</i>	ifc:IfcUnit [0:?]	Global unit settings based on the IFC4 specification for comparison of measurement values. If not defined, comparison of measurement values are based on SI units. NOTE: If no unit is defined for data checking, the comparison with the IFC data is done without unit conversion.
<i>@precision</i>	precisionValue (opt)	Precision setting for comparison of number values. Unless stated to be an absolute precision, the setting is used as relative precision.
<i>Templates</i>	ConceptTemplate [0:?]	Set of templates, which may be exchanged with or without referencing model view definitions.
<i>Views</i>	ModelView [0:?]	List of model view definitions, in order of listing in generated documentation. <ul style="list-style-type: none"> If empty, the mvdXML file is only used to exchange concept templates and cannot be used to fulfill other purposes such as data validation

Table 1: Common element references defined in the element *mvdXML*.

3.2 Concept Template

This element represents the reusable concepts as templates; it has zero-to-many *mvd:SubTemplates* and thereby may form a tree of related reusable concept templates. Within the tree it may refer to shared partial concepts. Each *mvd:ConceptTemplate* has an applicable schema and may have applicable root entities (i.e. concept roots to which the *mvd:ConceptTemplate* applies).

NOTE: For buildingSMART compliant MVD documentation generation, each *mvd:ConceptTemplate* appears in Chapter 4 of the resulting HTML based documentation, with descriptive text and diagram generated from rules.

EXAMPLE: Decomposition (the re-usable concept of decomposing elements into parts)

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
Definitions		See section 3.4.2

applicableSchema	extensible enum based on String	Identifies the default schema for which the template applies, such as IFC2X_FINAL, IFC2X2_FINAL, IFC2X3, IFC4 and newer IFC versions. The template may be used for model views of other schemas if all enclosed rules resolve to available attributes and types. NOTE: In future versions it might be of interest to support more than one IFC release. This can be supported by using a semicolon as schema name delimiter.
applicableEntity	String (list, opt)	Indicates the <i>IfcRoot</i> -based entities for which the concept applies. It is recommended to use a single base class (e.g. <i>IfcElement</i>). This value provides the context for any attribute rules and is used within MVD tools to filter the list of available templates for particular entities. For a sub-template, the applicable entity must be the same type or a subtype of the outer template. This value may be blank to indicate an abstract template that cannot be instantiated, containing sub-templates for specific entities. NOTE: The definition will include all derived entities from the IFC inheritance tree. If <i>applicableEntity</i> is for instance <i>IfcProduct</i> , then this <i>ConceptTemplate</i> also applies to all subtypes of <i>IfcProduct</i> .
isPartial	Boolean (opt)	A flag, indicating whether the concept template is a partial template, which shall only be used inside another concept template, or not.
Rules	AttributeRule [0:?]	Set of attributes defined at <i>applicableEntity</i> , where each attribute may have value constraints and/or graphs of object instances defined. If an attribute is not defined, then the requirements are the same as indicated for the schema. NOTE: For each attribute there should be no more than one <i>AttributeRule</i> . NOTE: It is allowed to define rules for attributes that are defined in a subtype of <i>applicableEntity</i> . This feature can be used in IFC for instance for the <i>PredefinedType</i> attribute defined for each subtype of <i>IfcElement</i> . NOTE: For generating a subset schema it is mandatory to add an <i>AttributeRule</i> for each optional attribute that shall be included in the subset schema. Otherwise this attribute will be removed.
SubTemplates	ConceptTemplate [0:?]	Set of sub-templates, having a subset of applicable entities, which further define a concept template for particular usage. For example, a template for material usage may have sub-templates for material layer sets, material profile sets, and material constituent sets. NOTE: Sub-templates have to repeat rule definitions from super-templates in case they apply. Further restrictions can be added if necessary. If rules are not repeated, they do not apply for the sub-templates.

Table 2: Common attributes and element references defined in the element *ConceptTemplate*.

3.2.1 Attribute Rule

This element represents the specification of an attribute on an entity, with related constraints, and/or entity rules.

Element/Attribute	Type	Description
AttributeName	String	The case-sensitive name of the attribute relative to the enclosing <i>EntityRule</i> (if exists) or the enclosing <i>applicableEntity</i> of the <i>ConceptTemplate</i> .
RuleID	String (opt)	Identifies the rule for referencing at template rules defined within concepts, where specific parameters are applied for this rule. NOTE: The same <i>RuleID</i> might be used multiple times within a concept template definition, but it must be unique within the scope of its usage.

		EXAMPLE: If two <i>AttributeRules</i> are defined within different <i>EntityRules</i> , for instance one for <i>IfcPropertySingleValue</i> and the other one for <i>IfcPropertyEnumeratedValue</i> , then the same <i>RuleID</i> can be used because they are used in different scopes.
Description	String (opt)	Optional description of the rule.
<i>EntityRules</i>	<i>EntityRule</i> [0:?]	An empty list indicates that any type may be used according to the schema. If one or more entities or types are defined, then instances must match one of the entries. The list of entries is expanded by each referencing <i>TemplateRule</i> defined at a <i>Concept</i> , where downstream rules apply according to the matching entity rule. EXAMPLE: An attribute rule "Quantities" for <i>IfcElementQuantity</i> could add entity rules for <i>IfcQuantityLength</i> , <i>IfcQuantityArea</i> , <i>IfcQuantityVolume</i> etc. Each entity rule then defines an own scope depending on the referenced quantity type.
<i>Constraints</i>	Constraint [0:?]	Set of expressions, which all must evaluate to TRUE for the referenced attribute. This implies a Boolean AND combination.
@precision	precisionValue/Double (opt)	Optional precision setting to be used in the <i>Constraints</i> for comparison of number values. Unless stated to be an absolute precision, the setting is used as relative precision. If not defined, precision settings from <i>DefaultSettings</i> are used. NOTE: If no precision is defined for data checking, then the precision as defined in the IFC model is used instead. HISTORY: added in mvdXML1.2
@relativePrecision	Boolean (opt)	Must be <i>false</i> if absolute precision is required. If not given or <i>true</i> then it is used a relative precision. HISTORY: added in mvdXML1.2

Table 3: Common attributes and element references defined in the element *AttributeRule*.

3.2.2 Entity Rule

This element represents the specification of an entity (or value type) referenced by an attribute, either as a scalar reference or a reference from within a collection.

Element/Attribute	Type	Description
EntityName	String	The case-sensitive name of the entity (e.g. " <i>IfcBeam</i> ") which must be assignable to the enclosing <i>AttributeRule</i> (i.e. entity subtype or select member).
RuleID	String (opt)	Identifies the rule for referencing at template rules defined within concepts, where specific parameters are applied for this rule. NOTE: The same <i>RuleID</i> might be used multiple times within a concept template definition, but it must be unique within the scope of its usage. See also description of <i>AttributeRule</i> .@RuleID.
Description	String (opt)	Optional description of the rule.
<i>Template</i>	<i>ConceptTemplate</i> [0:1]	Optional reference to a partial template. An optional attribute "IdPrefix" can be given to ensure that <i>RuleIDs</i> of partial templates are unique within the scope of its usage. This attribute is used as a prefix for all referenced <i>RuleIDs</i> . NOTE: This definition was simplified in mvdXML 1.2. Instead of using <i>References</i> @ <i>Template</i> the <i>Template</i> reference is directly defined at <i>EntityRule</i> . The functionality of <i>GenericReference</i> was extended by optional <i>IdPrefix</i> setting.
<i>AttributeRules</i>	<i>AttributeRule</i> [0:?]	Indicates a list of attributes included in the concept template and potentially constrained on the referenced entity.

<i>Constraints</i>	Constraint [0:?]	Set of expressions, which all must evaluate to <i>True</i> for the referenced entity. This implies a Boolean <i>AND</i> combination.
@precision	precision/Value/ Double (opt)	Optional precision setting to be used in the Constraints for comparison of number values. Unless stated to be an absolute precision, the setting is used as relative precision. If not defined, precision settings from <i>DefaultSettings</i> are used. NOTE: If no precision is defined for data checking, then the precision as defined in the IFC model is used instead. HISTORY: added in mvdXML1.2
@relativePrecision	Boolean (opt)	Must be <i>false</i> if absolute precision is required. If not given or <i>true</i> then it is used a relative precision. HISTORY: added in mvdXML1.2

Table 4: Common attributes and element references defined in the element *EntityRule*.

3.2.3 Constraint

This element is defined within the elements *mvd:EntityRule* and *mvd:AttributeRule* and represents a restriction on an attribute, which may require the value, type, or collection size to have equality (or other comparison) to a literal value or referenced value.

Element/Attribute	Type	Description
Expression	String	[v1.1] A grammar is used to simplify parsing of expressions and to introduce new features like AND, OR and XOR. With minor changes the form "{Metric}{Operator}{Benchmark}" from mvdXML 1.0 is still valid. The rule grammar is defined in chapter 4. A slightly changed and update version is provided with mvdXML1.2. NOTE: One major difference to 1.0 is that it is not possible to use {Benchmark} only, i.e. to omit {Metric} and {Operator}.
<i>RuleMessage</i>	RuleMessage [0:?]	Optional message that can be given for an <i>Applicability</i> to show success, error, warning or other informal messages in multiple languages. HISTORY: added in mvdXML1.2

Table 5: Common attributes defined in the element *Constraint*.

3.3 Model View

This element represents the description of a Model View Definition (MVD); it is specific to an IFC schema release and contains zero-to-many *mvd:ConceptRoot* elements. It also includes the reference to zero-to-many applicable *mvd:ExchangeRequirement* elements. Multiple model views from potentially different schema releases may be contained in the same file.

The set of entities and types regarded to be within scope of a model view is not explicitly defined; rather it is indirectly determined by constructing a graph of *mvd:ConceptRoot* elements and following the set of rules indicating referenced entities within scope. Thus, describing the set of rules automatically determines what is in or out of scope, preventing the possible mismatch of missing data structures that are required, or included data structures that are not documented for use.

EXAMPLE: The "CoordinationView_V2.0" is a Model View Definition; it is captured by an *mvd:ModelView* element. It has the @name="CoordinationView_V2.0", the @applicableSchema="IFC2X3", and a reference to the four exchange requirements currently defined for the Coordination View Version 2.0.

HISTORY: Roots changed to optional in mvdXML 1.1 to allow "incomplete" model view definitions with meta-data only.

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
Definitions		See section 3.4.2
applicableSchema	String	Identifies the schema using the ISO 10303 schema identifier, such as IFC2X_FINAL, IFC2X2_FINAL, IFC2X3, IFC4 or newer version. Use of correct names are enforced by an enumeration list for existing IFC releases and a naming pattern for future IFC releases. NOTE: In future versions it might be of interest to support more than one IFC release. This can be supported by using a semicolon as schema name delimiter.
BaseView	uuid anyURI (opt)	Reference to a base model view definition (in case that this model view represents an add-on model view that extents a base view).
Exchange Requirements	Exchange Requirement [0:?]	List of exchange requirements defined within this model view. They should appear in logical order.
Roots	ConceptRoot [0:?]	List of root concepts defined within scope of the model view.
Views	ModelView [0:?]	Collection that allows for hierarchical nesting of model views, where the outer view implicitly contains all content of inner views. NOTE: No cyclic structure is allowed, it must be a tree structure only. HISTORY: added in mvdXML1.2

Table 6: Common attributes and element references defined in the element *ModelView*.

3.3.1 Exchange Requirement

This element is the description of an Exchange Requirement Model (ERM) that is covered by the MVD. An ERM covers the Exchange Requirements (ER) that are identified for a particular exchange scenario that is covered by the MVD. ERM's may add additional constraints to the use of concepts and are an important part of later certification and validation processes.

