Document Number: DSP0219

Date: 2009-08-11

Version: 1.0.0

UML Profile for CIM

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

Contents

Figures

Tables

Foreword

DSP0219, *UML Profile for CIM*, was prepared by the DMTF Architecture Working Group in collaboration with the Object Management Group (OMG) under the alliance between DMTF and OMG.

Acknowledgments

DMTF acknowledges the following DMTF members for their contributions to this document:

- George Ericson – EMC
- Christoph Fehling – IBM
- Steve Jerman – Cisco
- Andreas Maier – IBM
- Kirk Wilson – CA, Inc.

DMTF acknowledges the following Object Management Group (OMG) members for their contributions to this document:

- Dusko Misic – IBM
- Pete Rivett – Adaptive
- Bran Selic – IBM

Participants from the DMTF Architecture Working Group:

- George Ericson – EMC
- Christoph Fehling – IBM
- Steve Jerman – Cisco
- Andreas Maier – IBM
- Jeff Piazza – Hewlett-Packard Company
- Kirk Wilson – CA, Inc.

Participants from the Object Management Group's Architecture Board:

- Sridhar Iyengar – IBM
- Jishnu Mukerji – Hewlett-Packard Company
- Pete Rivett – Adaptive
- Bran Selic – IBM

Introduction

The Unified Modeling Language (UML) from the Object Management Group (OMG) allows users to specify, visualize and document software systems. It includes twelve diagram types to allow various aspects of a system's design to be modeled. These diagrams are:

- **Structural Diagrams** include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
- **Behavior Diagrams** include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.
- **Model Management Diagrams** include Packages, Subsystems, and Models.

The CIM metamodel defined in the *CIM Infrastructure Specification* (DSP0004) is very similar to the UML metamodel. The DMTF has used the CIM metamodel to define the CIM Schema, which is a management schema that forms an ontology for computer and systems management. The CIM Schema defines a rich and detailed set of classes that establish a common framework for the description of the managed environment. The classes include methods as well as properties and so enable both active management and instrumentation. The CIM Schema is extensible so that vendors can define their own extensions by subclassing. Note that the CIM metamodel could also be used to define CIM models that are not part of the CIM Schema.

The purpose of this document is to formalize the relationship between CIM and UML.

The OMG has defined a four-layer metamodel architecture that is defined in the *OMG MOF Core Specification*. These four metamodel layers are called M0 to M3. Within this architecture, UML is defined as a metamodel at the M2 metamodel level. The CIM metamodel could be defined as another M2 metamodel; however, this would not meet the goal of allowing the use of standard UML tools. Instead, this document utilizes the defined UML extension points to map the CIM metamodel to a UML profile. A UML profile is a lightweight extension mechanism for the UML metamodel. This allows using standard UML modeling tools to develop any CIM models and the CIM schema.

One of the big advantages of using UML tools is that they provide a wealth of capabilities beyond just representing the CIM model. For example, they can be used to describe state diagrams, use cases or interactions between management applications and managed elements.

NOTE: The specific profile defined in this document is called the "UML Profile for CIM". It is a UML profile, not to be confused with DMTF management profiles. As a side note: The term "CIM profile" is ambiguous, is not defined by the DMTF, and should not be used for either kind of profile.

UML Profile for CIM

1 Scope

This document defines a UML profile which expresses the DMTF Common Information Model (CIM) using the OMG Unified Modeling Language (UML).

The specified mapping allows automated conversion between the DMTF defined CIM MOF file format or other equivalent representations of a CIM model and a UML model. This conversion can be done in both directions without loss of information, supporting round-trip engineering.

The mapping also allows creation of an XMI based representation of the UML Profile for CIM, for usage by UML tools. The *UML Superstructure Specification* and the *OMG MOF/XMI Mapping Specification* together define an XMI based file format for UML profiles.

This document is based on the CIM metamodel defined in the *CIM Infrastructure Specification, Version 2.5*. This document supports any CIM schema versions based upon that CIM metamodel.

This document uses the UML metamodel defined in the *UML Superstructure Specification, Version 2.1.1*.

2 Normative References

This document depends on the following other documents. For references to specific versions, only the version cited applies. For references to documents without specific versions, the latest version of the referenced document (including any amendments) applies.

2.1 Approved References

DMTF DSP0004, *CIM Infrastructure Specification 2.5*,
http://www.dmtf.org/standards/published_documents/DSP0004_2.5.pdf

OMG *UML Superstructure Specification, Version 2.1.1*,
http://www.omg.org/cgi-bin/doc?formal/07-02-05

OMG *UML Infrastructure Specification, Version 2.1.1*,
http://www.omg.org/cgi-bin/doc?formal/07-02-06

OMG *UML OCL Specification, Version 2.0*,
http://www.omg.org/cgi-bin/doc?formal/06-05-01

754-1985 IEEE *Standard for Binary Floating-Point Arithmetic*,
http://shop.ieee.org/ieeestore/Product.aspx?product_no=SH10116

IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF* January 2008,
http://www.faqs.org/rfcs/rfc5234.html

2.2 Other References

ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards, Annex H,
http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

3 Terms and Definitions

3.1 General Rules

The key phrases and words *shall*, *shall not*, *should*, *should not*, *may*, *need not*, *can*, *cannot* in this document are to be interpreted as described in Annex H of the ISO rules for the structure and drafting of international standards.

Any text in this document is normative, unless noted otherwise. Any paragraphs starting with "Note …" and any examples are non-normative.

3.2 Definitions Related to CIM Qualifier Values

3.1
default value
The default value for a qualifier as defined in the qualifier declaration (in CIM MOF or in UML).

3.2
inherited value
Defined only for qualifiers with flavor *ToSubclass*. The inherited value is the value in the next superclass (towards the top of the hierarchy) that has this qualifier defined. If there is no such superclass (i.e., the class is a top class), then it is the default value.

Note: As usual in CIM, properties and methods with the same name in a class hierarchy have an inheritance relationship only if they are connected via the *Override* qualifier.

3.3
effective value
The value a qualifier effectively has at a particular level in the class hierarchy, regardless of whether or not it has a value defined at that level.
The effective value is determined as follows:

- If a value for a qualifier is defined at the particular hierarchy level, the effective value is the value defined there.
- Otherwise, for qualifiers with flavor *ToSubclass* the effective value is the inherited value, and for qualifiers with flavor *Restricted* the effective value is the default value.

3.4
defined value
If a qualifier value is specified (or set) for a CIM element, it is said to be *defined* on the CIM element.

3.3 Other Terms and Definitions

3.3.1
CIM namespace
A CIM namespace, as defined in the *CIM Infrastructure Specification*.

3.3.2
covered property
A property in a CIM class that has the same name as a property in one of its subclasses, without being overridden. This is an undesired but possible condition that can arise, for example if an extension schema has added a property to an extension subclass, and later, the base CIM schema adds a property with the same name to a CIM-defined base class. CIM instances then have occurrences of all these properties.

3.3.3

inheritance chain

At the class level, an inheritance chain is the ordered set of a CIM class and all of its base classes. Since in CIM there is only single inheritance, the inheritance chain is a linear set of classes.

At the property and method level, an inheritance chain is the ordered set of overridden properties and methods, respectively.

3.3.4

package path

Ordered set of packages in a package tree from a reference point (usually the top of the tree) to a particular package, along the containing packages. This is very similar to a directory path in a hierarchical file system. UML defines syntax for package paths by concatenating the package names, separated by two colons. Example: "CIM::Device::Network"

3.4 Usage of ABNF

This document uses Augmented BNF with the following exception:

- Rules separated by a bar (|) represent choices. (Instead of using a slash (/) as defined in ABNF).

ABNF defines that any items in rules be concatenated without inserting any implicit whitespace characters between them, so any intended whitespace characters need to be specified explicitly in the ABNF.

ABNF defines that any literal strings and rule names be treated case-insensitively.

4 General Definition of a UML Profile

A UML profile is an extension mechanism that defines how the UML metamodel is tailored for different purposes. This includes the ability to tailor the UML metamodel for different platforms (such as J2EE or .NET) or domains (such as real-time, business process, or resource modeling). The UML profile mechanism is not a first-class extension mechanism in the way that it would allow for modifying the UML metamodel. Rather, it is a lightweight extension mechanism that defines specific extension points for the UML metamodel.

A UML profile is usually defined by means of a UML profile specification (like this document). UML tools are expected to create any artifacts that are needed in order to represent and implement the UML profile. Since starting with UML 2.0, a UML profile is represented by a UML metaclass *Profile*, UML profiles can now be exchanged between UML tools via XMI.

The *Profiles* package in the *UML Superstructure Specification* defines the extension mechanisms supported by UML profiles:

- Stereotypes – Allows to extend UML metaclasses with additional attributes (called "Tagged Values" in earlier UML versions)
- Additional metamodel constraints – allows to define additional constraints between UML metaclasses (For example, that only single inheritance is supported). These constraints can be expressed in natural language or in a formal language such as UML's Object Constraint Language (OCL).

Note: The *Classes* package already defines an extension mechanism, namely the Model Library mechanism which is typically used in UML profile specifications in order to define standard datatypes for the tailored UML metamodel.

This UML profile specification tailors the UML metamodel to represent the CIM metamodel.

5 Definition of the UML Profile for CIM

This clause normatively defines the UML profile for CIM.

The transformation of CIM MOF to a UML user model is fully defined by this clause, with the limitations defined in 5.20. This transformation does not lose any information.

The transformation of a UML user model that has the CIM profile for UML applied to CIM MOF is fully defined by this clause, with the limitations defined in 5.20. This transformation does not lose any information if the UML user model is a *pure CIM model*.

A UML user model that has the CIM profile for UML applied is a *pure CIM model* if it

- contains only instances of the UML metaclasses listed in 5.1 and
- satisfies all constraints defined in 5.17 and
- does not utilize any of the extension points defined in 5.18.

For example, the UML profile for CIM maps CIM association classes to UML association classes and does not use straight UML associations. Therefore, if a UML user model contains straight UML associations, it is not a pure CIM model. A tool or an automated transformation (such as XSLT on the XMI or OMG MOF QVT) could detect the usage of a straight UML association and could convert it into an association class.

As another example, using UML diagrams also causes the UML user model to no longer be a pure CIM model. Note that it makes perfect sense to amend a UML user model with diagrams; however, these diagrams are not representable in CIM MOF.

A UML user model that has the CIM profile for UML applied is a *valid CIM model* if it satisfies all constraints defined in 5.17.

5.1 UML Elements Used

The UML profile for CIM uses the following UML metaclasses, and implicitly all their super-metaclasses. Metaclasses used as attributes or as associated metaclasses of those used directly, are also shown.

From the *UML Superstructure Specification*:

- From the UML::Classes::Kernel package:
  - AggregationKind
  - Class
  - Comment
  - Constraint
  - ElementImport
  - Generalization
  - InstanceSpecification
  - LiteralBoolean
  - LiteralInteger
  - LiteralNull
  - LiteralString
  - LiteralUnlimitedNatural

  - Operation
  - Package
  - PackageImport
  - Parameter
  - ParameterDirectionKind
  - DataType
  - Property (merged with AssociationClasses::Property)
  - Slot
  - VisibilityKind
- From the UML::Classes::AssociationClasses package:
  - AssociationClass
  - Property (merged with Kernel::Property)
- From the UML::Profiles package:
  - Image
  - Profile
  - Stereotype

From the *UML Infrastructure Specification*:

- From the InfrastructureLibrary::Core::Abstractions package:
  - VisibilityKind
- From the InfrastructureLibrary::Core::Basic package:
- From the InfrastructureLibrary::Core::Constructs package:
  - Property

The UML compliance level required by this document is UML L2 plus the Classes::AssociationClasses package.

Sometimes, UML metaclasses are defined in multiple UML packages. (For example, the UML *Class* metaclass has a superclass *Classifier* which is defined in both the *Classes::Kernel* and the *Classes::Dependencies* package.) Such metaclasses usually have a *package merge* relationship that defines that one of the metaclasses merges the other. The package merge concept is defined in 7.3.40 (PackageMerge (from Kernel)) of the *UML Superstructure Specification*. For the UML *Property* metaclass, this means that even though there are two different packages defining a UML *Property* metaclass (*Kernel::Property* and *AssociationClasses::Property*) there is only one variant of the UML *Property* metaclass which is the "merge" of these two.

In this document, any reference to a UML metaclass means the merged variant as defined in the *UML Superstructure Specification*.