An ERM can be referenced from an *mvd:Concept* to impose specific constraints for exchanges that reference this ERM. An ERM can be specifically declared to be only applicable for import, export or both scenarios using the attribute *applicability*.

EXAMPLE: The ERM "Architecture" capturing the ER for exporting an architectural building model is an exchange requirement model within the CoordinationView_V2.0. It is captured by an *mvd:ExchangeRequirement* element. It has the @name="Architecture", and the @applicability="export".

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
Definitions		See section 3.4.2
applicability	Enum (opt)	Identifies if the ERM is specific for <ul style="list-style-type: none"> import export both If such value is provided, then any referencing requirements must match; for example, if such value indicates <i>export</i> , then referencing requirements may use <i>export</i> but not <i>import</i> ; if such value is not provided, then referencing requirements may use any value. NOTE: The differentiation between import and export origins from software certification and does not have any meaning for data checking applications.

		<p>Export means that some application must be able to create a data set that fulfills defined requirements.</p> <p>If an exchange requirement is defined for import only, it defines the data set that must be properly processed by an application.</p>
--	--	--

Table 7: Common attributes defined in the element *ExchangeRequirement*.

3.3.2 Concept Root

This element represents the root element (other terms are "leaf node class", "variable concept") that represent the fundamental parts of an MVD that is represented by a collection of supported concepts.

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
Definitions		See section 3.4.2
applicable RootEntity	String	<p>Identifies the class or data type of instance being described or validated, i.e. the IFC entity (deriving from <i>IfcRoot</i>) for which the concepts apply. The concepts apply to this IFC entity or its subtypes (respectively instances of those classes in case of validation).</p> <p>NOTE that non-rooted entities are described by referencing rules, as such instances cannot exist on their own where usage is always dependent upon the referencing <i>IfcRoot</i>-based instance.</p>
Applicability	Applicability [0:1]	<p>One or more <i>TemplateRule(s)</i>, based on one or more <i>ConceptTemplates</i>, which describe the conditions, under which the concepts apply to the <i>applicableRootEntity</i>. Those conditions need to validate to true as a prerequisite for checking the <i>TemplateRules</i> imposed at the concepts.</p> <p>NOTE the Applicability has been added to mvdXML1.1 in order to better support data validation. It is used to control the applicability of concepts to particular configurations of root entities, e.g. to only apply for load bearing walls, instead of any wall (declared by the applicable IFC entity <i>IfcWall</i>.)</p>
Specializations	Specialization [0:?]	<p>A set of Specialization settings that enable to apply or override definitions from other <i>ConceptRoots</i>.</p> <p>HISTORY: added in mvdXML1.2</p>
@Definitions		See section 3.4.2
@ConceptRoot	ConceptRoot [0:?]	A Specialization setting enables either to apply or to override definitions from other <i>ConceptRoots</i> .
@override	Boolean (opt)	<p>If override is set to <i>true</i>, then all definitions from referenced <i>ConceptRoots</i> do not apply for objects selected by this <i>ConceptRoot</i>. It enables to override definitions automatically inherited by the IFC inheritance tree. For example, definitions for <i>IfcBeam</i> objects can disable definitions inherited from <i>IfcBuildingElement</i> objects.</p> <p>If override is set to <i>false</i>, then all definitions from referenced <i>ConceptRoots</i> should apply for objects selected by this <i>ConceptRoot</i>. It enables to reuse definitions even if referenced applicability does not match.</p> <p>Default setting, if not given, is false.</p>
Concepts	Concept [0:?]	List of concepts for the applicable root entity. The order of elements indicates the sequence displayed in generated documentation.

Table 8: Common attributes and element references defined in the element *ConceptRoot*.

3.3.3 Applicability

This element defines those rules, being *TemplateRules* with a reference to a *ConceptTemplate*, that need to be validated before the concepts associated to the *ConceptRoot* are checked. Since mvdXML1.2 *Applicability* is also used to define further conditions for the use of *TemplateRule* and *TemplateRules*.

Element/Attribute	Type	Description
Definitions		See section 3.4.2
<i>Template</i>	TemplateRef [0:?]	Reference to <i>ConceptTemplate</i> by uuid, where such template may be defined within the same file (by @ref) or an external file (by @href). If no <i>ConceptTemplate</i> is defined, then only <i>minOccurrence</i> or <i>maxOccurrence</i> constraints on existence of instances can be defined. NOTE Current usage of mvdXML imposes the inclusion off all concept templates within the same data file. The external reference by @href is reserved for future usage. NOTE Cardinality of this reference has changed in mvdXML1.2. It now enables to use no, one or more than one <i>ConceptTemplate</i> for rule configuration. No link to a template is needed if existence via min/maxOccurrence of <i>applicableRootEntity</i> shall be constraint.
<i>TemplateRule(s)</i>	TemplateRules TemplateRule [0:1]	Tree structure of rules indicating how template applies to particular entity. Each <i>TemplateRules</i> element consists of a set of other <i>TemplateRules</i> or <i>TemplateRule</i> element and a logical operator. Each <i>TemplateRule</i> element defines the @Parameter that refer to the RuleID of the referenced <i>Template</i> . NOTE Added in mvdXML 1.1 to define any logical combination of rules, allowing for AND, OR, NAND, NOR, XOR, and NXOR logic. Since mvdXML1.2 the NOT operator at <i>TemplateRules</i> (for the logical combination of 2 or more <i>TemplateRule(s)</i>) has been removed. This unary operation is now handled in <i>TemplateRule.negation</i> exclusively. HISTORY: changed in mvdXML1.2
<i>RuleMessage</i>	RuleMessage [0:?]	Optional message that can be given for an <i>Applicability</i> to show success, error, warning or other informal messages in multiple languages. HISTORY: added in mvdXML1.2
minOccurrence	Integer, >=0 (optional)	Minimum number of occurrences that should fulfill the <i>Applicability</i> settings. If not given it is allowed to have zero occurrences. HISTORY: added in mvdXML1.2
maxOccurrence	Integer, >=0 (optional)	Maximum number of occurrences that should fulfill the applicability settings. If not given there is not upper limit. If given, maxOccurrence must be greater than or equal to minOccurrence. HISTORY: added in mvdXML1.2

3.3.4 Concept

This element represents a use definition for a particular entity with specific rules to be enforced.

Element/Attribute	Type	Description
Identity attributes		See section 3.4.1
Definitions		See section 3.4.2

@baseConcept	Concept [0:1]	Enables to select a concept definition that shall either be reused or redefined. This feature depends on the inheritance tree of the underlying schema. If <i>ConceptRoot.applicableRootEntity</i> is defined for an entity called <i>IfcProject</i> , then only those concepts can be selected as <i>BaseConcept</i> that are defined for a super type of <i>IfcProject</i> , i.e. <i>IfcContext</i> , <i>IfcObjectDefinition</i> or <i>IfcRoot</i> .
@override	Boolean (opt)	This value must be defined if a <i>BaseConcept</i> is selected. If <i>Override</i> = false then the Concept is reused without changes. If true then the Concept from the super type is redefined by this Concept.
Template	TemplateRef [0:?]	Reference to the <i>ConceptTemplate(s)</i> by uuid, where such template may be defined within the same file (by @ref) or an external file (by @href). NOTE Current usage of mvdXML imposes the inclusion off all concept templates within the same data file. The external reference by @href is reserved for future usage.
Requirements	Requirement [0:?]	Set of requirements which describe applicability of the concept to particular exchanges for import, export or both.
TemplateRules	TemplateRules TemplateRule [0:1]	Tree structure of rules indicating how template applies to particular entity. Each <i>TemplateRules</i> element consists of a set of other <i>TemplateRules</i> or <i>TemplateRule</i> element and a logical operator. Each <i>TemplateRule</i> element defines the @Parameter that refer to the <i>RuleID</i> of the referenced <i>Template</i> . NOTE Added in mvdXML 1.1 to define any logical combination of rules, allowing for AND, OR, NAND, NOR, XOR, and NXOR logic. Since mvdXML1.2 the NOT operator at <i>TemplateRules</i> (for the logical combination of 2 or more <i>TemplateRule(s)</i>) has been removed. This unary operation is now handled in <i>TemplateRule.negation</i> exclusively. HISTORY: changed in mvdXML1.2

Table 9: Common element references defined in the element *Concept*.

NOTE: The following options are possible for using *BaseConcept* and *Override*:

- an empty concept with BaseConcept="idref" (Override="false") to indicate that it applies with no change
- a non-empty concept with BaseConcept="idref" (Override="false") to indicate that it has additional rules (by extension)
- an empty concept with BaseConcept="idref" and Override="true" to indicate that it does not apply at all (overridden)
- a non-empty concept with BaseConcept="idref" and Override="true" to indicate that it has replacement rules (by restriction - no inherited rules apply, all are declared new)

3.3.5 Requirement

This element represents a use definition for a particular entity with specific rules to be enforced.

Element/Attribute	Type	Description
@exchange Requirement	uuid	Identifies the <i>ExchangeRequirement</i> by GUID within the same Model View Definition.
@requirement	Enum	Describes the interpretation of the result of the outermost <i>TemplateRule</i> specific for one exchange requirements. <ul style="list-style-type: none"> ▪ <i>mandatory</i>: must be true, otherwise create an error ▪ <i>recommended</i>: should be true, otherwise create a warning ▪ <i>not-relevant</i>: no requirement; ▪ <i>not-recommended</i>: should not be true, otherwise create a warning

		<ul style="list-style-type: none"> ▪ <i>excluded</i>: must not be true, otherwise create an error
@applicability	Enum (opt)	Identifies if the requirement applies to <ul style="list-style-type: none"> ▪ import ▪ export or ▪ both NOTE If such value is provided, then it must match, if given, with the applicability setting of the exchange requirement in which it is used.

Table 10: Common element references defined in the element *Requirement*.

Note: For a standard existence check, e.g. is the property value provided, it means:

- mandatory: error, if no value is provided,
- recommended: warning, if no value is provided,
- not-relevant – no check,
- not-recommended – warning, if a value is provided,
- excluded – error, if a value is provided.

Since mvdXML1.2 dedicated error, warning and success messages for checks can be given to better explain the rule.

3.3.6 TemplateRules

This element establishes the possibility to define a tree of logical expressions. Individual *TemplateRule* are grouped under a *TemplateRules* element and are logically interpreted by the *@operator* attribute.

NOTE: This improves the previous way to embed the logical operator in the *@Parameter* string at the *TemplateRule*. Due to its tree structure realized by the recursive definition of *TemplateRules*, the logical operators can be nested.

Element/Attribute	Type	Description
Description		See section 3.4.2
@operator	Enum	The logical operator, which is used to combine the nested <i>TemplateRules</i> and <i>TemplateRule</i> . The Boolean results of the nested rules are combined by the logical operation according to the Truth table. The following logical operators are defined: <ul style="list-style-type: none"> ▪ AND ▪ OR ▪ NAND ▪ NOR ▪ XOR ▪ NXOR NOTE: Since mvdXML1.2 these operations are always applied to 2 or more rules. The unary operation <i>NOT</i> was therefore removed and is now handled with <i>TemplateRule @negation</i> . HISTORY: changed in mvdXML1.2
<i>TemplateRule(s)</i>	TemplateRules TemplateRule [2:?]	Two or more <i>TemplateRule(s)</i> that are combined through the given logical operator. HISTORY: changed in mvdXML1.2

<i>RuleMessage</i>	RuleMessage [0:?]	Optional message that can be given for an <i>Applicability</i> to show success, error, warning or other informal messages in multiple languages. HISTORY: added in mvdXML1.2
<i>Applicability</i>	Applicability [0:1]	Optional condition for the given rules. If the condition evaluates to true, then the rule is applied. It can also be defined if the condition is optional (default) or must be fulfilled (minOccurrence=1). In case of checking a list of references the minimum and/or maximum number of occurrences can be given. If no maxOccurrence is given, the given rules are applied to all occurrences that fulfill this condition. NOTE: This feature for instance enables to check the existence of material properties for each layer of the wall. It also enables to exclude an air layer from this requirement check. The detailed example is given in 0 HISTORY: added in mvdXML1.2

Table 11: Common attributes defined in the element *TemplateRules*.