5.2 Mapping of CIM Elements

Table 1 gives a non-normative overview of the mapping.

Table 1 – Overview of CIM Element Mapping

CIM Element	UML Construct
Class	*Class* metaclass, as defined in 5.4.
Indication	*Class* metaclass, as defined in 5.5.
Association	*AssociationClass* metaclass, as defined in 5.6.
Property	*Property* metaclass, as defined in 5.7.
Reference	*Property* metaclass, as defined in 5.8.
Method	*Operation* metaclass, as defined in 5.9.
Parameter	*Parameter* metaclass, as defined in 5.10.
Schema	*CIM_Schema* stereotype, as defined in 5.3, and in addition the schema name stays a prefix on class names.
Namespace	The CIM Namespace in a CIM Server is not mapped since it is a runtime entity. The `#pragma namespace` directive in CIM MOF files is mapped to the *CIM_NamespacePath* stereotype, as defined in 5.14.
Instance	*InstanceSpecification* metaclass, as defined in 5.12.
Datatype	*DataType* metaclass, as defined in 5.11.
Qualifier	*Stereotype* metaclass and native UML constructs, as defined in 5.13.
UmlPackagePath Qualifier	*Package* hierarchy, as defined in 5.3.
OCL Constraint Qualifiers	*Constraint* metaclass, as defined in 5.13.4.
Values and ValueMap Qualifiers	*Enumeration* and *EnumerationLiteral* metaclasses, as defined in 5.13.4.16.
CIM MOF files	Several stereotypes, as defined in 5.14.
Trigger	not mapped, as explained in 5.20.

The following subclauses normatively define the mapping for each CIM element.

The tables used in these subclauses list all "non-derived" attributes and associations of the UML metaclasses as defined in the *UML Superstructure Specification*. UML attributes and associations that are marked as "derived" in the *UML Superstructure Specification* can be calculated from other entities and therefore are not listed in these tables.

Future versions of the *UML Superstructure Specification* may define additional UML attributes or associations. If this is done in a downward compatible way, this version of this document can be used to map that future version without change.

In the following subclauses, associations between UML metaclasses are only described in terms of those of their association ends that are owned by the associated metaclasses. The associations themselves, as well as any association ends owned by them are not described. This follows the way these associations are described in the *UML Superstructure Specification*.

NOTE: Associations between the UML metaclasses are not to be confused with UML associations (i.e., the *AssociationClass* metaclass), as they link UML metaclasses (for example a *Property* to a *Class*), whereas UML associations link instances of the UML *Class* metaclass (i.e., "classes").

For UML attributes and associations that are mapped to CIM elements, the mapping is mandatory unless noted otherwise. For some of the attributes and associations that are not mapped to CIM elements, a *compliance value* is defined. If the attribute or association does not have the compliance value, the UML user model is not a valid CIM model. Tools may implement validation against the compliance value in order to expose error or warning conditions when validating the UML user model or when exporting the UML user model to CIM MOF.

The tables used in these subclauses list the multiplicities of UML attributes and associations. In some cases, they are restrictions of the multiplicities defined in the UML metaclasses. The default multiplicity used in these tables is "[1]". A multiplicity of "[0]" for UML associations means that there are no associated elements. A multiplicity of "[0]" for UML attributes means that there are no values.

5.3 Mapping to UML Packages

NOTE: In UML, any element is contained in a package. When mapping CIM elements to UML, it can be assumed without restricting the general applicability of this document that a target package is defined, which acts as the root of a package tree containing the CIM elements.

The UML package containing a CIM class (or indication or association) shall be determined by the *UmlPackagePath* qualifier of that class, as follows: The UML *Class* to which a CIM class is mapped shall be (directly) owned by a UML *Package* whose (relative) package path under the above mentioned target package is:

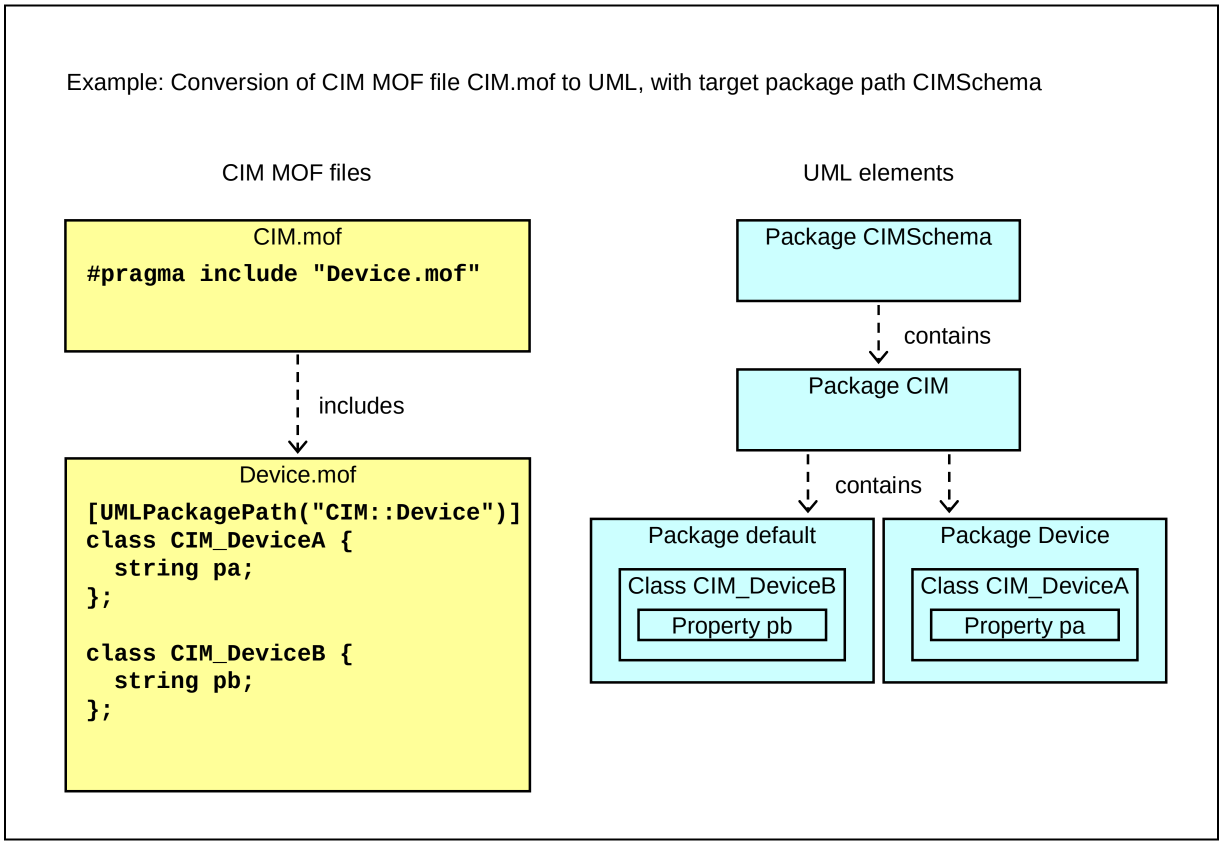
- the value of its *UmlPackagePath* qualifier if that value is not NULL, or
- otherwise, the schema name of the CIM class, followed by "`::default`".

The first package in this package path shall have the stereotype *CIM_Schema* applied. DSP0004 mandates that the first package in a package path of a class is the schema name of the class.

Any package owned directly or indirectly by a package that has the stereotype *CIM_Schema* applied, shall have the stereotype *CIM_Package* applied. The *CIM_Package* and *CIM_Schema* stereotypes are defined in 5.15.

NOTE FOR IMPLEMENTATIONS: If the conversion of CIM MOF to UML involves replacing an existing UML class with the CIM class being converted, then replacement of the UML class should be done in a minimally invasive way, for instance by emptying it of all owned elements (i.e., *ownedAttribute*, *ownedOperation*, *ownedRule*, *ownedComment*), and recreating these elements as defined in the CIM class being converted. This is better than deleting and recreating the entire UML class because it allows retaining any references to the class in the underlying UML storage format, which is advantageous for supporting roundtrip engineering through a UML/CIM-MOF cycle.

Figure 1 shows an example for the package mapping.



```
#pragma include "Device.mof"
```

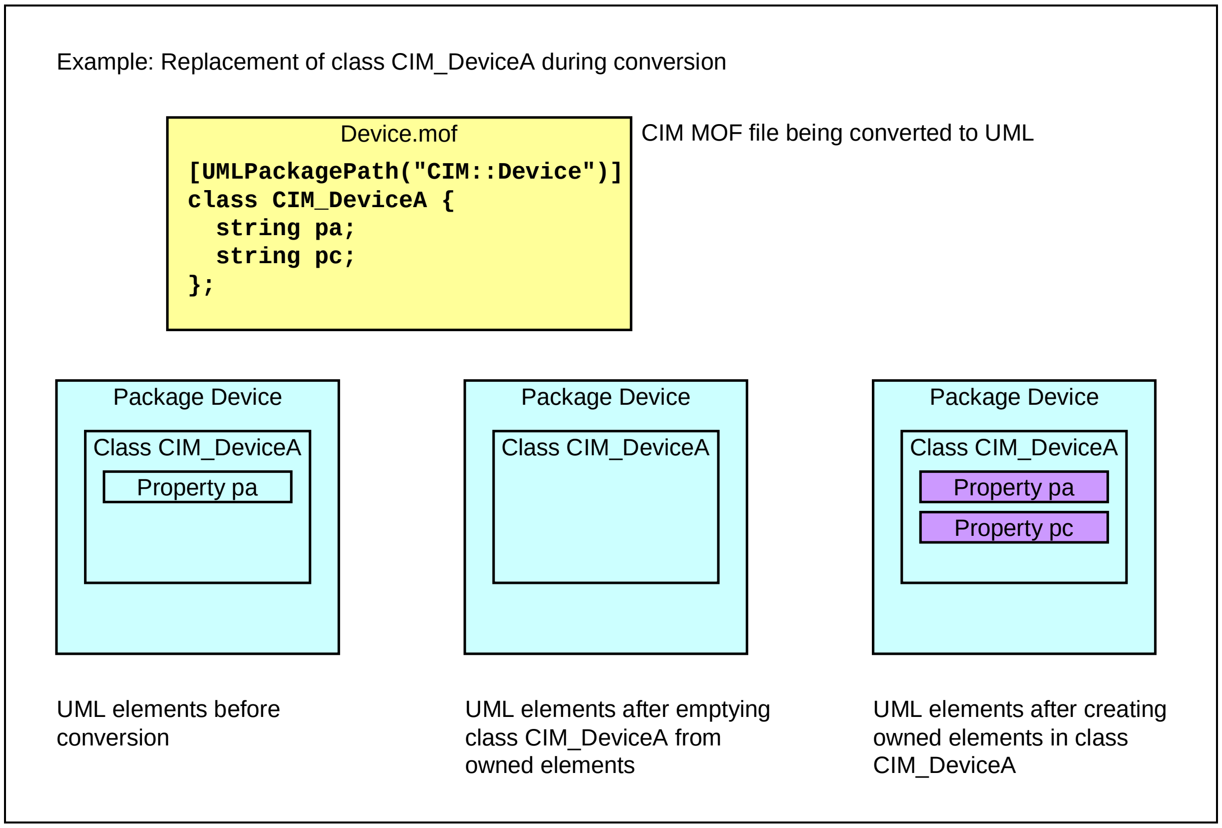
```
[UMLPackagePath("CIM::Device")]
class CIM_DeviceA {
  string pa;
};

class CIM_DeviceB {
  string pb;
};
```

Figure 1 – Example for Package Mapping

In that example, the *UMLPackagePath* qualifier is set for class *CIM_DeviceA*, so its value is used as the relative package path under the target package path. Since in the example, the target package path is *CIMSchema*, *CIM_DeviceA* is contained by package *Device* which is contained by package *CIM* which is contained by package *CIMSchema*. For class *CIM_DeviceB*, the *UMLPackagePath* qualifier is not set (i.e., it is NULL), so the default rule for the *UMLPackagePath* qualifier requires using "CIM::default", and so class *CIM_DeviceB* is contained by package *default* which is contained by package *CIM* which is contained by package *CIMSchema*.

Figure 2 shows a non-normative example for replacing an existing UML class while converting CIM MOF to UML.