The following truth tables are to be used with the @operator.

A	B	A AND B	A OR B	A NAND B	A NOR B	A XOR B	A NXOR B
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

Table 12: Truth table for operator attribute

3.3.7 TemplateRule

This element represents an instantiation of a rule with specified parameters. It allows repetitive definitions to be efficiently represented, such as lists of applicable ports, materials, units, property sets, etc. The *@RuleID* used in the Parameters of the template rule serves as a reference to the *@RuleID* of an *mvd:AttributeRule* or *mvd:EntityRule* at the referenced *mvd:ConceptTemplate* to be instantiated.

If the referenced *mvd:EntityRule* is part of a SET-based attribute, then the instance is required to uniquely exist once (having unique combination of defined parameters), but without regard for order (as a SET has no implied order). If the referenced *mvd:EntityRule* is part of a LIST-based attribute, then the instance is required to occur at the relative position of the *mvd:TemplateRule*.

Element/Attribute	Type	Description
Description		See section 3.4.2

Parameters	String	<p>[v1.1] mvdXML introduces a grammar definition for the <i>Parameters</i> string that is harmonized with the <i>Expression</i> string of <i>mvd:Constraint</i>. With minor changes the form “{Parameter}={Value};” from mvdXML 1.0 is still valid.</p> <p>In the new grammar the <i>Parameters</i> string is defined by <i>expression</i> where each <i>boolean_term</i> requires a <i>parameter</i>. The <i>parameter</i> corresponds to the <i>RuleID</i> of an <i>AttributeRule</i> or <i>EntityRule</i> at the referenced <i>ConceptTemplate</i>. The <i>operator</i> of the grammar is now more flexible and not only supports <i>Equals</i> as in mvdXML 1.0. The <i>Value</i> is also enhanced. It now supports the use of a <i>parameter</i>, not only a value. This enables to replace the agreement for the definition of parameter values using a ‘#’ sign. This agreement is no longer supported. Finally, each expression can be grouped and combined through AND, OR and XOR logic.</p> <p>NOTE: The differentiation between conditions and constraints as used in mvdXML 1.0 is no longer available.</p>
Order	Integer (opt)	<p>Attribute for indicating the order of rule for interleaving with rules from other concepts, such as used for properties of different types.</p> <p>STATUS: added in mvdXML1.2, clarification needed</p>
Usage	Enum (opt)	<p>Currently in clarification if needed</p> <p>Capturing usage implications, such as for data within a COBie spreadsheet. Allowed options are:</p> <ul style="list-style-type: none"> • required • optional • key • reference • system • calculation <p>STATUS: added in mvdXML1.2, clarification needed</p>
precision	precisionValue/ Double (opt)	<p>Optional precision setting to be used in the Constraints for comparison of number values. Unless stated to be an absolute precision, the setting is used as relative precision. If not defined, precision settings from <i>DefaultSettings</i> are used.</p> <p>NOTE: If no precision is defined for data checking, then the precision as defined in the IFC model is used instead.</p> <p>HISTORY: added in mvdXML1.2</p>
relativePrecision	Boolean (opt)	<p>Must be <i>false</i> if absolute precision is required. If not given or <i>true</i> then it is used a relative precision.</p> <p>HISTORY: added in mvdXML1.2</p>
negation	Boolean (opt)	<p>If true, the result of the logical expression defined by the <i>Parameters</i> must be negated.</p> <p>HISTORY: added in mvdXML1.2</p>
Requirements	Requirements [0:1]	STATUS: added in mvdXML1.2, clarification needed
RuleMessage	RuleMessage [0:?]	<p>Optional message that can be given for an <i>Applicability</i> to show success, error, warning or other informal messages in multiple languages.</p> <p>HISTORY: added in mvdXML1.2</p>
Applicability	Applicability [0:1]	<p>Optional condition for the given rules. If given, evaluation of <i>Applicability</i> must return <i>true</i> so that this <i>TemplateRule</i> is applicable.</p> <p>HISTORY: added in mvdXML1.2</p>

Table 13: Common attributes defined in the element *TemplateRule*.

3.3.8 RuleMessage

The element RuleMessage allows the definition of user specific messages. This mechanism is supported for *TemplateRules*, *TemplateRule*, *Applicability* and *Constraints*. Depending on its usage it helps to explain checking rules on different levels of granularity.

HISTORY: added in mvdXML1.2

Element/Attribute	Type	Description
@lang	language (opt)	Optional definition of the language of the message. Default if not given is "en".
@state	Enum	Describes the interpretation of the element where this message is used. <ul style="list-style-type: none"> ▪ <i>success</i> ▪ <i>failure</i> ▪ <i>warning</i> ▪ <i>comment</i>

3.4 Common type and attribute definitions

The *mvd:Definition*, and elements referenced by the element *Definition*, *mvd:Body*, and *mvd:Link* elements provide the capability to add multi-lingual descriptions at any element with own identity. Such elements are:

- *mvd:ModelView*
- *mvd:ExchangeRequirement*
- *mvd:ConceptTemplate*
- *mvd:ConceptRoot*
- *mvd:Concept*

The information provided by this element is mainly used for documentation purposes, in particular to generate HTML documentation as used by buildingSMART for the IFC data model.

3.4.1 Identity

Similar to IFC, the mvdXML schema makes a distinction between elements having identity and those that do not. All elements with identity have the following attributes and sub elements defined. The information provided in this attribute group is used for management purposes.

NOTE: The mvdXML.xsd does not incorporate an mvd:Identity abstract class, the common attributes are defined in the attributeGroup name="identity" and the definitions in the element name="Definition".

HISTORY: The attribute 'status' has been changed from string to an enumeration that includes the previously recommended values in mvdXML 1.1.

Attribute	Type	Description
uuid	uuid	Universally unique identifier. This is used as a persistent identifier, and must never change. It is string type with a fixed length of 36 characters, which should follow a specific pattern.

Attribute	Type	Description
name	String	Human readable name. This is used as the header of the section and entry within table of contents when generating documentation. The name is also reported for a validation against this MVD, if assigned to concepts checked against the MVD.
code	String (opt)	Human readable reference value of this element of the MVD definition
version	String (opt)	Sequential version number of this element of the MVD definition.
status	enumeration base: String (opt)	The status information of this element of the MVD definition. It has the following enumerators: <ul style="list-style-type: none"> ▪ Sample ▪ Proposal ▪ Draft ▪ Candidate ▪ Final ▪ Deprecated
author	String (opt)	The author(s) of his element of the MVD definition. Authors are separated by semicolon.
owner	String (opt)	The legal owner of this element of the MVD definition NOTE Official Model View Definitions by buildingSMART International shall have ownership assigned to buildingSMART or another accepted standardization organization.
copyright	String (opt)	The copyright under which the work is published. NOTE: If adopted by buildingSMART International, the copyright shall lie either with buildingSMART International, or is governed by a well-recognized open license (e.g. creative commons, open source BSD/ GNU).

Table 14: Common attributes defined in the attributeGroup *identity*.

3.4.2 Definition

The element *mvd:Definition* groups definition text and links to additional figures, diagrams, examples, and other external documents.

Attribute	Type	Description
<i>Body</i>	Body [0:1]	HTML-formatted description of the concept in the default language.
<i>Links</i>	Link [0:?]	List of additional content, each of which may be in separate languages.

Table 15: Common element references defined in the element *Definition*.

3.4.3 Body

The element *mvd:Body* holds the definition text or explanatory remarks. It is qualified by a language tag. It also holds tags that further classify the nature of the definition or remark.

NOTE: In order to correctly encapsulate the HTML formatted text, the content shall be tagged by `<![CDATA[]]>` to preserve the HTML code.

HISTORY: tags attribute available since mvdXML 1.1.

Page no.	Authors
35	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

Attribute	Type	Description
lang	String (opt)	Locale identifier based on RFC 1766 language codes to indicate the default locale. Examples are 'en', 'de', 'en-GB', 'de-CH'.
tags	String [0:?]	List of tags that classify the element. All tags are separated through whitespaces per default. A semicolon must be used if given tags consists of multiple words.
(content)	String	HTML-formatted content for generating documentation. Content within should be encapsulated by paragraph tags ("<p>") and/or list tags (""). Images should not be contained within; rather they should be specifically referenced by the Link element (allowing for automatic figure numbering).

Table 16: Common attributes defined in the element *Body*.

3.4.4 Link

The element *mvd:Link* holds all links to additional documentation content.

Attribute	Type	Description
lang	language	Locale identifier based on RFC 1766 language codes to indicate the default locale. Examples are 'en', 'de', 'en-GB', 'de-CH'.
title	String (opt)	Human readable name. This is used as the header of the link content and entry within table of contents when generating documentation
category	enumeration base: String (opt)	Indication about the category of the linked content. <i>definition</i> : formatted as documented definition in alternate locale <i>agreement</i> : formatted as NOTE in documentation <i>diagram</i> : formatted as custom figure in documentation based on <i>href</i> <i>instantiation</i> : formatted as instance diagram figure based on <i>href</i> <i>example</i> : formatted as EXAMPLE in documentation based on <i>href</i>
href	anyURI	URL to referenced content, particularly for diagrams and examples that are manually generated. This is used to reference any external files such that they are included when generating documentation. NOTE: URL's local to the file system shall be relative.
(content)	String	HTML-formatted description in specified language.

Table 17: Common attributes defined in the element *Link*.

3.4.5 GenericReference

The complex type *mvd:GenericReference* holds reference information to other elements identified by a uuid or a URI. It also enables to hold additional information for the use of this element, such as an *IdPrefix* for *RuleIDs*, *applicableEntityRuleID* and *baseTemplate*.

HISTORY: added in mvdXML1.2

Attribute	Type	Description
ref	uuid	Universally unique identifier of the reference element. It is string type with a fixed length of 36 characters, which should follow a specific pattern.
href	anyURI	URL to referenced content.
IdPrefix	String (opt)	An optional attribute "IdPrefix" can be given to ensure that RuleIDs of partial templates are unique within the scope of its usage. This attribute is used as a prefix for all referenced RuleIDs.

applicableEntityRuleID	String (opt)	If the referenced element is a <i>ConceptTemplate</i> this attribute specifies the RuleID of another <i>ConceptTemplate</i> being specified within the context of its usage to define the applicableEntity of this <i>ConceptTemplate</i> .
baseTemplate	Uuid (opt)	Optional <i>ConceptTemplate</i> that defines the RuleID as referenced by <i>applicableEntityRuleID</i> .

Table 18: Common attributes defined in the complex type *GenericReference*.

4 Rule Grammar

The grammar for parsing *expression* strings is defined below.
 It is used to specify:

- *mvd:TemplateRule.Parameters*, and
- *mvd:Constraint.Expression*.

Updated grammar using the style from mvdXML 1.1

```

grammar mvdXMLv1_2;
/*----- * PARSER RULES *-----*/
expression
  : boolean_expression ;
boolean_expression
  : NOT? boolean_term (logical_interconnection NOT? boolean_term)* ;
boolean_term
  : (( parameter ( metric )? | metric ) operator ( value | parameter ( metric )? ) ) |
    ( LPAREN boolean_expression RPAREN );
parameter
  : SIMPLEID | 'SELF';
metric
  : '[Value]' | '[Size]' | '[Type]' | '[Unique]' | '[Exists]';
logical_interconnection
  : AND | OR | XOR | NAND | NOR | NXOR ;
operator
  : EQUAL | NOT_EQUAL | GREATER_THAN | GREATER_THAN_OR_EQUAL | LESS_THAN |
    LESS_THAN_OR_EQUAL;
value
  : logical_literal | real_literal | string_literal | regular_expression;
logical_literal
  : FALSE | TRUE | UNKNOWN ;
real_literal
  : (sign)? ( DIGIT | INT ) ( '.' )? ( ( DIGIT | INT ) )? ( 'e' (sign)? ( DIGIT | INT ) )? ;
string_literal
  : STRING ;
regular_expression
  : 'reg' STRING ;
sign
  : '+' | '-' ;

/*----- * LEXER RULES *-----*/
AND
  : 'AND' | 'and' | '&' | ';' ;
OR
  : 'OR' | 'or' | '|' ;
XOR
  : 'XOR' | 'xor' ;
NAND
  : 'NAND' | 'nand' ;
NOR
  : 'NOR' | 'nor' ;
NXOR
  : 'NXOR' | 'nxor' ;
NOT
  : 'NOT' | 'not' | '!';
EQUAL
  : '=' ;
NOT_EQUAL
  : '!=' ;
GREATER_THAN
  : '>' ;
  
```

Page no.	Authors
38	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

GREATER_THAN_OR_EQUAL
: '>=' ;
LESS_THAN
: '<' ;
LESS_THAN_OR_EQUAL
: '<=' ;
FALSE
: 'FALSE' | 'false' ;
TRUE
: 'TRUE' | 'true' ;
UNKNOWN
: 'UNKNOWN' | 'unknown' ;
DIGIT
: '0'..'9' ;
INT
: '0'..'9'+;
HEX_DIGIT
: DIGIT | ('a'..'f' | 'A'..'F') ;
LETTER
: ('a'..'z') | ('A'..'Z') ;
SIMPLEID
: LETTER ( LETTER | DIGIT | '_' )* ;
LPAREN
: '(';
RPAREN
: ')';
OCTAL_ESC
: '\\\' ('0'..'3') ('0'..'7') ('0'..'7') |
  '\\\' ('0'..'7') ('0'..'7') |
  '\\\' ('0'..'7') ;
UNICODE_ESC
: '\\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT ;
ESC_SEQ
: '\\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\\'|'\\'|'\\') | UNICODE_ESC | OCTAL_ESC ;
STRING
: '\\\' ( ESC_SEQ | ~('\\'|'\\') )* '\\';
WS
: (' '|'\t'|'\n'|'\r')+ { $channel = HIDDEN; } ;

```

Modernized grammar for parsing with ANTLR:

```

grammar mvdXMLv1_2;

/*-----
*  PARSER RULES
*-----*/
expression
    : boolean_expression EOF;
boolean_expression
    : boolean_term (logical_interconnection boolean_term)* ;
boolean_term
    : NOT? ((leftside operator rightside) | (LPAREN boolean_expression RPAREN));
leftside
    : parameter_metric | metric;
rightside
    : parameter_metric | value;
parameter_metric
    : parameter (metric)?;
parameter
    : simple_id | SELF;
simple_id
    : letter ( letter | ZERO | DIGITNONZERO | '_' )* ;
metric
    : VALUE | SIZE | TYPE | UNIQUE | EXISTS;
logical_interconnection
    : AND | OR | XOR | NAND | NOR | NXOR;
operator
    : EQUAL | NOT_EQUAL | GREATER_THAN_OR_EQUAL | GREATER_THAN | LESS_THAN_OR_EQUAL | LESS_THAN;
value
    : logical_literal | real_literal | regular_expression | string_literal;
logical_literal
    : FALSE | TRUE | UNKNOWN ;
real_literal
    : sign? (trailing | int) (DOT decimal_part)? exp? ;
string_literal
    : STRING ;
regular_expression
    : REG STRING ;
int
    : DIGITNONZERO (digit)*;
decimal_part
    : trailing? int?;
exp
    : EXP sign int;
sign
    : PLUS | MINUS ;
digit
    : ZERO | DIGITNONZERO;
trailing
    : ZERO+;
letter
    : EXP | LOWER | UPPER;

/*-----
*  LEXER RULES
*-----*/
EQUAL
    : '=' ;
NOT_EQUAL
    : '!=' ;
GREATER_THAN_OR_EQUAL

```

Page no.	Authors
40	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;


```

: '>=' ;
GREATER_THAN
: '>' ;
LESS_THAN_OR_EQUAL
: '<=' ;
LESS_THAN
: '<' ;
AND
: 'AND' | 'and' | '&' | ';' ;
OR
: 'OR' | 'or' | '|' ;
XOR
: 'XOR' | 'xor' ;
NAND
: 'NAND' | 'nand' ;
NOR
: 'NOR' | 'nor' ;
NXOR
: 'NXOR' | 'nxor' ;
NOT
: 'NOT' | 'not' | '!';
VALUE
: '[' ('V'|'v') ('alue'|'ALUE') ']' ;
SIZE
: '[' ('S'|'s') ('ize'|'IZE') ']' ;
TYPE
: '[' ('T'|'t') ('ype'|'YPE') ']' ;
UNIQUE
: '[' ('U'|'u') ('nique'|'NIQUE') ']' ;
EXISTS
: '[' ('E'|'e') ('xists'|'XISTS') ']' ;
FALSE
: 'FALSE' | 'false' ;
TRUE
: 'TRUE' | 'true' ;
UNKNOWN
: 'UNKNOWN' | 'unknown' ;
REG
: 'reg';
SELF
: ( 'S' | 's' ) ( 'ELF' | 'elf' ) ;
EXP
: 'e' | 'E';
PLUS
: '+';
MINUS
: '-';
DOT
: '.';
LPAREN
: '(';
RPAREN
: ')';
ZERO
: '0';
DIGITNONZERO
: '1'..'9';
LOWER
: 'a'..'z';
UPPER
: 'A'..'Z';
ESC_SEQ

```

Page no.	Authors
41	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```
: '\\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\') ;  
STRING  
: '\\\' ( ESC_SEQ | ~('\\\\'|'\\') ) * '\\';  
WS  
: (' '\\t'|'\\n'|'\\r')+ -> skip;
```

The following tables describe more details about the meaning of keywords.

metric	Description
Value	Indicates the value of the attribute (value uses syntax according to the attribute type, defined below).
Size	Indicates the size of a collection or STRING (value is an INTEGER).
Type	Indicates the type of the value assigned to the attribute (value is a STRING).
Unique	Indicates whether value must be unique within the population of instances described within the Concept Template (BOOLEAN).
Exists	Indicates if an attribute or reference must exist ("true") or not ("false").

Table 19: Description of *metric* values.

operator	XML Escaped	Description
=	=	Equal.
!=	!=	Not Equal.
>	>	Greater Than.
>=	>=	Greater Than Or Equal.
<	<	Less Than.
<=	<=	Less Than Or Equal

Table 20: Description of *operators*.

Benchmark may be either a literal value or a parameter. The syntax of literal values varies according to the EXPRESS attribute type and follows ifcXML (ISO-10303-28) format:

Type	Operators	Description
INTEGER	=, !=, >, >=, <, <=	The integer value.
REAL	=, !=, >, >=, <, <=	The real value including decimal point, where equality is exact (no epsilon offset). NOTE: If a real value has a unit of measurement, then it is necessary to make sure that the unit defined in mvdXML for this measurement is the same as the unit of the data that is checked. If they are different (for example milli-metre vs. metre), the data needs to be converted first before doing the comparison.
BOOLEAN	=, !=	The boolean value as "true" or "false".
LOGICAL	=, !=	The logical value as "true", "false", or "unknown".
ENUM	=, !=	The enumeration value by case-insensitive name.
STRING	=, !=, >, >=, <, <=	The string value, which may optionally be enclosed by single quotes (if escaping required). Comparison operators indicate alphabetical sorting (e.g. ">=" can indicate "must start with" such as the scope of a classification reference, or earliest date/time). String datatypes can be compared with a regular expression. Such regular expression should be defined according to POSIX (ISO/IEC 9945).
BINARY	=, !=	The binary value encoded as hexadecimal prefixed by "%" and number of unused bits.
ENTITY	=, !=, >, >=, <, <=	The name of the entity type (e.g. "IfcWall"). Equality means exact type match; ">" means subtype of; ">=" means same type or subtype; "<" means supertype of; "<=" means same type or supertype.

Table 21: Description of *operators* that can be applied to different data types.

5 mvdXML Use Cases

This chapter describes typical use cases of mvdXML. Although the main structure of a Model View Definition is always the same there are differences regarding mandatory, optional and not relevant data. This review should enable to focus on features of mvdXML that are needed to support those use cases.

5.1 MVD Documentation

On basis of an mvdXML definition it is possible to generate a set of interlinked HTML files. The IfcDoc tool from buildingSMART enables to export HTML files using the style of the IFC4 documentation (see Figure 4). The main focus of such documentation is to describe how a subset of IFC must be implemented and used to fulfil specific requirements. Such kind of information is needed by software developers for proper interpretation of the IFC specification.

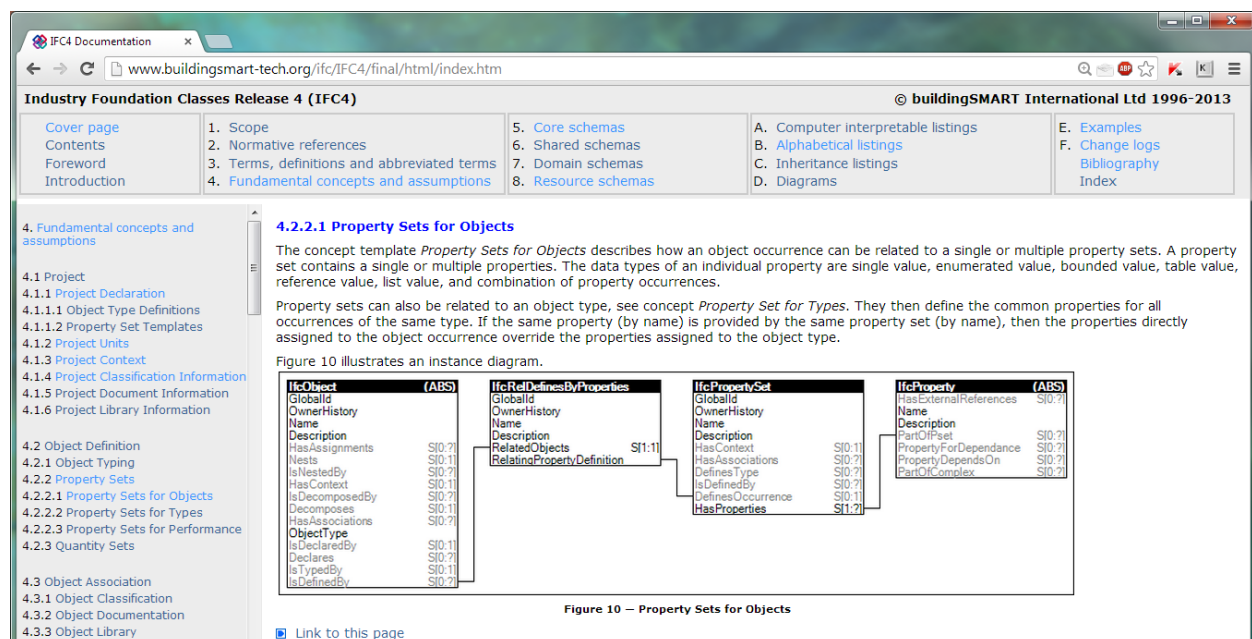


Figure 4: HTML documentation of IFC4

The following parts of mvdXML are of main interest:

mvd:Definition attached to *mvd:ModelView*, *mvd:ExchangeRequirement*, *mvd:ConceptTemplate*, *mvd:ConceptRoot* and *mvd:Concept* for storing definition text and links to additional figures, diagrams, examples, and other external documents

mvd:ConceptTemplate definitions that enable to generate instance diagrams as shown in Figure 4

NOTE: For documentation purposes it is not necessary to specify an *mvd:ConceptTemplate* in full detail. Instead it is sufficient to focus on elements that shall be shown in an instance diagram. This is different to other use cases, in particular for data filtering and data validation, where all required elements must be defined.