Example: Replacement of class CIM_DeviceA during conversion
Device.mof
CIM MOF file being converted to UML
Package Device
Class CIM_DeviceA
Property pa
Package Device
Class CIM_DeviceA
Package Device
Class CIM_DeviceA
Property pa
Property pc
UML elements before conversion
UML elements after emptying class CIM_DeviceA from owned elements
UML elements after creating owned elements in class CIM_DeviceA

```
[UMLPackagePath("CIM::Device")]
class CIM_DeviceA {
  string pa;
  string pc;
};
```

Figure 2 – Example for Replacement of UML Class During Conversion of CIM MOF to UML

In that example, the class *CIM_DeviceA* that is converted to UML already exists. During the conversion, the owned elements of the class (property *pa*) are deleted, and then new owned elements are created according to the definition of the class in the CIM MOF file (properties *pa*, *pc*). The UML element for the class *CIM_DeviceA* is retained, and so all references to that class using identifiers created by the underlying UML storage format continue to be valid.

Per DSP0004, the schema name of CIM classes (e.g., "CIM" in "CIM_LogicalElement") shall be used as the first package segment in the *UMLPackagePath* qualifier for all CIM classes.

5.4 Mapping of CIM Classes

Each CIM class shall be mapped to a UML *Class* metaclass instance. The CIM inheritance hierarchy shall be maintained.

Table 2 defines the mapping for the attributes and associations of the UML *Class* metaclass.

499 **Table 2 – UML *Class* Metaclass**

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
isAbstract	Boolean	Classifier	Shall be mapped to the CIM *Abstract* qualifier as defined in 5.13.4.
isLeaf	Boolean	RedefinableElement	Shall be mapped to the CIM *Terminal* qualifier as defined in 5.13.4.
name	String	NamedElement	Shall be the name of the CIM class, including the CIM schema name.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
nestedClassifier	Classifier [0]	Class	Compliance value: no associated elements.
ownedAttribute	Property [*]	Class	Shall own and shall be associated with the *Property* metaclass instances to which the CIM properties are mapped, as defined in 5.7. The order of elements in *ownedAttribute* shall match the order of properties in the CIM class.
ownedOperation	Operation [*]	Class	Shall own and shall be associated with the *Operation* metaclass instances to which the CIM methods are mapped, as defined in 5.9. The order of elements in *ownedOperation* shall match the order of methods in the CIM class.
generalization	Generalization [0..1]	Classifier	If this CIM class has a superclass, shall own and shall be associated with a *Generalization* metaclass instance, as defined in 5.16.6, whose *general* association shall be associated with the *Class* metaclass instance to which the CIM superclass of this class is mapped. If this CIM class has no superclass, shall not have any associated elements.
redefinedClassifier	Classifier [0]	Classifier	Compliance value: no associated elements.
package	Package	Type (since UML 2.1)	Shall be mapped as defined in 5.3.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
ownedRule	Constraint [*]	Namespace	Shall be mapped to "inv:" constraints of the CIM *ClassConstraint* qualifier of the CIM class, as defined in 5.13.4.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0..1]	Element	Shall be mapped to CIM *Description* qualifier of the CIM class, as defined in 5.13.4.

5.5 Mapping of CIM Indications

Each CIM indication shall be mapped to a UML *Class* metaclass instance. The CIM inheritance hierarchy shall be maintained.

Table 2 defines the mapping for the attributes and associations of the UML *Class* metaclass.

5.6 Mapping of CIM Associations

Each CIM association class shall be mapped to a UML *AssociationClass* metaclass instance with owned association ends. The CIM inheritance hierarchy shall be maintained.

Table 3 defines the mapping for the attributes and associations of the UML *AssociationClass* metaclass.

NOTE: The color and other graphical appearance of the association link are not normatively defined in this document, because the UML Superstructure Specification does not allow specifying the graphical appearance. It is recommended that tools apply the DMTF conventions for the color of association links (i.e., red for associations and green for aggregations and compositions).

Table 3 – UML *AssociationClass* Metaclass

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
all attributes from the CIM class mapping			
isDerived	Boolean	Association	Compliance value: false (UML default)
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
all associations from the CIM class mapping			
memberEnd	Property [2..*]	Association	Shall be associated with the *Property* metaclass instances to which the CIM references in this CIM association class are mapped, as defined in 5.8. The order of elements in *memberEnd* does not need to match the order in *ownedEnd*[1].
ownedEnd	Property [2..*]	Association	Shall own and shall be associated with the *Property* metaclass instances to which the CIM references in this CIM association class are mapped, as defined in 5.8. The order of elements in *ownedEnd* shall match the order of the CIM references in the CIM association class.
navigableOwnedEnd	Property [2..*]	Association	Shall be associated with the *Property* metaclass instances to which the CIM references in this CIM association class are mapped, as defined in 5.8. The order of elements in *navigableOwnedEnd* does not need to match the order in *ownedEnd*[2].

Notes: [1] UML infrastructures are supposed to set the *memberEnd* elements automatically as a result of setting any elements in *ownedEnd*, in order to automatically satisfy the constraint that *ownedEnd* subsets *memberEnd*.

[2] In UML2 when association ends are owned by the association (as defined by this document), navigability of the association is determined by the presence of the UML *Properties* that are associated by *ownedEnd*, in the *navigableOwnedEnd* association. Note that while UML navigability cannot be represented in CIM, navigability is still required to be set by valid CIM models.

5.7 Mapping of CIM Properties

Each CIM property defined in a CIM class shall be mapped to a UML *Property* metaclass instance associated with the UML *Class* metaclass instance to which the CIM class is mapped. Covered properties are treated like any other properties.

Table 4 defines the mapping for the attributes and associations of the UML *Property* metaclass.

Table 4 – UML *Property* Metaclass Used to Map CIM Properties

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
aggregation	AggregationKind	Property	Compliance value: none (UML default).
isDerived	Boolean	Property	Compliance value: false (UML default).
isDerivedUnion	Boolean	Property	Compliance value: false (UML default).
isReadOnly	Boolean	Property	Shall be mapped to CIM *Write* qualifier as defined in 5.13.4.
isStatic	Boolean	Feature	Shall be mapped to CIM *Static* qualifier as defined in 5.13.4.
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	Shall be the name of the CIM property.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
isOrdered	Boolean	MultiplicityElement	Shall be mapped to CIM *ArrayType* qualifier as defined in 5.13.4.
isUnique	Boolean	MultiplicityElement	Compliance value: false (UML default is true).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
association	Association [0]	Property	Compliance value: no associated elements.
owningAssociation	Association [0]	Property	Compliance value: no associated elements.
datatype	DataType [0]	Property	Compliance value: no associated elements (because this property is not owned by a structured datatype).
defaultValue	ValueSpecification [0..1]	Property	If the CIM property has no default value (i.e., an implied default value of NULL) or a specified default value of NULL, this association shall have either no associated elements or it shall own and shall be associated with a *LiteralNull* metaclass instance. Otherwise, shall own and shall be associated with a subclass of the abstract *LiteralSpecification* metaclass that matches the type of the CIM property as defined in Table 8 (i.e., *LiteralBoolean*, *LiteralString*, *LiteralInteger*). If *defaultValue* has no associated elements, this shall be interpreted as a default value of NULL.
redefinedProperty	Property [0..1]	Property	Mapped to the CIM *Override* qualifier as defined in 5.13.4.
subsettedProperty	Property [0]	Property	Compliance value: no associated elements.
associationEnd	Property [0]	Property	Compliance value: no associated elements.

UML Association	UML Type	Defined in UML Metaclass	CIM Mapping
qualifier	Property [0]	Property	Compliance value: no associated elements.
type	Type	TypedElement	Shall be mapped as defined in Table 8.
lowerValue	ValueSpecification [0..1]	MultiplicityElement	If the property has no array type, lowerValue either shall have no associated elements (implying the UML default value of "1"), or it shall own and shall be associated with a LiteralInteger metaclass instance whose value attribute is set to "1". Otherwise, lowerValue shall own and shall be associated with a LiteralInteger metaclass instance whose value attribute is set as follows: To "0", if the CIM property has array type of variable length; To the array size, if the CIM property has array type of fixed length. Note that the CIM qualifier Required does not define the presence of properties; it only defines whether or not a property can be NULL.
upperValue	ValueSpecification [0..1]	MultiplicityElement	If the property has no array type, upperValue either shall have no associated elements (implying the UML default value of "1"), or it shall own and shall be associated with a LiteralUnlimitedNatural metaclass instance whose value attribute is set to "1". Otherwise, upperValue shall own and shall be associated with a LiteralUnlimitedNatural metaclass instance whose value attribute is set as follows: To "*", if the CIM property has array type of variable length; To the array size, if the CIM property has array type of fixed length.
ownedComment	Comment [0..1]	Element	Shall be mapped to CIM Description qualifier of the CIM property, as defined in 5.13.4.
class (since UML 2.1)	Class	Property	Shall be owned by and associated with the Class metaclass instance to which the CIM class defining the property is mapped, as defined in 5.4.
Stereotype Property	**UML Type**	**Defined in Stereotype**	**CIM Mapping**
ClassReferenceType	Enumeration CIM_ClassReferenceType_Enum	CIM_ClassReference Type	Shall be mapped as defined in Table 8.

5.8 Mapping of CIM References

Each CIM reference defined in a CIM association class shall be mapped to a UML *Property* metaclass instance associated with the UML *AssociationClass* metaclass instance to which the CIM association class is mapped.

The mapping for CIM references used as method parameters is defined in 5.10.

Table 5 defines the mapping for the attributes and associations of the UML *Property* metaclass.

Table 5 – UML *Property* Metaclass Used to Map CIM References

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
All attributes from the CIM property mapping unless redefined in the remainder of this table.			
aggregation	AggregationKind	Property	Shall be mapped to CIM *Association*, *Aggregation*, *Composition* and *Aggregate* qualifiers as defined in 5.13.4.
isStatic	Boolean	Feature	Compliance value: false (UML default).
isOrdered	Boolean	MultiplicityElement	Compliance value: false (UML default).
isUnique	Boolean	MultiplicityElement	Compliance value: true unless there are non-reference key properties defined in the same class (UML default is true).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
All associations from the CIM property mapping unless redefined in the remainder of this table.			
association	Association	Property	Shall be associated with the *AssociationClass* metaclass instance to which the CIM association class that defines the CIM reference is mapped.
owningAssociation	Association	Property	Shall be associated with the *AssociationClass* metaclass instance to which the CIM association class that defines the CIM reference is mapped.
type	Type	TypedElement	Shall be mapped as defined in Table 8.
lowerValue	ValueSpecification	MultiplicityElement	Shall be mapped to the CIM *Min* qualifier as defined in 5.13.4.
upperValue	ValueSpecification	MultiplicityElement	Shall be mapped to the CIM *Max* qualifier as defined in 5.13.4.

5.9 Mapping of CIM Methods

Each CIM method shall be mapped to a UML *Operation* metaclass instance associated with the UML *Class* metaclass instance to which the CIM class owning the method is mapped.

While the method return value in CIM is part of the *Method* meta element, it is not in UML. In UML, an instance of the *Parameter* metaclass with its *direction* attribute set to *return* is used to represent a method return value. Therefore, the mapping of the return value of CIM methods is defined in 5.10.

Table 6 defines the mapping for the attributes and associations of the UML *Operation* metaclass.

Table 6 – UML *Operation* Metaclass Used to Map CIM Methods

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
isQuery	Boolean	Operation	Compliance value: false (UML default).
isStatic	Boolean	Feature	Shall be mapped to the CIM *Static* qualifier as defined in 5.13.4.
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	Shall be the name of the CIM method, without signature.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).

UML Association	UML Type	Defined in UML Metaclass	CIM Mapping
class	Class	Operation	Shall be associated with the UML *Class* metaclass instance to which the CIM class is mapped.
bodyCondition	Constraint [0]	Operation	Compliance value: no associated elements.
ownedParameter	Parameter [*]	Operation	Shall own and shall be associated with the *Parameter* metaclass instances to which the CIM method parameters and the return value are mapped, as defined in 5.10. The order of elements in *ownedParameter* shall match the order of parameters in the CIM class.
postcondition	Constraint [*]	Operation	Shall be mapped to "post:" constraints of CIM *MethodConstraint* qualifier of the CIM method, as defined in 5.13.4.
precondition	Constraint [*]	Operation	Shall be mapped to "pre:" constraints of CIM *MethodConstraint* qualifier of the CIM method, as defined in 5.13.4.
raisedException	Type [0]	Operation	Compliance value: no associated elements.
redefinedOperation	Operation [0..1]	Operation	Shall be mapped to the CIM *Override* qualifier as defined in 5.13.4.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
ownedRule	Constraint [*]	Namespace	Shall be mapped to "body:" constraints of CIM *MethodConstraint* qualifier of the CIM method, as defined in 5.13.4.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0..1]	Element	Shall be mapped to the CIM *Description* qualifier of the CIM method, as defined in 5.13.4.