5.2 Specification of subset schemas

A subset schema includes only those parts of IFC that are relevant for implementation. Such subset schema, which can be defined in EXPRESS or XML Schema, supports software implementation as it enables to generate software code for file parsing (ifc or ifcXML), data management and data serialization.

Page no.	Authors
44	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

But a subset schema does not include additional constraints that define proper use of the subset schema. Therefore, the use of mvdXML can be simplified as it is only necessary to select entities and attributes that shall be part of the subset schema. Also, no documentation or other meta-data is required as it cannot be exported to EXPRESS or XML Schema.

The ifcDoc tool supports the generation of a partial EXPRESS schema. All non-mandatory definitions that are not part of the subset schema are removed. If attributes and references of supported entities must be removed, the non-instantiable type *IfcStrippedOptional* is used. The following EXPRESS snippet shows the use of this data type where *IfcBuildingSystem.LongName* was excluded from the sub schema.

```
SCHEMA IFC4;

TYPE IfcStrippedOptional = BOOLEAN;
END_TYPE;
..
ENTITY IfcBuildingSystem
  SUBTYPE OF (IfcSystem);
    PredefinedType : OPTIONAL IfcBuildingSystemTypeEnum;
    LongName : OPTIONAL IfcStrippedOptional;
END_ENTITY;
..
END_SCHEMA;
```

5.3 Data Filtering

Data filtering creates a model subset and is similar to generation of a subset schema. But a data filter is working on instance level, i.e. with data instead of schema definitions. Many instances can be defined for a data type, but not all instances must be part of a Model View Definition. Accordingly, data filtering may require adding further conditions that enable to differentiate instances of the same data type. This is leading to more complex definitions. For example, a condition is necessary if space properties must be distinguished from wall properties in an IFC model, because both are defined by *IfcPropertySet* instances.

5.4 Data Validation

Data validation checks if a data set fulfils all constraints of an exchange requirement. If required data is missing or wrong, then the check fails. In addition to data filtering, it is therefore necessary to specify *Concept.Requirements*, i.e. to differentiate between mandatory and excluded data, and to restrict possible values of instances by using *TemplateRule.Parameters* of a *Concept* and *Constraint.Expression* of *AttributeRule* and *EntityRule* used by *ConceptTemplate*. Similar to generation of subset schemas and data filtering no documentation or other meta-data is required for this use case.

6 Glossary

6.1 ER

ER = Exchange Requirement; defines the data that is needed to fulfil a specific task. If not all mandatory data is available then the task cannot be carried out. Exchange requirement definitions are independent from a technical solution.

6.2 ERM

ERM = Exchange Requirement Model; implementation of an Exchange Requirement by an IT specification.

6.3 MVD

MVD = Model View Definition; defines a subset of an IT specification that is able to store data for a set of Exchange Requirements.

7 Examples

This section shows a series of small mvdXML example files, used for the different purposes for which the mvdXML standard is developed. Given examples are currently covering the two main use cases: MVD documentation and Data validation.

7.1 Example for MVD documentation

The following example shows an mvdXML file to be used for documenting a model view definition, it is a direct output of the IFC document generator ifcDoc.

From this mvdXML file, in conjunction with the IFC schema definition, the following output is rendered.

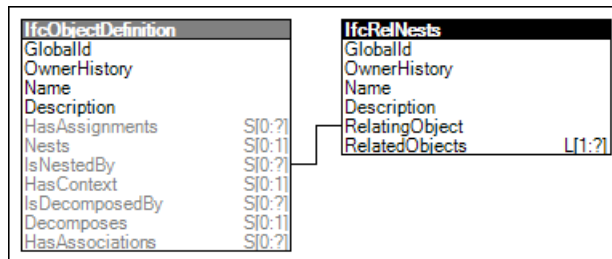


Figure 5: Graphical representation of the concept template "Nesting"

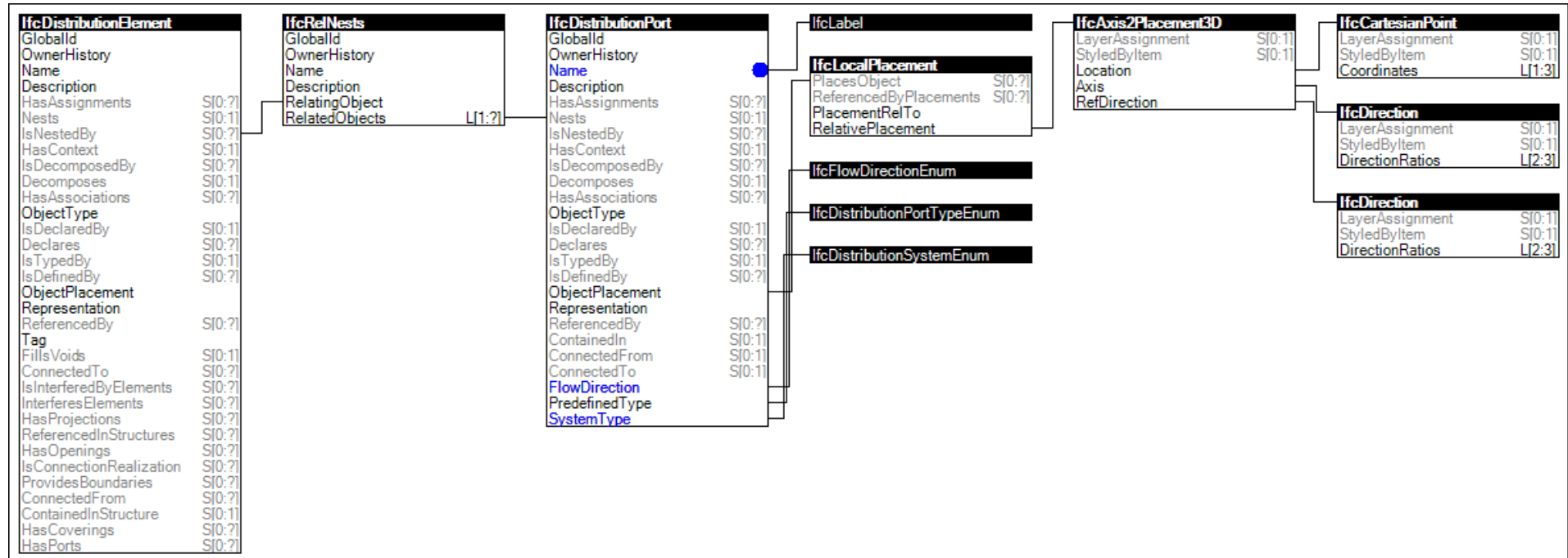


Figure 6: graphical representation of the concept template “port nesting”

Property Sets for Objects

The Property Sets for Objects concept template applies to this entity as shown in Table 552.

PsetName	Properties			
Pset_HeatExchangerTypeCommon	Template	PropertyName	Value	Reference
	Single Value	Reference	IfcIdentifier	
	Enumerated Value	Status	IfcLabel	
	Enumerated Value	Arrangement	IfcLabel	

Table 552 — IfcHeatExchanger Property Sets for Objects

Figure 7: rendering of HTML tables to document the exchange requirements for the different ports

Page no.	Authors
48	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

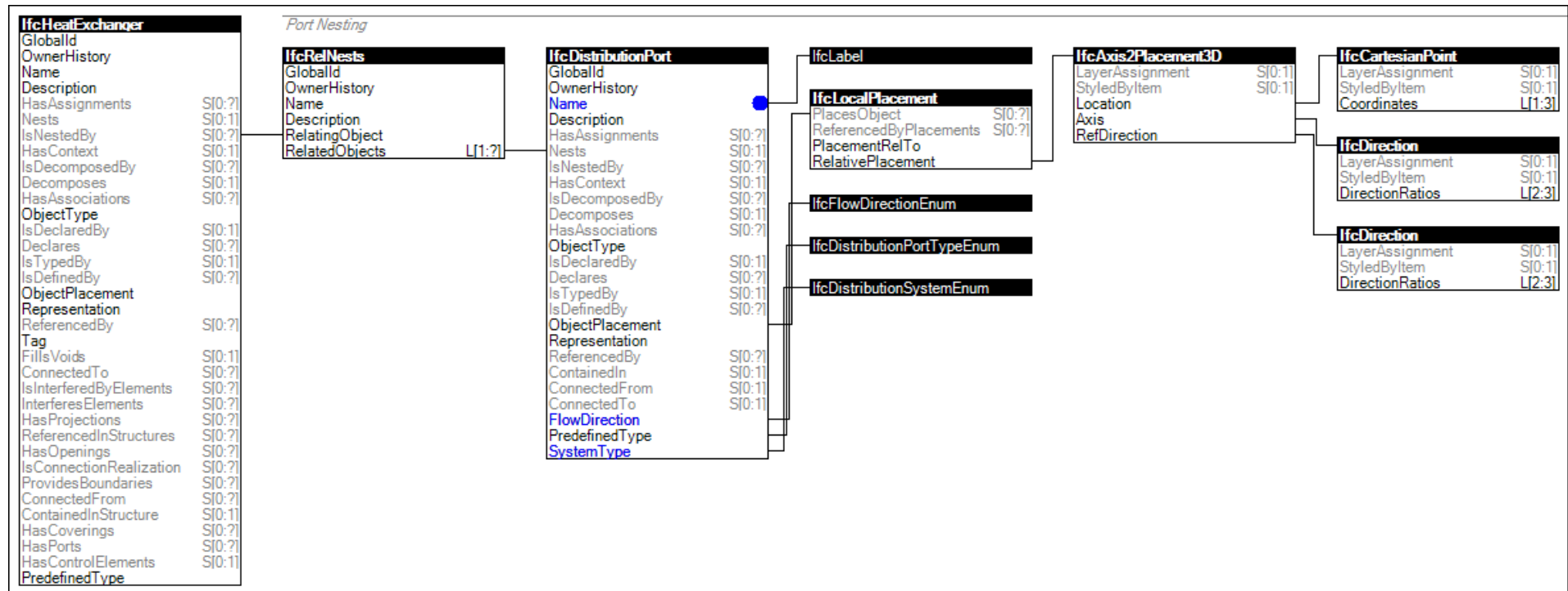


Figure 8: graphical representation of the usage of concept "port-nesting" at the root concept IfcHeatExchanger