5.10 Mapping of CIM Parameters and CIM Return Values

Each parameter and the return value of CIM methods shall be mapped to a UML *Parameter* metaclass instance associated with the UML *Operation* metaclass instance to which the CIM method is mapped.

Table 7 defines the mapping for the attributes and associations of these UML *Parameter* metaclass, for both the cases of parameters and return values. It applies both to reference types and non-reference types.

Table 7 – UML *Parameter* Metaclass Used to Map CIM Parameters and CIM Return Values

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
direction	ParameterDirectionKind	Parameter	For CIM parameters: shall be mapped to the CIM *In* and *Out* qualifiers of the CIM parameter, as defined in 5.13.4. For the CIM return value: shall be mapped to the *ParameterDirectionKind* enumeration value "return".
isOrdered	Boolean	MultiplicityElement	Shall be mapped to the CIM *ArrayType* qualifier as defined in 5.13.4.
isUnique	Boolean	MultiplicityElement	Compliance value: false (UML default is true).

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String	NamedElement	For CIM parameters: shall be the name of the CIM parameter. For the CIM return value: should be the string “ReturnValue”.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification [0]	Parameter	Compliance value: no associated elements.
type	Type	TypedElement	Shall be mapped as defined in Table 8.
lowerValue	ValueSpecification	MultiplicityElement	If the parameter has no array type, *lowerValue* either shall have no associated elements (implying the UML default value of "1"), or it shall own and shall be associated with a *LiteralInteger* metaclass instance whose *value* attribute is set to "1". Otherwise, shall own and shall be associated with a *LiteralInteger* metaclass instance whose *value* attribute is set as follows: To "0", if the CIM property has array type of variable length; To the array size, if the CIM property has array type of fixed length. Note: The CIM qualifier *Required* does not define the presence of parameters; it only defines whether a parameter can be NULL.
upperValue	ValueSpecification	MultiplicityElement	If the parameter has no array type, *upperValue* either shall have no associated elements (implying the UML default value of "1"), or it shall own and shall be associated with a *LiteralUnlimitedNatural* metaclass instance whose *value* attribute is set to "1". Otherwise, shall own and shall be associated with a *LiteralUnlimitedNatural* metaclass instance whose *value* attribute is set as follows: To "*", if the CIM property has array type of variable length; To the array size, if the CIM property has array type of fixed length
ownedComment	Comment [0..1]	Element	Shall be mapped to the CIM *Description* qualifier of the CIM parameter, as defined in 5.13.4.
Stereotype Property	**UML Type**	**Defined in Stereotype**	**CIM Mapping**
ClassReferenceType	Enumeration CIM_ClassReferenceType_Enum	CIM_ClassReferenceType	Shall be mapped as defined in Table 8.

5.11 Mapping of CIM Datatypes

This subclause normatively defines how CIM datatypes are mapped to UML constructs.

Table 8 defines the mapping of CIM datatypes. Any *DataType* instances defined in this table shall be owned by a UML Type Library, as defined in 5.21.3.

The UML construct specified in the "UML construct for *type* association" column shall be used for the *type* association in the UML metaclass instance using the type.

Since the *CIM Infrastructure Specification* defines that CIM datatypes have no subtype relationships between each other, the *DataType* instances defined in Table 8 shall have no supertype.

The "UML metaclass used for values" column defines the UML metaclass that shall be used for values of this CIM datatype in a UML user model, for example in default values or instance property values.

Table 8 – Mapping of CIM Datatypes

CIM Datatype	UML Construct for *type* Association	UML Metaclass Used for Values
uint8	*DataType* instance named "uint8"	LiteralInteger[1]
sint8	*DataType* instance named "sint8"	LiteralInteger[1]
uint16	*DataType* instance named "uint16"	LiteralInteger[1]
sint16	*DataType* instance named "sint16"	LiteralInteger[1]
uint32	*DataType* instance named "uint32"	LiteralInteger[1]
sint32	*DataType* instance named "sint32"	LiteralInteger[1]
uint64	*DataType* instance named "uint64"	LiteralInteger[1]
sint64	*DataType* instance named "sint64"	LiteralInteger[1]
string	*DataType* instance named "string"	LiteralString[1]
string with *EmbeddedInstance* qualifier	UML *Class* instance to which the CIM class specified as a value of *EmbeddedInstance*, is mapped. In addition, the *ClassReferenceType* stereotype property of the UML metaclass instance to which the qualified CIM element is mapped, shall be set to *ByValue*.	See "UML construct" column
string [ ] with *Octetstring* qualifier	Mapped like an array of *DataType* "octetstring". See the last row in the table for how arrays are mapped.	LiteralString[1]
uint8 [ ] with *Octetstring* qualifier	*DataType* instance named "octetstring"	LiteralString[1]
boolean	*DataType* instance named "boolean"	LiteralBoolean[1]
real32	*DataType* instance named "real32"	LiteralInteger[1]
real64	*DataType* instance named "real64"	LiteralInteger[1]
datetime	*DataType* instance named "datetime"	LiteralString[1]
<classname> ref	UML *Class* instance to which the CIM class that is referenced in the CIM reference is mapped. In addition, for parameters, the *ClassReferenceType* stereotype property of the UML *Parameter* to which the CIM parameter is mapped, shall be set to *ByReference*.	Ends of association instances
char16	*DataType* instance named "char16"	LiteralString[1]

CIM Datatype	UML Construct for *type* Association	UML Metaclass Used for Values
arrays	The CIM datatype of the array elements is mapped as defined in this table, and the array characteristic is mapped to the *lowerValue* and *upperValue* associations of the UML *Property* and *Parameter* metaclasses, as defined in 5.7 and 5.10.	See "UML construct" column

Note: [1]The specified UML metaclass is preferred for simplicity and interoperability. All other valid representations of such values as defined in the *UML Superstructure Specification* may be used in addition.

5.12 Mapping of CIM Instances

Each modeled CIM instance shall be mapped to a UML *InstanceSpecification* metaclass instance. A modeled CIM instance is one that is owned by a UML user model. This includes CIM instances defined in CIM MOF and imported into UML.

NOTE: Modeled CIM instances do not necessarily represent the run-time CIM instances in a CIM server. In case where the modeled instance resulted from importing a CIM MOF definition of a CIM instance into UML, and where the same CIM MOF definition is imported into a CIM server, the modeled instance represents at least the initial state of a run-time instance. However, modeled instances can also be used to describe use cases in UML, and in such cases typically only a subset of the properties is described, and the run-time instances cannot be deducted from that. Also, there may be multiple modeled instances that represent different points in time for the same run-time instance.

Table 9 defines the mapping for the attributes and associations of the UML *InstanceSpecification* metaclass.

Table 9 – UML *InstanceSpecification* Metaclass Used for Modeled CIM Instances

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String	NamedElement	No compliance value is defined; extension point.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
classifier	Classifier [1]	InstanceSpecification	Shall be associated with the UML *Class* metaclass instance to which the CIM creation class of the modeled CIM instance is mapped.
slot	Slot [*]	InstanceSpecification	Shall own and shall be associated with an instance of the UML *Slot* metaclass for each CIM property that has a value defined in the modeled CIM instance, as defined in Table 10.
specification	ValueSpecification [0]	InstanceSpecification	Compliance value: no associated elements.
ownedComment	Comment [0..1]	Element	No compliance value is defined; extension point.

Table 10 defines the mapping for the attributes and associations of the UML *Slot* metaclass referenced from Table 9.

Table 10 – UML *Slot* Metaclass Used for Property Values of Modeled CIM Instances

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
No attributes.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
definingFeature	StructuralFeature [1]	Slot	Shall be associated with the UML *Property* metaclass instance to which the CIM property is mapped.
owningInstance	InstanceSpecification [1]	Slot	Shall be associated with the UML *InstanceSpecification* metaclass instance to which the CIM instance that has the CIM property value is mapped.
value	ValueSpecification [*]	Slot	Shall own and shall be associated with instances of a subclass of the abstract *LiteralSpecification* metaclass that matches the type of the CIM property as defined in Table 8 (i.e., *LiteralBoolean*, *LiteralString*, *LiteralInteger*) or *LiteralNull* to represent the CIM NULL value. There shall be one instance for each value (i.e., for non-arrays, there is one instance, and for arrays, there is one instance for each array element that has a value).
ownedComment	Comment [0..1]	Element	No compliance value is defined; extension point.

5.13 Mapping of CIM Qualifiers

5.13.1 Overview

This subclause gives an overview of the mapping of CIM qualifier types and CIM qualifier values to UML constructs. Note that CIM qualifier types are sometimes called "qualifier declarations", but we use the term "qualifier types" in this document.

In CIM, there is a distinction between qualifier types and qualifier values: The definition of a qualifier type is needed in order to define a qualifier value of that type on a CIM element. Since the definition of qualifier types is dynamic in CIM, this document defines a UML mapping for the qualifier types in order to represent the definition of a qualifier type, as well as a UML mapping for the qualifier values.

The set of DMTF defined qualifier types is defined in each release of the CIM Schema. The *CIM Infrastructure Specification* specifies all DMTF qualifier types at a particular point in time, but typically the CIM Schema is on shorter release cycle than the *CIM Infrastructure Specification*, and so the CIM Schema ends up to be the normative definition of the set of DMTF defined qualifier types.

The set of CIM qualifiers is vendor extensible, allowing vendors to define additional qualifier types. In a particular vendor extension schema, the DMTF defined qualifier types and the vendor defined qualifier types make up the complete set of qualifier types that is to be mapped to UML.

These qualifier types (i.e., the qualifier type definitions) are mapped to UML as defined in 5.13.2, for all qualifier types.

The qualifier values are mapped to UML using two mapping approaches, depending on the qualifier type:

- **Direct Qualifier Mapping**

  Subclause 5.13.4 defines the mapping rules for values of a specific set of qualifier types. That subclause covers the subset of CIM defined standard and optional qualifier types that are mapped to specific UML features. A tool that imports CIM MOF into a UML user model needs to recognize these qualifiers by their name in order to implement the specific mapping rules.

- **Generic Qualifier Mapping**

  Subclause 5.13.3 defines the mapping rules for values of all qualifier types that are not in the specific set covered by the direct qualifier mapping. A tool that imports CIM MOF into a UML user model can implement generic support for this mapping approach without requiring knowledge about the specific qualifier types; hence the mapping approach allows to deal with an unknown set of qualifiers without having to update this document nor the mapping support in tools.

The inheritance behavior of CIM qualifiers and the inheritance behavior of the UML constructs to which they are mapped are somewhat different, as explained in the following paragraphs.

The inheritance behavior of CIM qualifiers is defined in the *CIM Infrastructure Specification* and is repeated here briefly:

- CIM qualifier values do not need to be defined explicitly on each CIM element.
- However, all qualifiers have an effective value on each CIM element, as follows:
  - If a value for a qualifier is defined explicitly (for example, "Write=true" or "Write") on the CIM element, the effective qualifier value is the value defined there.
  - If no value for a qualifier is defined explicitly (for example, "", which means that "Write" is not specified) on the CIM element, the effective qualifier value is determined as follows:
    - Qualifiers with flavor *ToSubclass* inherit their effective value along the inheritance chain. Classes that have no superclass (i.e., top classes) and CIM elements within such classes obtain their effective value from the default value defined in the declaration of the qualifier type.
    - Qualifiers with flavor *Restricted* obtain their effective value from the default value defined in the declaration of the qualifier type.

The inheritance behavior of the UML constructs to which the CIM qualifier values are mapped, is as follows:

- Some UML constructs are required to be present at each level of a class inheritance chain (for example, the *isAbstract* attribute). For these constructs, there is no automatic propagation of values along the inheritance chain, i.e., each value stands on its own and a mapping rule needs to cover each level of the inheritance chain even if the qualifier value is not defined at each of those levels.
- Some UML constructs are optionally present at each level of a class inheritance chain (for example, stereotypes or associated UML metaclass instances). These UML constructs are not inherited by subclasses in a UML user model.

The following general principles are followed in order to map these different inheritance behaviors:

1) For CIM qualifiers mapped to UML constructs that are required to be present at each level of an inheritance chain, the qualifier value represented in the UML construct is the effective qualifier value for that level in the inheritance chain. This leads to redundant qualifier values in the UML user model even if they were not redundant in the CIM model, but that is unavoidable because the UML constructs are required to be present.

2) For CIM qualifiers mapped to UML constructs that are optionally present at each level of an inheritance chain, the UML construct is present if and only if the effective qualifier value for that level in the inheritance chain is different from the default value (for *Restricted* qualifiers) or the inherited value (for *ToSubclass* qualifiers). In other words, the presence of the UML constructs is minimized as much as possible, removing any redundancy, even if the value was redundantly defined in the CIM model. If the UML construct is present, the qualifier value represented in the UML construct is the effective qualifier value for that level in the inheritance chain.

During the course of editing a UML user model with a UML editor, optional UML constructs representing qualifiers may be present even if their value is the same as the inherited value. A tool can provide a means to remove such redundancies.

Because the second principle avoids redundant qualifier value definitions, a CIM MOF that contained such redundancies may be different from the CIM MOF that is created when the original CIM MOF is imported into a UML user model and exported again. Semantically, both versions are the same however and no information is lost. There are a few qualifiers in CIM for which redundant values are used commonly even though this is not mandated by the CIM Infrastructure Specification. This document accommodates these special cases.

These general principles are reflected by the normative mapping rules in the subsequent subclauses.

5.13.2 Mapping of Qualifier Types

NOTE: The general principle for the mapping of CIM qualifier types (i.e., the qualifier declarations) to UML is that the UML construct to which they are mapped is able to represent the qualifier type itself as a self-contained entity, independently of any usages of the qualifier type in qualifier values.

Each CIM qualifier type shall be mapped to an instance of the UML *Stereotype* metaclass. These stereotypes are called "Qualifier Type Stereotypes" in this document. The Qualifier Type Stereotypes are owned by a UML profile as defined in 5.21.

Table 11 defines the mapping for the attributes and associations of the UML *Stereotype* metaclass for this purpose.

NOTE: The UML *Stereotype* metaclass has mostly attributes and associations from the UML Infrastructure Library as defined in the UML Infrastructure Specification.

Table 11 – UML *Stereotype* Metaclass Used as Qualifier Type Stereotype

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: true. Note that the qualifier type stereotypes have no extensions defined and are not applied directly. Instead, they are inherited into the qualifier scope stereotypes defined in Table 15.
isLeaf	Boolean	Not defined in the CIM Infrastructure Specification at this point	If supported by the UML tool then compliance value: false.

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
name	String	NamedElement	Shall be the name of the CIM qualifier type, using the following syntax: `"CIM_Qualifier_" qualifier-name` Where `qualifier-name` is the qualifier name as defined in the CIM MOF declaration of the qualifier type. The lexical case of the name shall be preserved. Note that this rule does not contradict the rule that names in CIM are compared case-insensitively. Note that the CIM defined qualifiers as described in the *CIM Infrastructure Specification* use all-upper-case characters, while the CIM MOF declaration in the CIM Schema uses mixed case.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private)
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property [1]	Class	Shall own and shall be associated with a single UML *Property* metaclass instance representing the definitional aspects of the qualifier value, as defined in Table 12.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [0]	Class	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [*]	UML *Stereotype* metaclass	No compliance value is defined; extension point.

Each Qualifier Type Stereotype shall own a single property representing the definitional aspects of the qualifier value, as defined in Table 12.

NOTE: Even though the *Stereotype.ownedAttribute* association is owned by a metaclass contained in a package from the *InfrastructureLibrary* package, the *Property* metaclass it is associated with is from the *UML::Classes* package.

The set of all qualifier types defined in the "CIM" profile shall be determined by enumerating any stereotypes defined in the "CIM" profile that have a name starting with "CIM_Qualifier_".

A stereotype *CIM_QualifierType* (defined further down) shall be applied to this property, adding some attributes needed for qualifiers. Table 12 also defines how these additional attributes are mapped.

Table 12 – UML *Property* Metaclass Used for Definitional Aspects of CIM Qualifier Value Within Qualifier Type Stereotype

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
aggregation	AggregationKind	Property	Compliance value: none (UML default).
isDerived	Boolean	Property	Compliance value: false (UML default).
isDerivedUnion	Boolean	Property	Compliance value: false (UML default).
isReadOnly	Boolean	Property	Compliance value: false (UML default).

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
isStatic	Boolean	Feature	Compliance value: false (UML default).
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	Shall be the name of the CIM qualifier as defined in CIM MOF, preserving the lexical case.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
isOrdered	Boolean	MultiplicityElement	For CIM qualifiers that are arrays: compliance value: true. The order of the array entries shall be preserved. For CIM qualifiers that are not arrays: no compliance value defined. Note that *isOrdered* is meaningless for non-arrays.
isUnique	Boolean	MultiplicityElement	Compliance value: false (UML default is true).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
association	Association [0]	Property	Compliance value: no associated elements.
owningAssociation	Association [0]	Property	Compliance value: no associated elements.
datatype	DataType [0]	Property	Compliance value: no associated elements (because this property is not owned by a structured datatype).
defaultValue	ValueSpecification [0..1]	Property	If the CIM qualifier type has no default value (i.e., an implied default value of NULL) or a specified default value of NULL, this association shall have either no associated elements or it shall own and shall be associated with a *LiteralNull* metaclass instance. Otherwise, shall own and shall be associated with a subclass of the abstract *LiteralSpecification* metaclass that matches the type of the CIM qualifier as defined in Table 8 (i.e., *LiteralBoolean*, *LiteralString*, *LiteralInteger*) or *LiteralNull* to represent the CIM NULL value. If defaultValue has no associated elements, this shall be interpreted as a default value of NULL.
redefinedProperty	Property [0]	Property	Compliance value: no associated elements.
subsettedProperty	Property [0]	Property	Compliance value: no associated elements.
associationEnd	Property [0]	Property	Compliance value: no associated elements.
qualifier	Property [0]	Property	Compliance value: no associated elements.
type	Type	TypedElement	Shall be mapped as defined in Table 8.
lowerValue	ValueSpecification [0..1]	MultiplicityElement	Shall own and shall be associated with a *LiteralInteger* metaclass instance whose *value* attribute is set to "0". For a discussion of the reasons for this mapping, refer to the notes in 5.13.3.

679

UML Association	UML Type	Defined in UML Metaclass	CIM Mapping
upperValue	ValueSpecification [0..1]	MultiplicityElement	If the property has no array type, *upperValue* either shall have no associated elements (implying the UML default value of "1"), or it shall own and shall be associated with a *LiteralUnlimitedNatural* metaclass instance whose *value* attribute is set to "1". Otherwise, *upperValue* shall own and shall be associated with a *LiteralUnlimitedNatural* metaclass instance whose *value* attribute is set as follows: To "*", if the CIM qualifier has array type of variable length; To the array size, if the CIM qualifier has array type of fixed length.
ownedComment	Comment [*]	Element	Currently, there is no description for qualifiers in CIM. Therefore: no compliance value defined, extension point.
class (since UML 2.1)	Class	Property	Shall be owned by and associated with the *Stereotype* metaclass instance owning the property.
Stereotype Property	**UML Type**	**Defined in Stereotype**	**CIM Mapping**
InheritanceRule	Enumeration CIM_QualifierInheritanceRule	CIM_QualifierType	Shall be mapped to the inheritance flavors of the CIM qualifier, as defined in Table 13.
Translatable	Boolean	CIM_QualifierType	Shall be the *Translatable* flavor of the CIM qualifier.

680 The stereotype *CIM_QualifierType* shall be defined in the UML profile "CIMQualifierType" and shall have
681 the Properties defined in Table 13.

682 **Table 13 – Properties of Stereotype *CIM_QualifierType***

Name	Type	Default Value	Meaning
InheritanceRule	Enumeration CIM_QualifierInheritanceRule	ToSubclass EnableOverride	Inheritance flavor of the qualifier type.
Translatable	Boolean	False	*Translatable* flavor of the qualifier type.

683 The enumeration *CIM_QualifierInheritanceRule* shall be defined in the UML profile "CIMQualifierType"
684 and shall have the literals defined in Table 14.

685 **Table 14 – Literals of Enumeration *CIM_QualifierInheritanceRule***

Literal Name	Meaning
ToSubclassEnableOverride	Represents qualifier inheritance flavor *ToSubclass EnableOverride*.
ToSubclassDisableOverride	Represents qualifier inheritance flavor *ToSubclass DisableOverride*.
Restricted	Represents qualifier inheritance flavor *Restricted*.

686 The following stereotypes shall be defined and owned by the UML profile "CIM". These stereotypes
687 are called "Qualifier Scope Stereotypes" in this document because there is one stereotype for each
688 qualifier scope. They form an inheritance tree. Table 15 defines these stereotypes.

Table 15 – Qualifier Scope Stereotypes

Stereotype Name	Abstract	Base Stereotypes	Meaning
CIM_Any_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Any*.	Represents the qualifiers with scope Any. In CIM, scope *Any* is a shorthand for all defined scopes.
CIM_Class_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Class* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Class* (including *Any*).
CIM_Association_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Association* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Association* (including *Any*).
CIM_Indication_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Indication* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Indication* (including *Any*).
CIM_Property_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Property* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Property* (including *Any*).
CIM_Reference_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Reference* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Reference* (including *Any*).
CIM_Method_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Method* but not *Any*, and CIM_Any_Qualifiers.	Represents the qualifiers with scope *Method* (including *Any*).
CIM_Parameter_Qualifiers	Yes	All qualifier type stereotypes of qualifiers with scope *Parameter* but not *Any*, and *CIM_Any_Qualifiers*.	Represents the qualifiers with scope *Parameter* (including *Any*).

5.13.3 Generic Mapping of Qualifier Values

This subclause defines a mapping of the qualifier values that shall be used for any CIM qualifier types that are not covered in 5.13.4. In this document, these qualifier types are called the *generically mapped* qualifier types.

NOTE: Through the mapping of the CIM qualifier types as defined in 5.13.2, each CIM element in a UML user model has the capability to have qualifier values according to their scope. These qualifier values are represented by the values of the qualifier properties inherited into the Meta Element Stereotype applied to the UML elements representing CIM elements. Meta Element Stereotypes are defined in 5.15.

The *UML Superstructure Specification* does not currently define whether or not the values of stereotype properties in applied stereotypes may be omitted. However, optionality of the property values is needed in order to represent the optional presence of qualifier value definitions on CIM elements.

This document assumes that the values of stereotype properties may be omitted if the lower bound of their multiplicity is 0.

The *UML Superstructure Specification* currently leaves some room for interpretation with respect to how an applied stereotype is represented in UML. However, this document needs to define the mapping of qualifier values to property values within applied stereotypes in terms of such a representation.

This document chooses the interpretation that an applied stereotype is represented as a separate UML metaclass named after the stereotype name, which has an accordingly typed attribute for each property owned by the defining stereotype.

Figure 6 shows an example of applied stereotypes with their attributes that represent the qualifier values.

Table 16 defines how the qualifier values for generically mapped qualifiers are represented using applied Meta Element Stereotypes.

Table 16 – Applied Meta Element Stereotypes Used for CIM Qualifier Values

UML attribute	UML type	Defined in UML metaclass	CIM Mapping
<qualifier-name>	<qualifier-type> [0..x]	(each Meta Element Stereotype)	For each CIM qualifier defined on the CIM element represented by the UML element the stereotype is applied to, this attribute shall have the value of the CIM qualifier. If a qualifier is not defined on the CIM element, the attribute shall not have a value.

Where:	<qualifier-name>	is the value of the *name* attribute of the stereotype property, as defined in Table 12.
	<qualifier-type>	is a datatype according to the *type* association of the stereotype property, as defined in Table 12.
	[0..x]	is the multiplicity of the stereotype property, as defined in Table 12.

Qualifier types declared with the CIM flavor DisableOverride shall not be overridden, as defined in 4.5.4 (Class and Property Qualifiers) of the *CIM Infrastructure Specification*.

As described in 5.13.1, each qualifier type that applies to a CIM element has an effective value of the qualifier on that CIM element, regardless of whether or not the qualifier is defined (i.e., set) on the CIM element.