NOTE: The following example was already part of the mvdXML 1.1 documentation and still valid in mvdXML 1.2. No changes are needed.

```
<?xml version="1.0"?>
<mvdXML xmlns="https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/" uuid="cb830d34-5696-4263-a7e9-2259ea343117" name="example 7.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2 https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/mvdXML_V1-2.xsd">
  <Templates>
    <ConceptTemplate uuid="5098cd13-bf4b-473a-a846-a60f69e9b738" name="Object Composition" code="" applicableSchema="IFC4"
      applicableEntity="IfcObjectDefinition">
      <Definitions>
        <Definition>
```

```

    <Body><![CDATA[
<p>Objects may be composed into parts to indicate levels of detail, such as a building having multiple storeys, a framed wall having studs, or a task having subtasks. Composition may form a hierarchy of multiple levels, where an object must have a single parent, or if a top-level object then declared within the single project or a project library.</p>
]]></Body>
</Definition>
</Definitions>
<SubTemplates>
  <ConceptTemplate uuid="d1e6b86e-7658-443c-b708-86b7dd8b12f4" name="Nesting" applicableSchema="IFC4" applicableEntity="IfcObjectDefinition">
    <Definitions>
      <Definition>
        <Body><![CDATA[<p>A nesting indicates an external ordered part composition relationship between the hosting structure, referred to as the "host", and the attached components, referred to as the "hosted elements". The concept of nesting is used in various ways. Examples are:</p>
<ul> <li>Nesting is used on product elements to indicate external connectable parts such as faucets mounted on a sink, or switches within a junction box.</li> <li>Nesting is used on control objects to indicate specification hierarchies.</li> <li>Nesting is used on process objects to indicate subordinate processes which may occur in parallel or in series.</li> <li>Nesting is used on resource objects to indicate subordinate resource allocations which may occur in parallel or in series.</li> </ul> <p>Nesting is a bi-directional relationship, the relationship from the hosting structure to its attached components is called Nesting, and the relationship from the components to their containing structure is called Hosting.</p>]]></Body>
        </Definition>
      </Definitions>
      <Rules>
        <AttributeRule AttributeName="IsNestedBy">
          <EntityRules>
            <EntityRule EntityName="IfcRelNests" />
          </EntityRules>
        </AttributeRule>
      </Rules>
    </SubTemplates>
    <ConceptTemplate uuid="bafc93b7-d0e2-42d8-84cf-5da20ee1480a" name="Port Nesting" code="" applicableSchema="IFC4" applicableEntity="IfcDistributionElement">
      <Definitions>
        <Definition>
          <Body><![CDATA[<p>Ports indicate possible connections to other objects according to specified system types, flow direction, and connection properties. Ports are typically connected between devices via cables, pipes, or ducts.</p> <p>Ports may have placement defined indicating the position and outward orientation of the port relative to the product or product type. Ports may also have material profile sets defined indicating the flow area and connection enclosure.</p>]]></Body>
          </Definition>
        </Definitions>
        <Rules>
          <AttributeRule RuleID="PredefinedType" AttributeName="PredefinedType" />
        </Rules>
      </ConceptTemplate>
    </SubTemplates>
  </ConceptTemplate>
</SubTemplates>

```

Page no.	Authors
50	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```
<AttributeRule AttributeName="IsNestedBy">
  <EntityRules>
    <EntityRule EntityName="IfcRelNests">
      <AttributeRules>
        <AttributeRule AttributeName="RelatedObjects">
          <EntityRules>
            <EntityRule EntityName="IfcDistributionPort">
              <AttributeRules>
                <AttributeRule RuleID="Name" Description="The name of the port." AttributeName="Name">
                  <EntityRules>
                    <EntityRule EntityName="IfcLabel" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule RuleID="Flow" Description="The flow direction of the port." AttributeName="FlowDirection">
                  <EntityRules>
                    <EntityRule EntityName="IfcFlowDirectionEnum" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule RuleID="Type" AttributeName="SystemType">
                  <EntityRules>
                    <EntityRule EntityName="IfcDistributionSystemEnum" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule AttributeName="ObjectPlacement">
                  <EntityRules>
                    <EntityRule EntityName="IfcLocalPlacement">
                      <AttributeRules>
                        <AttributeRule AttributeName="RelativePlacement">
                          <EntityRules>
                            <EntityRule EntityName="IfcAxis2Placement3D">
                              <AttributeRules>
                                <AttributeRule AttributeName="Location">
                                  <EntityRules>
                                    <EntityRule EntityName="IfcCartesianPoint" />
                                  </EntityRules>
                                </AttributeRule>
                                <AttributeRule AttributeName="Axis">
                                  <EntityRules>
                                    <EntityRule EntityName="IfcDirection" />
                                  </EntityRules>
                                </AttributeRule>
                              </AttributeRules>
                            </EntityRule>
                          </EntityRules>
                        </AttributeRule>
                      </AttributeRules>
                    </EntityRule>
                  </EntityRules>
                </AttributeRule>
              </AttributeRules>
            </EntityRule>
          </EntityRules>
        </AttributeRule>
      </EntityRules>
    </EntityRule>
  </EntityRules>
</AttributeRule>
```

Page no.	Authors
51	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

        </EntityRules>
      </AttributeRule>
      <AttributeRule AttributeName="RefDirection">
        <EntityRules>
          <EntityRule EntityName="IfcDirection" />
        </EntityRules>
      </AttributeRule>
    </AttributeRules>
  </EntityRule>
</EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
<AttributeRule AttributeName="IsDeclaredBy" />
<AttributeRule AttributeName="PredefinedType">
  <EntityRules>
    <EntityRule EntityName="IfcDistributionPortTypeEnum" />
  </EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
</Rules>
</ConceptTemplate>
</SubTemplates>
</ConceptTemplate>
</SubTemplates>
</ConceptTemplate>
</Templates>
<Views>
  <ModelView uuid="dae06832-07d3-4b1c-b4a7-ee32e11d0189" name="HVAC Sample Model View" code="HVAC" applicableSchema="IFC4">
    <ExchangeRequirements>

```

Page no.	Authors
52	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    <ExchangeRequirement uuid="a5846830-de9a-4195-9339-31169ecb7b0e" name="HVAC Sample Exchange" applicability="both" />
  </ExchangeRequirements>
  <Roots>
    <ConceptRoot uuid="3ca6a49f-81c4-4010-9589-578cab9d4428" name="" applicableRootEntity="IfcHeatExchanger">
      <Concepts>
        <Concept uuid="875138d3-6911-4e35-9369-2d273f731250" name="Port" override="false">
          <Template ref="bafc93b7-d0e2-42d8-84cf-5da20ee1480a" />
          <Requirements>
            <Requirement applicability="both" requirement="mandatory" exchangeRequirement="a5846830-de9a-4195-9339-31169ecb7b0e" />
          </Requirements>
          <TemplateRules operator="and">
            <TemplateRule Description="Inlet of substance to be heated."
              Parameters="Name[Value]='HeatingInlet' AND Flow[Value]='SINK' AND Type[Value]='NOTDEFINED'" />
            <TemplateRule Description="Outlet of substance to be heated."
              Parameters="Name[Value]='HeatingOutlet' AND Flow[Value]='SOURCE' AND Type[Value]='NOTDEFINED'" />
            <TemplateRule Description="Inlet of substance to be cooled."
              Parameters="Name[Value]='CoolingInlet' AND Flow[Value]='SINK' AND Type[Value]='NOTDEFINED'" />
            <TemplateRule Description="Outlet of substance to be cooled."
              Parameters="Name[Value]='CoolingOutlet' AND Flow[Value]='SOURCE' AND Type[Value]='NOTDEFINED'" />
          </TemplateRules>
        </Concept>
      </Concepts>
    </ConceptRoot>
  </Roots>
</ModelView>
</Views>
</mvdXML>

```

7.2 Examples for MVD validation

The following chapter includes various examples for the model checking use case. The first example is an update of an example that was already included in the mvdXML 1.1 documentation. All other examples have been added to explain new features of mvdXML 1.2.

Page no.	Authors
53	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

7.2.1 Checking IFC Properties on load bearing, external walls

The following example shows an mvdXML file to be used to validate the completeness of IFC data in an IFC file. It checks that for every load bearing and external wall the following is true: a property set “*Pset_WallCommon*” with the properties “*FireRating*” and “*ThermalTransmittance*” are provided, and that the direct optional attribute “*PredefinedType*” is given. It is permissible to either assign the properties on the occurrence object or at the type object.

Therefore a *ConceptTemplate* "Property Sets for Objects and Types" is included, that declares that part of the overall IFC structure, which is used to assign a property set to an occurrence object, a type object to an occurrence object, and a property set to the type object. Since the definition of property set (and the referenced definition of the property) is used twice (for occurrence and type object) it is declared as a partial concept template and referenced twice from the main concept template.

As the stipulated completeness checks for “*FireRating*”, “*ThermalTransmittance*” and “*PredefinedType*” are only applicable to those walls, represented by *IfcWall* or its subtypes, that are external and load bearing, a separate applicability check is performed as a precondition, before validating the rules themselves.

Each rule checking for the provision of the properties “*FireRating*” and “*ThermalTransmittance*” need to check, whether they are assigned to the occurrence (instance of *IfcWall*) or the type (associated instance of *IfcWallType*). A recursive structure of *TemplateRules* with a @operator attribute is used to hold the logical combinations.

NOTE: The following example was already part of the mvdXML 1.1 documentation. Minor adjustments in the configuration part were necessary to be compatible with mvdXML 1.2. Those changes are further documented in the text. With mvdXML 1.2 this example could have been further simplified by reusing existing standard templates instead of creating a new template for checking occurrence and type properties. Such simplification is shown in later example provided in 7.2.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<mvdXML xmlns="https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/" uuid="8a70d456-c609-4ef7-b496-b92fd1e12796" name="example 7.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2 https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-2/mvdXML_V1-2.xsd">
  <Templates>
```

The concept template "Property Sets for Objects and Types" defined the concept template structure about how to associate property sets and type objects with property sets to an occurrence object.

```
<ConceptTemplate uuid="5c252c86-5bff-4372-9a27-b794069f9fbb" name="Property Sets for Objects and Types" applicableSchema="IFC4"
  applicableEntity="IfcObject">
  <Rules>
```

Page no.	Authors
54	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

<AttributeRule RuleID="PredefinedType" AttributeName="PredefinedType"/>
<AttributeRule AttributeName="IsDefinedBy">
  <EntityRules>
    <EntityRule EntityName="IfcRelDefinesByProperties">
      <AttributeRules>
        <AttributeRule AttributeName="RelatingPropertyDefinition">
          <EntityRules>
            <EntityRule EntityName="IfcPropertySet">
              <References IdPrefix="O_">
                <Template ref="7c4c45c5-7ba9-4e19-b473-3e97093b3e0d"/>
              </References>
            </EntityRule>
          </EntityRules>
        </AttributeRule>
      </AttributeRules>
    </EntityRule>
  </EntityRules>
</AttributeRule>

```

Here a partial concept template is referenced. In order to prevent a duplication of *@RuleID* names, which would otherwise occur, when the same partial template “Property Set” is referenced twice from the same main concept template, a *@IdPrefix* attribute is added. The *TemplateRule* Parameters at the *Concept* need to use these prefixes to unambiguously address the rule id.