The *BitMap* and *BitValues* qualifiers are intentionally mapped generically, because their usage in the CIM Schema 2.x is minimal and did not seem to warrant the effort to define a direct mapping.

As defined in the *CIM Infrastructure Specification*, qualifiers with the *Translatable* flavor have related implicitly defined qualifier types with locale specific suffixes in the qualifier name. Such locale specific qualifiers are mapped generically, regardless of whether the related qualifier without locale suffix is mapped generically or directly. For example, the *Description* qualifier is mapped directly, and its related locale specific qualifier *Description_de_DE* is mapped generically.

Values of qualifiers with the Translatable flavor shall be interpreted using the effective value of the *ClassLocale* stereotype property of the UML metaclass instance to which the qualified CIM class or the CIM class owning the qualified element is mapped. Subclause 5.14 describes how to determine the effective value of this stereotype property.

5.13.4 Direct Mapping of Qualifier Values

This subclause defines the mapping of the qualifier values of a specific set of CIM qualifier types that have semantically equivalent UML constructs, or for which specific UML stereotypes are defined. These qualifier types are called the *directly mapped* qualifier types in this document.

5.13.4.1 Abstract Qualifier

The *isAbstract* attribute of any UML *Class* metaclass instance shall be equal to the effective value of the *Abstract* qualifier of the corresponding CIM class.

The *CIM::Abstract* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.2 Association, Aggregation, Composition, Aggregate Qualifiers

As defined in 5.6, each CIM association class is mapped to a UML *AssociationClass* metaclass instance, and each CIM reference within a CIM association class is mapped to a UML *Property* metaclass instance. The UML *Property* metaclass has an *aggregation* attribute which specifies the type of aggregation that applies to the association end represented by that UML Property.

The value of the *aggregation* attribute of any UML *Property* metaclass instance representing a CIM reference shall be mapped to the effective values of these qualifiers on the corresponding CIM reference, as defined in Table 17.

Table 17 – Mapping of *Association/Aggregation/Composition/Aggregate* Qualifiers

Kind of CIM Association Class	Value of UML *aggregation* Attribute	
	On Aggregate Reference	On Other Reference
Association	None	None
Aggregation	None	Shared
Composition	None	Composite

The "Aggregate reference" is the CIM reference in a CIM association class that has an effective value of *true* for its *Aggregate* qualifier.

NOTE 1: In CIM, the *Aggregate* qualifier is set on the association end that is the "whole". In UML, the diamond adornment for aggregation is also set on the association end that is the "whole", while the "shared" value of the UML *aggregation* attribute is set on the association end that is the "part".

NOTE 2: In UML, aggregations and compositions are restricted to an arity of two. CIM has no such restriction. This causes a limitation in the mapping because CIM aggregations and compositions that have an arity larger than two cannot be mapped to UML. Tools importing CIM-MOF into UML shall reject such CIM elements.

NOTE 3: The valid value combinations of the boolean CIM qualifiers *Association*, *Aggregation* and *Composition* are not precisely defined in the *CIM Infrastructure Specification*.

For the purpose of this document, the combinations that can be mapped by this document are defined in Table 18.

NOTE 4: It is expected that the *CIM Infrastructure Specification* will define any other combination to be invalid.

Table 18 – Valid Combinations for *Association/Aggregation/Composition* Qualifiers

Value of *Association* Qualifier	Value of *Aggregation* Qualifier	Value of *Composition* Qualifier	Resulting Kind of CIM Class
True	False	False	Association
True	True	False	Aggregation
True	True	True	Composition
False	False	False	Normal Class

The *CIM::Association*, *CIM::Aggregation*, *CIM::Composition* and *CIM::Aggregate* stereotype properties of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.3 ArrayType Qualifier

The *isOrdered* attribute of any UML *Property* or *Parameter* metaclass instances shall be mapped to the effective value of the *ArrayType* qualifier on the corresponding CIM element, as defined in Table 19.

This mapping is motivated by the semantic equivalence of CIM Indexed with UML isOrdered. In both cases, the semantic is that the array shall honor the order of array elements that is provided when setting the array values (i.e., it does not re-order the array elements in a random or algorithmic way).

Table 19 – Mapping of *ArrayType* Qualifier

Value of *ArrayType* Qualifier	Value of *isOrdered* Attribute
"Bag"	False
"Ordered"	False
"Indexed"	True

In addition, in order to represent the distinction between *ArrayType* values "Bag" and "Ordered", the *ArrayType* qualifier value shall be mapped generically as defined in 5.13.3, with the following additional rules:

- The UML datatype of the *ArrayType* property of the *CIM_Qualifier_ArrayType* stereotype shall be the enumeration *CIM_Qualifier_ArrayType_Enum* as defined in Table 20. This enumeration shall be owned by the UML profile "CIM".

Table 20 – Literals of Enumeration *CIM_Qualifier_ArrayType_Enum*

Literal Name	Meaning
Bag	Represents *ArrayType* qualifier value "Bag"
Ordered	Represents *ArrayType* qualifier value "Ordered"
Indexed	Represents *ArrayType* qualifier value "Indexed"

- The value of that *ArrayType* stereotype property shall be set to the effective value of the *ArrayType* qualifier for the particular level in the inheritance chain, as defined in Table 20.

5.13.4.4 Deprecated Qualifier

The value of any *Deprecated* qualifier defined on a CIM element shall be mapped to a Marker Stereotype as defined in 5.13.5, and in addition generically as defined in 5.13.3, with the following additional rules:

- The *name* attribute of the Marker Stereotype shall be set to "Deprecated".
- The image of the Marker Stereotype icon should be set to a red square with the letter "D" in white font, for example as shown in the following image:

- If the effective value of the CIM *Deprecated* qualifier is non-NULL on a CIM element, the Marker Stereotype *Deprecated* shall be applied to the UML element to which that CIM element is mapped, and the string array typed value of the *Deprecated* property of the qualifier type stereotype *CIM_Qualifier_Deprecated* shall be set to the string array typed qualifier value, preserving the order of elements.
- If the effective value of the *Deprecated* qualifier is NULL for a CIM element (i.e., it is not defined on the element), the Marker Stereotype *Deprecated* shall not be applied to the UML element to which that CIM element is mapped, and the string array typed value of the *Deprecated* property of the qualifier type stereotype *CIM_Qualifier_Deprecated* shall be empty.

5.13.4.5 Description Qualifier

The value of any *Description* qualifier defined on a CIM element shall be mapped to a *Comment* metaclass instance owned by and associated, via *ownedComment*, with the UML metaclass instance to which the CIM element is mapped.

Tool specific mappings:

- IBM Rational Software Architect:

  The *Comment* metaclass instance shall have the *Documentation* stereotype applied.

Table 21 defines the mapping for the attributes and associations of the UML *Comment* metaclass.

Table 21 – UML *Comment* Metaclass Used in Multiple UML Metaclasses

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
body	String	Comment	Shall be the transformed value of the CIM *Description* qualifier. The qualifier value is transformed by resolving any backslash escape sequences (for example, "\n") into their corresponding characters. See the description of the *string* datatype in the *CIM Infrastructure Specification* for a definition of supported escape sequences.
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
annotatedElement	Element [1]	Comment	Shall be associated with the UML metaclass instance to which the CIM element that carries the *Description* qualifier is mapped.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

The *CIM::Description* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.6 EmbeddedInstance Qualifier

The value of any *EmbeddedInstance* qualifier defined on a CIM element shall be mapped to the *type* association of the UML metaclass to which the CIM element is mapped. The *type* association shall associate to the UML *Class* metaclass instance to which the CIM class specified in the qualifier value is mapped.

Table 22 specifies the *CIM_ClassReferenceType* stereotype used for UML elements mapped to CIM properties, methods and parameters in order to specify whether instances are referenced or contained by value.

Table 22 – *CIM_ClassReferenceType* Stereotype Used for *Property* and *Parameter* Metaclasses

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false (UML default).
isLeaf	Boolean	Not defined in the *CIM Infrastructure Specification* at this point	If supported by the UML tool then compliance value: false.
name	String	NamedElement	Shall be "CIM_ClassReferenceType".
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).

UML Association	UML Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
ownedAttribute	Property [*]	Class	Shall own and shall be associated with the stereotype properties defined in Table 23.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [0]	Class	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [0]	UML *Stereotype* metaclass	Compliance value: no associated elements.

Table 23 – Properties of Stereotype *CIM_ClassReferenceType*

Name	Type	Default Value	Meaning
ClassReferenceType	Enumeration CIM_ClassReferenceType_ Enum	NotApplicable	Indicates whether the instance is referenced or contained by value

Table 24 – Literals of Enumeration *CIM_ClassReferenceType_Enum*

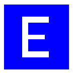
Order Number	Literal Name	Literal Value	Meaning
1	NotApplicable	Integer 0	The element does not reference or contain an instance.
2	ByReference	Integer 1	The instance is referenced through a CIM reference.
3	ByValue	Integer 2	The instance is contained by value in an embedded instance.

The *CIM::EmbeddedInstance* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.7 Experimental Qualifier

The value of any *Experimental* qualifier defined on a CIM element shall be mapped to a Marker Stereotype as defined in 5.13.5, with the following additional rules:

- The *name* attribute of the Marker Stereotype shall be set to "Experimental".
- The image of the Marker Stereotype icon should be set to a blue square with the letter "E" in white font, for example as shown in the following image:



- If the effective value of the *Experimental* qualifier is *True* on a CIM element, the Marker Stereotype *Experimental* shall be applied to the UML element to which that CIM element is mapped.

- If the effective value of the *Experimental* qualifier is *False* on a CIM element, the Marker Stereotype *Experimental* shall not be applied to the UML element to which that CIM element is mapped.

The *CIM::Experimental* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.8 In, Out Qualifiers

The *direction* attribute of any UML *Parameter* metaclass instance shall be mapped to the effective values of the *In* and *Out* qualifiers on the corresponding CIM parameter, as defined in Table 25. All other value combinations of these qualifiers are invalid.

Table 25 – Mapping of *In/Out* Qualifiers

Value of *In* Qualifier	Value of *Out* Qualifier	Value of *direction* Attribute
true	false	in
false	true	out
true	true	inout

The *CIM::In* and *CIM::Out* stereotype properties of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.9 Key Qualifier

The value of any *Key* qualifier defined on a CIM element shall be mapped to a Marker Stereotype as defined in 5.13.5, with the following additional rules:

- The *name* attribute of the Marker Stereotype shall be set to "Key".
- The image of the Marker Stereotype icon should be set to a green square with the letter "K" in white font, for example as shown in the following image:

  K

- If the effective value of the *Key* qualifier is *True* on a CIM element, the Marker Stereotype *Key* shall be applied to the UML element to which that CIM element is mapped.
- If the effective value of the *Key* qualifier is *False* on a CIM element, the Marker Stereotype *Key* shall not be applied to the UML element to which that CIM element is mapped.

The *CIM::Key* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.10 Min, Max Qualifiers

The *lowerValue* association end of any UML *Property* metaclass instance to which a CIM reference is mapped, shall be mapped to the effective value of the *Min* qualifier on the corresponding CIM reference. The *Property* metaclass shall own a *LiteralInteger* metaclass instance as defined in 5.16.3 whose *value* attribute is set to the *Min* qualifier value.

NOTE: A *Min* qualifier value of NULL cannot be mapped by this document. DSP0004 clarifies that the *Min* qualifier shall not be NULL.

The *upperValue* association end of any UML *Property* metaclass instance to which a CIM reference is mapped, shall be mapped to the effective value of the *Max* qualifier on the corresponding CIM reference. The *Property* metaclass shall own a *LiteralUnlimitedNatural* metaclass instance as defined in 5.16.4 whose *value* attribute is set to the *Max* qualifier value, if non-NULL, or to "*" if NULL.

NOTE 1: DSP0004 clarifies that a NULL value for the *Max* qualifier means an upper value of unlimited for the cardinality of the so qualified CIM reference.

NOTE 2: UML uses the multiplicity elements *upperValue* and *lowerValue* in the *Property* metaclass slightly different, depending on whether or not the *Property* is used as an association end of an association that owns its association ends. If the association ends are owned by the association, the multiplicity elements still indicate the multiplicity of the associated classes, but that is different from the multiplicity of the owned association ends themselves (which is always "1"). The mapping of the *Min* and *Max* qualifiers follows the UML definition of multiplicity for association ends owned by the association.