```

    </EntityRule>
  </EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
</AttributeRule>
<AttributeRule AttributeName="IsTypedBy">
  <EntityRules>
    <EntityRule EntityName="IfcRelDefinesByType">
      <AttributeRules>
        <AttributeRule AttributeName="RelatingType">
          <EntityRules>
            <EntityRule EntityName="IfcTypeObject">
              <AttributeRules>
                <AttributeRule AttributeName="HasPropertySets">
                  <EntityRules>
                    <EntityRule EntityName="IfcPropertySet">
                      <References IdPrefix="T_">
                        <Template ref="7c4c45c5-7ba9-4e19-b473-3e97093b3e0d"/>
                      </References>
                    </EntityRule>
                  </EntityRules>
                </AttributeRule>
              </AttributeRules>
            </EntityRule>
          </EntityRules>
        </AttributeRule>
      </AttributeRules>
    </EntityRule>
  </EntityRules>
</AttributeRule>

```

Here the partial concept template for property sets is referenced a second time. Therefore a different *@IdPrefix* is used.

Page no.	Authors
55	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    </EntityRule>
  </EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
</AttributeRule>
</AttributeRules>
</EntityRule>
</EntityRules>
</AttributeRule>
</Rules>
</ConceptTemplate>

```

The following concept templates are partial concept templates used to allow for a more modular structure of concept template definitions. It includes the definition of property sets and the definition of a property with single value, which is referenced from the partial concept template for property sets. Hence partial concept templates can be nested.

```

<ConceptTemplate uuid="6655f6d0-29a8-47b8-8f3d-c9fce9c9a620" name="Single Value" applicableSchema="IFC4"
  applicableEntity="IfcPropertySingleValue" isPartial="true">
  <Rules>
    <AttributeRule RuleID="PName" AttributeName="Name">
      <EntityRules>
        <EntityRule EntityName="IfcIdentifier"/>
      </EntityRules>
    </AttributeRule>
    <AttributeRule AttributeName="Description">
      <EntityRules>
        <EntityRule EntityName="IfcText"/>
      </EntityRules>
    </AttributeRule>
    <AttributeRule RuleID="PSingleValue" AttributeName="NominalValue">
      <EntityRules>
        <EntityRule EntityName="IfcValue"/>
      </EntityRules>
    </AttributeRule>
  </Rules>
</ConceptTemplate>
<ConceptTemplate uuid="7c4c45c5-7ba9-4e19-b473-3e97093b3e0d" name="Property Sets" code="" applicableSchema="IFC4"
  applicableEntity="IfcPropertySet" isPartial="true">

```

Page no.	Authors
56	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;


```
<Rules>
  <AttributeRule RuleID="PsetName" AttributeName="Name">
    <EntityRules>
      <EntityRule EntityName="IfcLabel"/>
    </EntityRules>
  </AttributeRule>
  <AttributeRule AttributeName="Description">
    <EntityRules>
      <EntityRule EntityName="IfcText"/>
    </EntityRules>
  </AttributeRule>
  <AttributeRule AttributeName="HasProperties">
    <EntityRules>
      <EntityRule EntityName="IfcPropertySingleValue">
        <References>
          <Template ref="6655f6d0-29a8-47b8-8f3d-c9fce9c9a620"/>
        </References>
      </EntityRule>
    </EntityRules>
  </AttributeRule>
</Rules>
</ConceptTemplate>
</Templates>
<Views>
  <ModelView uuid="72dad5df-6f61-49f2-ba8c-baccf24a6ce5" name="design phase" applicableSchema="IFC4" code="LPH 3">
    <Definitions>
      <Definition>
        <Body lang="de"><![CDATA[Model progression requirements for design phase]]></Body>
      </Definition>
    </Definitions>
    <ExchangeRequirements>
      <ExchangeRequirement uuid="ae70f764-938b-4cf7-9814-c29a47f56b0e" name="design phase coordination" code="LPH 3a" applicability="export">
        <Definitions>
          <Definition>
            <Body lang="de"><![CDATA[Model progression requirements for design phase for coordination.]]></Body>
          </Definition>
        </Definitions>
      </ExchangeRequirement>
    </ExchangeRequirements>
  </ModelView>
</Views>
```

Page no.	Authors
57	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```
<Roots>
  <ConceptRoot uuid="0e93f597-f5e1-475b-87a7-eb007993a50d" name="load bearing external walls" applicableRootEntity="IfcWall">
    <Definitions>
      <Definition>
        <Body lang="de"><![CDATA[...]]></Body>
      </Definition>
    </Definitions>
  </ConceptRoot>
```

This concept has a precondition that needs to be met before the template rules are executed. The Applicability imposes that only those instances of *IfcWall* are validated, that have the property set “*Pset_WallCommon*” and the two properties “*IsExternal*” and “*LoadBearing*” assigned:

- The value of “*IsExternal*” shall be “true” and the value of “*LoadBearing*” shall be “true”.

In addition, it checks, whether the properties are provided at the occurrence or at the type, and if both are provided, that the override value from the occurrence is used for the check.

NOTE: Since mvdXML 1.2 the operator “NOT” in *TemplatesRules* is no longer supported. The updated example replaces such definitions by the *negation* attribute in *TemplateRule*. Instead of using the negation attribute of *TemplateRules*, this logic could also be embedded in the logical expression of *Parameters* by using the new NOT statement.

```
<Applicability>
  <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
  <!-- Applicability: check that IsExternal and LoadBearing property are both set to true (AND) -->
  <TemplateRules operator="and">
    <!-- two alternatives to provide the IsExternal property (as property on occurrence or type) (OR) -->
    <TemplateRules operator="or">
      <!-- check occurrence property -->
      <TemplateRule Parameters="O_PsetName[Value]='Pset_WallCommon' AND O_PName[Value]='IsExternal' AND O_PSingleValue[Value]=TRUE"/>
      <!-- for type properties two criteria must be checked: 1) defined on type and 2) not redefined on occurrence -->
      <TemplateRules operator="and">
        <TemplateRule Parameters="T_PsetName[Value]='Pset_WallCommon' AND T_PName[Value]='IsExternal' AND T_PSingleValue[Value]=TRUE"/>
        <TemplateRule negation=true Parameters="O_PsetName[Value]='Pset_WallCommon' AND O_PName[Value]='IsExternal'"/>
      </TemplateRules>
    </TemplateRules>
  </TemplateRules operator="or">
    <TemplateRule Parameters="O_PsetName[Value]='Pset_WallCommon' AND O_PName[Value]='LoadBearing' AND O_PSingleValue[Value]=TRUE"/>
    <!-- for type properties two criteria must be checked: 1) defined on type and 2) not redefined on occurrence -->
    <TemplateRules operator="and">
      <TemplateRule Parameters="T_PsetName[Value]='Pset_WallCommon' AND T_PName[Value]='LoadBearing' AND T_PSingleValue[Value]=TRUE"/>
    </TemplateRules>
  </TemplateRules>
```

Page no.	Authors
58	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    <TemplateRule negation=true Parameters="O_PsetName[Value]='Pset_WallCommon' AND O_PName[Value]='LoadBearing'"/>
  </TemplateRules>
</TemplateRules>
</TemplateRules>
</Applicability>

```

At the concepts the validation rules for the required provision of “*FireRating*”, “*ThermalTransmittance*” and “*PredefinedType*” are defined. There are three individual concepts, each of them is validated separately.

In this example, they all refer to the same concept template “Property Sets for Objects and Types” via the IDREF link “5c252c86-5bff-4372-9a27-b794069f9fbb”. The validation is enforced for the exchange requirement “design phase coordination” via the IDREF link “ae70f764-938b-4cf7-9814-c29a47f56b0e”, the requirement is set to “mandatory”, meaning, that an error is displayed, if the outermost template rule validates to “false”.

```

<Concepts>
  <!-- Test #1: check existence of FireRating property -->
  <Concept uuid="983ddc5d-c0c8-47c9-8491-97add7677139" name="load bearing external walls required to have property 'FireRating'">
    <Definitions>
      <Definition>
        <Body lang="de"><![CDATA[For all load bearing external walls the property 'FireRating' shall be applied]]></Body>
      </Definition>
    </Definitions>
    <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
    <Requirements>
      <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
    </Requirements>
    <TemplateRules operator="or">
      <TemplateRule Parameters="O_PsetName[Value]='Pset_WallCommon' AND O_PName[Value]='FireRating' AND O_PSingleValue[Exists]=TRUE"/>
      <TemplateRule Parameters="T_PsetName[Value]='Pset_WallCommon' AND T_PName[Value]='FireRating' AND T_PSingleValue[Exists]=TRUE"/>
    </TemplateRules>
  </Concept>

```

The first concept checks, that every instance of *IfcWall*, that passes the *Applicability* check (meaning that is an external and load bearing wall), has a property of name “*FireRating*” within a property set with name “*Pset_WallCommon*”. Thereby it can either be the property set assigned to the occurrence, or the property set assigned to the type, or both. The variable “*O_PsetName*” refers to the *@RuleID* “*PsetName*” defined in the partial concept template that has been referenced by the main concept template “Property Sets for Objects and Types” with the *@IdPrefix* “*O_*” – i.e. to the property set assigned directly to the *IfcWall* occurrence. Similarly the variable “*T_PsetName*” refers to the property set assigned to the associated *IfcWallType*.

```

  <!-- Test #2: check existence of ThermalTransmittance property -->
  <Concept uuid="e9941408-82a6-4c00-a397-11087e6c5d1f" name="load bearing external walls required to have property 'ThermalTransmittance'">

```

Page no.	Authors
59	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

<Definitions>
  <Definition>
    <Body lang="de"><![CDATA[For all load bearing external walls the property 'ThermalTransmittance' shall be applied]]></Body>
  </Definition>
</Definitions>
<Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
<Requirements>
  <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
</Requirements>
<TemplateRules operator="or">
  <TemplateRule Parameters="O_PsetName[Value]='Pset_WallCommon' AND
    O_PName[Value]='ThermalTransmittance' AND O_PSingleValue[Exists]=TRUE"/>
  <TemplateRule Parameters="T_PsetName[Value]='Pset_WallCommon' AND
    T_PName[Value]='ThermalTransmittance' AND T_PSingleValue[Exists]=TRUE"/>
</TemplateRules>
</Concept>

```

The second concepts checks the provision of the property “*ThermalTransmittance*” using the same method.

```

<!-- Test #3: check existence of PredefinedType attribute -->
<Concept uuid="a14ab957-e65d-48c1-84fe-8f99c2630646" name="load bearing external walls required to have attribute PredefinedType">
  <Definitions>
    <Definition>
      <Body lang="de"><![CDATA[For all load bearing external walls the property 'PredefinedType' shall be applied]]></Body>
    </Definition>
  </Definitions>
  <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
  <Requirements>
    <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
  </Requirements>
  <TemplateRule Parameters="PredefinedType[Exists]=TRUE"/>
</Concept>

```

The third concept checks, whether the direct, optional attribute *PredefinedType* at *ifcWall* has a value associated. The metric “[Exists]” checks, that the value of *PredefinedType* is not NIL or false (in other words, that a value is provided for the optional attribute).

NOTE: In mvdXML 1.2 a *TemplatesRules* element must include 2 or more *TemplateRule* elements. The check of the *PredefinedType* attribute is thus no longer encapsulated by a *TemplatesRules* element.

```
</Concepts>
```

Page no.	Authors
60	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    </ConceptRoot>
  </Roots>
</ModelView>
</Views>
</mvdXML>

```

7.2.2 Checking existence of objects with error messages

This example explains two new features introduced with mvdXML 1.2: (1) the ability to check the existence of specific object types and (2) the possibility to include user-friendly messages in case of success, errors, warnings or other informal purposes.