The *CIM::Min* and *CIM::Max* stereotype properties of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.11 OCL Constraint Related Qualifiers

The *CIM Infrastructure Specification* defines a qualifier named *OCL*. DSP0004 deprecates the *OCL* qualifier and adds three new OCL constraint related qualifiers (*ClassConstraint*, *MethodConstraint* and *PropertyConstraint*). This document maps these three new qualifiers and ignores the deprecated *OCL* qualifier.

Each array entry in any CIM *ClassConstraint*, *MethodConstraint* and *PropertyConstraint* qualifier value defined on a CIM element shall be mapped to a UML *Constraint* metaclass instance as defined in Table 27. That *Constraint* metaclass instance shall be associated with the UML metaclass instance to which the CIM element defining the qualifier is mapped, via the UML associations defined in Table 26.

Table 26 – Mapping of OCL Constraint Related Qualifiers

CIM Qualifier	CIM Elements the Qualifier is Used On	OCL Constraint Type	Constraint:: constrainedElement	Constraint:: context
ClassConstraint	Class Association Class Indication	def:	UML *Class* instance to which the qualified CIM class is mapped.	UML *Class* instance to which the qualified CIM class is mapped.
ClassConstraint	Class Association Class Indication	inv:	UML *Class* instance to which the qualified CIM class is mapped.	UML *Class* instance to which the qualified CIM class is mapped.
MethodConstraint	Method	body:	UML *Operation* instance to which the qualified CIM method is mapped.	UML *Class* instance owning the UML *Operation* instance to which the qualified CIM method is mapped.
MethodConstraint	Method	pre:	UML *Operation* instance to which the qualified CIM method is mapped.	UML *Class* instance owning the UML *Operation* instance to which the qualified CIM method is mapped.
MethodConstraint	Method	post:	UML *Operation* instance to which the qualified CIM method is mapped.	UML *Class* instance owning the UML *Operation* instance to which the qualified CIM method is mapped.

CIM Qualifier	CIM Elements the Qualifier is Used On	OCL Constraint Type	Constraint:: constrainedElement	Constraint:: context
PropertyConstraint	Property Reference	init:	UML *Class* instance owning the UML *Property* instance to which the qualified CIM property is mapped.	UML *Class* instance owning the UML *Property* instance to which the qualified CIM property is mapped.
PropertyConstraint	Property Reference	derive:	UML *Class* instance owning the UML *Property* instance to which the qualified CIM property is mapped.	UML *Class* instance owning the UML *Property* instance to which the qualified CIM property is mapped.

Table 27 – Usage of UML *Constraint* Metaclass for OCL Related Constraint Qualifiers

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
visibility	VisibilityKind	PackagableElement	No compliance value is defined. NOTE: It is not clear what visibility should mean on a constraint, OMG issue #10382 has been raised to clarify that, targeting UML 2.2.
name	String [0]	NamedElement	No compliance value is defined; extension point.
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
constrainedElement	Element [1]	Constraint	Shall be associated with the UML metaclass instance specified in the *Constraint::constrainedElement* column in Table 26.
context	Namespace [1]	Constraint	Shall be associated with the UML metaclass instance specified in the *Constraint::context* column in Table 26.[1]
specification	ValueSpecification	Constraint	Shall own and shall be associated with an *OpaqueExpression* metaclass instance as defined in Table 28.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

[1] In the *UML Superstructure Specification V2.1.1*, the *context* association is a derived association. An OMG issue has been raised proposing that *context* gets changed to a non-derived association. This document assumes *context* is not derived and therefore defines its mapping.

Table 28 – Usage of UML *OpaqueExpression* Metaclass Used for OCL Related Constraint Qualifiers

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
visibility	VisibilityKind	PackagableElement	No compliance value defined.
name	String [0]	NamedElement	Compliance value: no value.
body	String [1]	OpaqueExpression	Shall be the value of the array entry of the OCL related constraint qualifier.
language	String [1]	OpaqueExpression	Shall be the string "OCL".
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::Boolean.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

The *CIM::ClassConstraint*, *CIM::MethodConstraint* and *CIM::PropertyConstraint* stereotype properties of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.12 Octetstring Qualifier

The datatype of any CIM element on which the CIM *Octetstring* qualifier has an effective value of *true* shall be mapped as defined in 5.11.

The values of CIM elements qualified with *Octetstring* that have a CIM datatype array of uint8 shall be transformed into a hexadecimal string format when represented in the UML element to which the CIM element is mapped. The hexadecimal string format shall conform to the format defined in the *CIM Infrastructure Specification* for octetstring values of type array of string.

The values of CIM elements qualified with *Octetstring* that have a CIM datatype array of string shall be used unchanged when represented in the UML element to which the CIM element is mapped.

The *CIM::Octetstring* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.13 Override Qualifier

The value of any *Override* qualifier defined on a CIM element shall be mapped as follows:

- For CIM properties and references, to the *redefinedProperty* association of the UML *Property* metaclass to which the CIM property or reference carrying the *Override* qualifier is mapped. The *redefinedProperty* association shall reference the UML *Property* metaclass instance to which the CIM property or reference being overridden is mapped.
- For CIM methods, to the *redefinedOperation* association of the UML *Operation* metaclass to which the CIM method carrying the *Override* qualifier is mapped. The *redefinedOperation* association shall reference the UML *Operation* metaclass instance to which the CIM property or reference being overridden is mapped.

NOTE: The class of the CIM element being overridden is either specified in the *Override* qualifier value, or is the next class defining the element when searching towards the superclasses from the CIM class at which level the *Override* qualifier is defined.

The *CIM::Override* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.14 Static Qualifier

The *isStatic* attribute of any UML *Property* and *Operation* metaclass instances shall be equal to the effective value of the *Static* qualifier on the corresponding CIM element.

The *CIM::Static* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.15 Terminal Qualifier

The *isLeaf* attribute of any UML *Class* metaclass instance shall be equal to the effective value of the *Terminal* qualifier on the corresponding CIM element.

The *CIM::Terminal* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.16 Values and ValueMap Qualifiers

If the effective values of the *Values* qualifier or the *ValueMap* qualifier or both are non-NULL, the qualified CIM element shall be mapped using the "*ValueMap mapping*" defined in this subclause.

The UML profile "CIM" shall define and own:

- A UML stereotype *CIM_Enumeration* as defined in Table 30.

In the user model, the qualifier values of *Values* and *ValueMap* shall be mapped to UML enumerations as follows:

- A UML *Enumeration* metaclass instance as defined in Table 29 shall be owned by and defined in the scope of the UML class to which the CIM class owning the qualified CIM element is mapped.
- The stereotype *CIM_Enumeration* defined in Table 30 shall be applied to that UML *Enumeration* metaclass instance.
- The *type* association of the UML metaclasses *Property* and *Parameter* to which the CIM element qualified with *Values* or *ValueMap* is mapped shall be associated with that UML *Enumeration* metaclass instance.
- For each pair of entries in the *Values* or *ValueMap* arrays, a UML *EnumerationLiteral* metaclass instance as defined in Table 32 shall be owned by that UML *Enumeration* metaclass instance.

The UML enumeration is defined in the scope of the CIM class that owns the qualified CIM element in order to support the downward compatible changes to the CIM Schema defined in the *CIM Infrastructure Specification* without changing the name of the enumeration (e.g., moving the qualified element into a superclass).

The representation of *Values* and *ValueMap* qualifier values as UML *EnumerationLiteral* metaclass instances always has both name and value specification, regardless of whether only one of the *Values* and *ValueMap* qualifiers was defined when importing from CIM MOF. The stereotype properties *IsValuesDefined* and *IsValueMapDefined* allow supporting round-trip engineering between CIM MOF format and UML model. They may be used during export of a UML model to CIM MOF format for deciding whether the *Values* and *ValueMap* qualifiers need to be generated.

CIM allows the introduction of a ValueMap qualifier on an overriding element (i.e., if there was no valueMap qualifier defined on the overridden element). This is supported in a type compatible way by the mapping to UML such that the UML *Enumeration* metaclass instance used as a UML datatype for the overriding element is a specialization of the CIM datatype used for the overridden element.

Table 29 – UML *Enumeration* Metaclass Used for *Values* and *ValueMap* Qualifiers

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
isAbstract	Boolean	Classifier	Compliance value: false (UML default).
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	If the qualified CIM element is a property or a method, *name* shall be the name of the element in unchanged lexical case, followed by the string "_Enum". If the qualified CIM element is a parameter, *name* shall be the name of the method in unchanged lexical case, followed by the string "_", followed by the name of the parameter in unchanged lexical case, followed by the string "_Enum".
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
ownedLiteral	EnumerationLiteral [*]	Enumeration	Shall own and shall be associated with an *EnumerationLiteral* metaclass instance for each pair of *Values* and *ValueMap* qualifier values. The order of pairs shall be preserved.
ownedAttribute	Property [0]	DataType	Compliance value: no associated elements.
ownedOperation	Operation [0]	DataType	Compliance value: no associated elements.
generalization	Generalization [1]	Classifier	Shall own and shall be associated with a *Generalization* metaclass instance as defined in 5.16.6 whose *general* association shall be associated with a *Classifier* as follows: If the CIM element qualified with *ValueMap* is overriding another element that has an effective *ValueMap* value other than NULL, with the *Enumeration* instance to which that overridden *ValueMap* value is mapped. Otherwise, with an instance of the *DataType* metaclass instance to which the CIM datatype of the element qualified with *ValueMap* is mapped, as defined in Table 8. In other words, the first Enumeration in an override chain specializes the CIM datatype, and any further Enumerations specialize the previous Enumeration.
redefinedClassifier	Classifier [0]	Classifier	Compliance value: no associated elements.

UML Association	UML Type	Defined in UML Metaclass	CIM Mapping
package	Package [0]	Type (since UML 2.1)	Compliance value: no associated elements. Note that the reason for this is that the *Enumeration* is owned by a *Class*, not by a *Package*. Note that the derived association end *owner* allows traversing to the *Class* owning this *Enumeration*.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
ownedRule	Constraint [*]	Namespace	No compliance value is defined; extension point.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
Stereotype Property	**UML Type**	**Defined in Stereotype**	**CIM Mapping**
IsValuesDefined	Boolean	CIM_Enumeration	Shall be *True* (default) if the *Values* qualifier is to be defined on the qualified CIM element in its MOF representation, and *False* otherwise. If set to False, the *name* attribute of each enumeration literal of this enumeration shall match the value associated with the *specification* association.
IsValueMapDefined	Boolean	CIM_Enumeration	Shall be *True* (default) if the *ValueMap* qualifier is defined on the qualified CIM element, and *False* otherwise. If set to False, the value associated with the *specification* association of each enumeration literal shall be an ordered list starting with 0 and incrementing by 1.

Table 30 – *CIM_Enumeration* Stereotype Used for *Enumeration* Metaclass Used for *Values* and *ValueMap* Qualifiers

UML attribute	UML Infrastructure Library type	Defined in UML Infrastructure Library metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false (UML default).
isLeaf	Boolean	Not defined in *CIM Infrastructure Specification* at this point	If supported by the UML tool then compliance value: false.
name	String	NamedElement	Shall be "CIM_Enumeration".
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).

UML Association	UML Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
ownedAttribute	Property [*]	Class	Shall own and shall be associated with the stereotype properties defined in Table 31.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [0]	Class	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [0]	UML *Stereotype* metaclass	Compliance value: no associated elements.

970 **Table 31 – Properties of Stereotype *CIM_Enumeration***

Name	Type	Default Value	Meaning
IsValuesDefined	Boolean	True	Indicates that the *Values* qualifier is defined on the CIM element.
IsValueMapDefined	Boolean	True	Indicates that the *ValueMap* qualifier is defined on the CIM element.

971 **Table 32 – UML *EnumerationLiteral* Metaclass Used for Pair of *Values***
972 **and *ValueMap* Qualifier Values**

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String	NamedElement	Shall be the value of the *Values* qualifier of the pair, if a *Values* qualifier is defined. Otherwise, shall be the value of the *ValueMap* qualifier of the pair. In any case, the backslash character ('\') as well as any characters or character sequences invalid for UML names (for example, double colons) need to be escaped with a backslash.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
enumeration	Enumeration [1]	EnumerationLiteral	Shall be associated with the UML Enumeration metaclass instance owning this EnumerationLiteral metaclass instance.
classifier	Classifier [1]	InstanceSpecification	No compliance value defined. NOTE: OMG issue #9841 suggests two alternatives on how to constrain this association.
slot	Slot [0]	InstanceSpecification	Compliance value: no associated elements. NOTE: This is not defined in the UML Superstructure Specification but suggested in OMG issue #9841.