The example checks that at least one instance of *IfcSpace* is available. This requirement is expressed by the *minOccurrence* attribute in the *Applicability* element, which also includes the error messages in English and German.

```

<ConceptRoot uuid="28443da6-000b-40df-9c6f-62553300f55f" name="Test IfcSpace availability" applicableRootEntity="IfcSpace">
  <!-- There must be at least one instance of IfcSpace (minOccurrence) -->
  <Applicability minOccurrence="1">
    <!-- Define error message in English and German if the applicability rule fails -->
    <RuleMessage lang="en" state="failure">At least one room (IfcSpace) is required</RuleMessage>
    <RuleMessage lang="de" state="failure">Mindestens ein Raum (IfcSpace) muss vorhanden sein</RuleMessage>
  </Applicability>
</ConceptRoot>

```

7.2.3 Use of SELF expression for defining exception from list of subtypes

This example shows an extension of the rule grammar. The keyword **SELF** is used to check the type of an instance. In the given example it is used to exclude the subtypes *IfcAnnotation* and *IfcVirtualElement* of *IfcProduct* from the list of applicable entities for checking the existence of the name attribute.

```

<ConceptRoot uuid="2aa33a62-6f0e-4d03-9f00-ba1796c62f1e" name="Test name of IfcProduct (without IfcAnnotation, IfcVirtualElement)"
applicableRootEntity="IfcProduct">
  <Applicability>
    <Template ref="6d70aadb-bd68-41b3-8b0f-0b56c5c09e86"/>
    <!-- Define exceptions for the selection of instances via SELF statement. Exclude instances of type IfcAnnotation and IfcVirtualElement -->
    <TemplateRule Parameters="SELF[Type]!='IfcAnnotation' and SELF[Type]!='IfcVirtualElement'"/>
  </Applicability>
  <Concepts>
    <Concept uuid="5fc09487-885f-4185-98e2-cb3ba2386087" name="Name exists">
      <Template ref="6d70aadb-bd68-41b3-8b0f-0b56c5c09e86"/>
      <TemplateRule Parameters="Name[Exists]=TRUE"/>
    </Concept>
  </Concepts>
</ConceptRoot>

```

Page no.	Authors
61	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    </Concept>
  </Concepts>
</ConceptRoot>

```

7.2.4 Combination of ConceptTemplate

The combination of different *ConceptTemplates* enables to focus on templates with basic IFC functionality. If model checking needs to test alternative representations or to combine preconditions linked to different IFC functionalities, then such combination of *ConceptTemplates* can help to avoid defining new *ConceptTemplates*. There are a few examples in the IFC4 and IFC4 RV1.2 specifications showing such combination of basic IFC functionalities, leading to the need to define new ConceptTemplates. This includes:

- “Property Sets with Override”: combination of “Property Sets for Objects” and “Property Sets for Types” via “Object Typing”
- “Classification for Objects with Override” (in RV only): combination of generic “Classification for Objects” and generic “Classification” via “Object Typing”
- “Material Single for Objects with Override” (in RV only): combination of generic “Material Single” via “Object Typing”
- “Material Constituent Set with Override” (in RV only): combination of generic “Material Constituent Set” via “Object Typing”

The principle of combining several ConceptTemplates is shown for the first override specification. Instead of using the new ConceptTemplate “Property Sets with Override” a combination of “Property Sets for Objects”, “Property Sets for Types” and “Object Typing” can be used in mvdXML 1.2.

```

<Concept uuid="00000168-1162-1163-0000-000000171167" name="LoadBearing : Property Sets for Objects : Beam_1-04">
  <!-- Reference to one override template "Property Sets for Objects with Override"
    <Template ref="e26040e8-82e2-4f6a-bc63-ac8e6da2d0ae"/> -->
  <!-- Reference to "Property Sets for Objects" -->
  <Template ref="f74255a6-0c0e-4f31-84ad-24981db62461"/>
  <!-- Reference to "Object Typing" -->
  <Template ref="35a2e10e-20df-40f4-ab2f-dacf0a6744f4"/>
  <!-- Reference to "Property Sets for Types", pointing to "Object Typing" as base template and the RuleID where this template shall be used -->
  <Template ref="4be3312a-3199-492e-b204-cac2229a9ade" IdPrefix="Type" applicableEntityRuleID="RelatingType"
    baseTemplate="35a2e10e-20df-40f4-ab2f-dacf0a6744f4"/>
  <Requirements>
    <Requirement applicability="import" exchangeRequirement="00000168-1162-1163-0000-000000000000" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="or">
    <TemplateRule Parameters="PsetName[Value]='Pset_BeamCommon' AND PropertyName[Value]='LoadBearing' AND Value[Value]=TRUE"/>
    <TemplateRules operator="and">
      <TemplateRule Parameters="TypePsetName[Value]='Pset_BeamCommon' AND TypePropertyName[Value]='LoadBearing' AND TypeValue[Value]=TRUE"/>
    </TemplateRules>
  </TemplateRules>
</Concept>

```

Page no.	Authors
62	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    <TemplateRule negation="true" Parameters="PsetName[Value]='Pset_BeamCommon' AND PropertyName[Value]='LoadBearing' AND
Value[Exists]=TRUE"/>
  </TemplateRules>
</TemplateRules>
</Concept>

```

7.2.5 Check existence of a particular information for all material layers

The following advanced example shows how to check a set of references and the use of an additional condition. Such type of checking rule is for instance needed if material properties of a multilayered wall must be checked. A set of material properties must be provided for all layers except for the ventilation layer.

The first example defines a check for material layer set usage (exclude ventilation layer).

```

<Concept uuid="d5fe9c9c-258b-4f1a-b6a1-ec987bb22e4a" name="Check material properties for building elements">
  <!-- link to template "Material Profile Set Usage", extended by link to properties (IfcMaterial(Layer).HasProperties) -->
  <Template ref="39a4420d-ab6f-4dd1-b4ce-a4a85bd2bd13"/>
  <Requirements>
    <Requirement applicability="import" exchangeRequirement="92f50b6b-557f-4e5a-9b27-7034e7b6f3b6" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="and">
    <!-- optional check for existence of MaterialRelation -->
    <TemplateRule Parameters="MaterialRelation[Size]=1"/>
    <TemplateRules operator="and">
      <!-- MaterialLayerSetUsage must be used (may combine with other alternatives) !-->
      <TemplateRule Parameters="RelatingMaterialLayerSetUsage[Type]='IfcMaterialLayerSetUsage'"/>
      <TemplateRules operator="and">
        <!-- LayerSetName must be given -->
        <TemplateRule Parameters="LayerSetName[Exists]=TRUE"/>
        <!-- 2 or more layers should exist -->
        <TemplateRule Parameters="MaterialLayers[Size]>1"/>
        <TemplateRule operator="and">
          <!-- Conditional statement for execution of template rules; check should be performed only,
            if "IsVentilated" of IfcMaterialLayers equals to false, i.e. no material properties are required if material layer
            represents an air gap minOccurrence=2 there must be at least two layers that are not an air gap maxOccurrence is not
            given, the condition should apply FOR ALL layers that are not an air gap -->
          <Applicability minOccurrence="2">
            <!-- If no template ref is given, use the templates as defined by the Concept
              <Template ref="39a4420d-ab6f-4dd1-b4ce-a4a85bd2bd13"/> -->
            <TemplateRule Parameters="IsVentilated[Value]=FALSE"/>
          </Applicability>
          <!-- If MaterialLayer is not an air gap (controlled by IsVentilated attribute), check that MaterialName exists and .. -->
          <TemplateRule Parameters="MaterialName[Exists]=TRUE"/>
          <!-- .. the following properties are defined and that the values are greater than 0.0
            NOTE: Parameters PropertySetName, SimplePropertyName and NominalValue must be defined in the used template

```

Page no.	Authors
63	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```

    "Material Constituent Set with Override"! -->
    <TemplateRules operator="and">
      <TemplateRule Parameters="PropertySetName[Value]='Pset_MaterialThermal' and
        SimplePropertyName[Value]='ThermalConductivity' and
        NominalValue[Value] > 0.0"/>
      <TemplateRule Parameters="PropertySetName[Value]='Pset_MaterialThermal' and
        SimplePropertyName[Value]='SpecificHeatCapacity' and
        NominalValue[Value] > 0.0"/>
      <TemplateRule Parameters="PropertySetName[Value]='Pset_MaterialCommon' and
        SimplePropertyName[Value]='MassDensity' and
        NominalValue[Value] > 0.0"/>
    </TemplateRules>
  </TemplateRule>
</TemplateRules>
</TemplateRules>
</Concept>

```

The second example is like the first example but defines the checking rule for testing material constituent sets instead of material layers. The assumption in that example is that by agreement the ventilation layer shall be identified by the material category “Air” and will be excluded from the requirement of having a above requested properties.

```

<!-- check for multilayered walls defined by material constituents sets (with exception for air gaps based on IfcMaterial.Category = 'Air')-->
<Concept uuid="d5fe9c9c-258b-4f1a-b6a1-ec987bb22e4a" name="Check material properties for layers building elements">
  <!-- link to template "Material Constituent Set with Override" -->
  <Template ref="b31e8fb4-e7b9-48da-bbf4-cfd422f5f6cd"/>
  <Requirements>
    <Requirement applicability="import" exchangeRequirement="92f50b6b-557f-4e5a-9b27-7034e7b6f3b6" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="and">
    <TemplateRule Parameters="MaterialRelation[Size]=1"/>
    <TemplateRules operator="and">
      <TemplateRule Parameters="MaterialConstituents[Type]='IfcMaterialConstituent'"/>
      <TemplateRule Parameters="MaterialConstituents[Size]>1"/>
      <TemplateRules operator="and">
        <Applicability minOccurrence=1>
          <TemplateRule Parameters="MaterialCategory[Value]!='Air'"/>
        </Applicability>
        <TemplateRule Parameters="ConstituentName[Exists]=TRUE"/>
        <TemplateRule Parameters="Material[Exists]=TRUE AND Material[Type]='IfcMaterial'"/>
        <TemplateRule Parameters="MaterialProperties[Size]>2"/>
        <TemplateRule Parameters="MaterialPropertyName[Exists]=TRUE"/>
      </TemplateRules>
    <TemplateRule Parameters="PropertySetName[Value]='Pset_MaterialThermal' and

```

Page no.	Authors
64	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;


```

        SimplePropertyName[Value]='ThermalConductivity' and
        NominalValue[Value] > 0.0"/>
      <TemplateRule Parameters="PropertySetName[Value]='Pset MaterialThermal' and
        SimplePropertyName[Value]='SpecificHeatCapacity' and
        NominalValue[Value] > 0.0"/>
      <TemplateRule Parameters="PropertySetName[Value]='Pset_MaterialCommon' and
        SimplePropertyName[Value]='MassDensity' and
        NominalValue[Value] > 0.0"/>
    </TemplateRules>
  </TemplateRules>
</TemplateRules>
</TemplateRules>
</Concept>

```

7.2.6 Adding information about precision and units

The following example shows the use of precision settings for the comparison of floating-point values, the elevation hight of a building storey.

```

<ConceptRoot uuid="00000168-1162-1163-0000-000000171183" name="Building Storey_02" applicableRootEntity="IfcBuildingStorey">
  <Applicability>
    <Template ref="c19ec186-9cfd-47fc-a4d4-9fb35008d04a"/>
    <TemplateRule Parameters="ObjectName[Value]='First Floor'"/>
  </Applicability>
  <Concepts>
    <Concept uuid="00000168-1162-1163-0000-000000171185" name="Storey Attributes : Building Storey_02">
      <Template ref="488c9bb2-3102-4b89-bd20-42aa7060ae08"/>
      <Requirements>
        <Requirement applicability="import" exchangeRequirement="00000168-1162-1163-0000-000000000000" requirement="mandatory"/>
      </Requirements>
      <TemplateRules operator="and">
        <TemplateRule Parameters="Name[Value]='First Floor'"/>
        <!-- Set relative precision to 0.001 for the check of Elevation value
          (given in global length unit defined in the Default Settings) -->
        <TemplateRule Precision="1e-4" RelativePrecision="true" Parameters="Elevation[Value]=3.8"/>
      </TemplateRules>
    </Concept>
  </Concepts>
</ConceptRoot>

```

The global unit for length measurements can be defined in the *DefaultSettings* element being a child of the *mvdXML* element. In this example the length unit as being relevant for the check of the *Elevation* attribute is defined to be in metre. The precision defined at global level (1e-6) is redefined to be 1e-4 at the *TemplateRule*.

Page no.	Authors
65	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

```
<DefaultSettings precision="1e-6">  
  <Units>  
    <IfcSIUnit UnitType="lengthunit" Name="metre" />  
  </Units>  
</DefaultSettings>
```

Page no.	Authors
66	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;

8 XSD Listing

The XSD can be download from: <https://github.com/buildingSMART/mvdXML/tree/master/mvdXML1.2/xsd>

Page no.	Authors
67	Weise, M.; Geiger A.; Muhic, S; Chipman, T; Liebich, T;