973

UML Association	UML Type	Defined in UML Metaclass	CIM Mapping
specification	ValueSpecification [1]	InstanceSpecification	Shall own and shall be associated with an instance of an appropriate subclass of the UML *ValueSpecification* metaclass whose *value* attribute shall be set as follows: To the value of the *ValueMap* qualifier of the pair, if a *ValueMap* qualifier is defined. Non-range values should be represented as integers, and range values as strings. Otherwise, to an integer value reflecting the relative order of the entry within the *Values* array, starting with 0. Note this is the default rule defined in DSP004. Note that the usage of the *specification* association for the value of the enumeration literal is not defined in the *UML Superstructure Specification* but suggested in OMG issue #9842.
ownedComment	Comment [0..1]	Element	No compliance value is defined; extension point.

The *CIM::ValueMap* and *CIM::Values* stereotype properties of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.4.17 Write Qualifier

The *isReadOnly* attribute of any UML *Property* metaclass instance shall be mapped to the effective value of the *Write* qualifier on the corresponding CIM element, as defined in Table 33.

Table 33 – Mapping of *Write* Qualifier

Value of *Write* Qualifier	Value of *isReadOnly* Attribute
True	False
False	True

The *CIM::Write* stereotype property of the UML metaclass instance mapped to the qualified CIM element shall not be set.

5.13.5 Definition of Marker Stereotypes

As defined in 5.13.4, some directly mapped qualifier types use Marker Stereotypes. This subclause defines the Marker Stereotypes for this purpose.

Table 34 defines the mapping for the attributes and associations of the UML *Stereotype* metaclass for use as a Marker Stereotype.

The UML *Stereotype* metaclass has mostly attributes and associations from the UML Infrastructure Library defined in the *UML Infrastructure Specification*.

989 **Table 34 – UML *Stereotype* Metaclass Used as Marker Stereotype**

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false (UML default).
isLeaf	Boolean	Not defined in the *UML Infrastructure Specification* at this point	If supported by the UML tool then compliance value: false.
name	String	NamedElement	The mapping of this attribute is defined in the description of the qualifier that is using this marker stereotype, in 5.13.4.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property [0..1]	Class	Depends on the usage context for the marker stereotype. The rules are defined in each usage.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [0]	Class	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [1]	UML *Stereotype* metaclass	Shall own and shall be associated with an *Image* metaclass instance as defined in Table 35.

990 **Table 35 – UML *Image* Metaclass Used by Marker Stereotype**

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
content	String [1]	UML *Image* metaclass	Shall contain the serialization of the image of the stereotype icon, in the format defined by the format attribute. The image of the stereotype icon is defined in the description of the qualifier that is using this marker stereotype, in 5.13.4.
format	String [1]	UML *Image* metaclass	Shall contain the format used for the content attribute.
location	String [0]	UML *Image* metaclass	Compliance value: no elements.
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.13.6 Typical Use Cases for Qualifier Inheritance Mapping

This subclause provides algorithms for some typical use cases that are derived from the mapping rules defined in the previous subclause.

5.13.6.1 Use Case: Importing CIM MOF into a UML Tool

Determination of the CIM qualifier value to be used, and whether the UML construct to which the qualifier is mapped, needs to be created:

- For UML requirements level = Required:

  The UML construct is already present. Set its value from the effective qualifier value in the CIM MOF.

- For UML requirements level = Optional:

  If a qualifier value is defined in the CIM MOF with a value other than the default value (for *Restricted* qualifiers) or the inherited value (for *ToSubclass* qualifiers):

  – then create the UML construct and set its value from the qualifier value defined in the CIM MOF;

  – else do not create the UML construct.

NOTE: Determining the inherited value or the effective qualifier value requires knowledge about the superclasses, so these superclasses need to be imported first. This results in automatically propagating the values down the class hierarchy towards the leaf nodes as classes are imported.

For qualifiers with *DisableOverride*, the import logic may validate that the *DisableOverride* constraint is not violated.

5.13.6.2 Use Case: Editing a UML User Model with a UML Tool

Displaying the effective qualifier value at a given class level:

- If the corresponding UML construct exists in the class, then determine the qualifier value from that construct.
- Else, use the default qualifier value (for *Restricted* qualifiers) or the inherited value (for *ToSubclass* qualifiers).

NOTE: It is useful to also display the origin of the effective qualifier value along with the value.

Modification of a qualifier value at a particular class level:

- For UML requirements level = Required:

  The UML construct is already present. Set its value from the new qualifier value.

- For UML requirements level = Optional:

  If the new qualifier value is the default value (for *Restricted* qualifiers) or the inherited value (for *ToSubclass* qualifiers):

  – then remove the UML construct if it exists;

  – else create the UML construct if it does not exist and set its value from the new qualifier value.

5.13.6.3 Use Case: Exporting from a UML Tool into CIM MOF

Determining whether a qualifier value definition should be created in the exported CIM MOF:

- If the CIM qualifier is in the set (Association, Aggregation, Composition, Indication, Exception, In) and its effective value is *True*, then create a qualifier definition in the exported CIM MOF. Note that this is needed for compatibility with existing CIM tools.
- Else, if the corresponding UML construct exists in the class that is exported, AND if the qualifier value is not equal to the default value (for *Restricted* qualifiers) or the inherited value (for *ToSubclass* qualifiers):
  - then create a qualifier definition in the exported CIM MOF;
  - else do not create a qualifier definition in the exported CIM MOF.

5.14 Mapping MOF File Information

CIM MOF files contain information in addition to CIM classes and instances. In order to support roundtripping from CIM MOF to UML and back, this information needs to be represented in UML. This subclause normatively defines how information related to CIM MOF files is mapped to UML.

The following information related to CIM MOF files is mapped to UML:

- The name of the CIM MOF file
- The directory path of the CIM MOF file
- The ordered list of header lines in the CIM MOF file
- The value of the `#pragma locale` directive
- The value of the `#pragma instancelocale` directive
- The value of the `#pragma namespace` directive

The ordered list of header lines is defined to be the lines at the top of the MOF file before the first non-empty and non-comment line.

CIM MOF files have no directly corresponding entity in the UML user model. Therefore, the information items listed above are mapped to properties of stereotypes applied to the classes, instances and packages in the UML user model, as defined in Table 36:

Table 36 – Mapping of CIM MOF File Related Information

CIM MOF File Related Information Item	Stereotype Property	Defined in Stereotype	Applied to UML Metaclass
Name of the CIM MOF file	FileName	CIM_MofFileInfo	Class (required) InstanceSpecification (required) Package (required)
Directory path of the CIM MOF file	DirectoryPath		
Ordered list of header lines in the CIM MOF file	HeaderLines		
Value of the `#pragma locale` directive	ClassLocale	CIM_ClassLocale	Class (required) Package (required)

CIM MOF File Related Information Item	Stereotype Property	Defined in Stereotype	Applied to UML Metaclass
Value of the `#pragma instancelocale` directive	InstanceLocale	CIM_InstanceLocale	InstanceSpecification (required) Package (required)
Value of the `#pragma namespace` directive	NamespacePath	CIM_NamespacePath	Class (required) InstanceSpecification (required) Package (required)

These stereotypes and their properties are defined in 5.14.1.

As defined in Table 36, each such information item may be set at a package level in addition to the class or instance level. The value of each such information item in effect for a CIM class or CIM instance shall be determined as follows:

1) If the information item has a non-null value at the class or instance, that value is used.
2) Else, if the information item has a non-null value at the package containing the class or instance, that value is used.
3) Else, the packages are searched outwards until a non-null value is found.
4) If no non-null value is found at the outermost package of the UML user model, the effective value is null.

This algorithm is used for each information item separately, so the values in effect for a class or instance may be defined at different levels.

The information items are represented from the point of view of a CIM class or CIM instance. This does not allow reconstructing a CIM MOF file exactly as it was before importing it into UML. However, it allows reconstructing a CIM MOF file that has equivalent information as the imported CIM MOF file. For example, two consecutive `#pragma locale` directives before a CIM class definition in the original CIM MOF file get mapped to a single *ClassLocale* property of that class in the UML user model, and result in a single `#pragma locale` directive when exported to CIM MOF.

5.14.1 Stereotypes Used for Mapping MOF File Information

5.14.1.1 Stereotype *CIM_ClassLocale*

The stereotype *CIM_ClassLocale* represents a `#pragma locale` directive in the CIM MOF file. This stereotype shall be defined in the UML profile "CIM", as defined in Table 37:

Table 37 – Stereotype *CIM_ClassLocale*

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false.
name	String	NamedElement	Shall be the string "CIM_ClassLocale".
Other attributes as defined in Table 47.			

UML Association	UML Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
ownedAttribute	Property	Class	Shall be associated with and own the property defined in Table 38.
Other associations as defined in Table 47.			

1077 **Table 38 – Property *ClassLocale* Owned by Stereotype *CIM_ClassLocale***

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String	NamedElement	Shall be the string "ClassLocale".
Other attributes as defined in Table 48.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

1078 5.14.1.2 Stereotype *CIM_InstanceLocale*

1079 The stereotype *CIM_InstanceLocale* represents a `#pragma instancelocale` directive in the CIM MOF
1080 file. This stereotype shall be defined in the UML profile "CIM", as defined in Table 39:

1081 **Table 39 – Stereotype *CIM_InstanceLocale***

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false.
name	String	NamedElement	Shall be the string "CIM_InstanceLocale".
Other attributes as defined in Table 47.			
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property	Class	Shall be associated with and own the property defined in Table 40.
Other associations as defined in Table 47.			

1082 **Table 40 – Property *InstanceLocale* Owned by Stereotype *CIM_InstanceLocale***

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String	NamedElement	Shall be the string "InstanceLocale".
Other attributes as defined in Table 48			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

1083 ### 5.14.1.3 Stereotype *CIM_NamespacePath*

1084 The stereotype *CIM_NamespacePath* represents a `#pragma namespace` directive in the CIM MOF file.
1085 This stereotype shall be defined in the UML profile "CIM", as defined in Table 41:

1086 **Table 41 – Stereotype *CIM_NamespacePath***

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	Compliance value: false.
name	String	NamedElement	Shall be the string "CIM_NamespacePath".
Other attributes as defined in Table 47.			
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property	Class	Shall be associated with and own the property defined in Table 42.
Other associations as defined in Table 47.			

Table 42 – Property *NamespacePath* Owned by Stereotype *CIM_NamespacePath*

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
name	String	NamedElement	Shall be the string "NamespacePath".
Other attributes as defined in Table 48.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

5.14.1.4 Stereotype *CIM_MofFileInfo*

The stereotype *CIM_MofFileInfo* represents information related to a CIM MOF file. This stereotype shall be defined in the UML profile "CIM", as defined in Table 43:

Table 43 – Stereotype *CIM_MofFileInfo*

UML Attribute	**UML Infrastructure Library Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
isAbstract	Boolean	Class	Compliance value: false.
name	String	NamedElement	Shall be the string "CIM_MofFileInfo".
Other attributes as defined in Table 47.			
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property [3]	Class	Shall be associated with and own the properties defined in Table 44, Table 45 and Table 46.
Other associations as defined in Table 47.			

1092 **Table 44 – Property *FileName* Owned by Stereotype *CIM_MofFileInfo***

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
name	String	NamedElement	Shall be the string "FileName".
Other attributes as defined in Table 48.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

1093 **Table 45 – Property *DirectoryPath* Owned by Stereotype *CIM_MofFileInfo***

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
name	String	NamedElement	Shall be the string "DirectoryPath".
Other attributes as defined in Table 48.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

Table 46 – Property *HeaderLines* Owned by Stereotype *CIM_MofFileInfo*

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
name	String	NamedElement	Shall be the string "HeaderLines".
Other attributes as defined in Table 48.			
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
defaultValue	ValueSpecification	Property	Shall own and shall be associated with an instance of the *LiteralNull* metaclass.
type	Type	TypedElement	Shall be associated with the *PrimitiveType* metaclass instance representing the datatype UML::String.
lowerValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
upperValue	ValueSpecification	MultiplicityElement	Shall have the value "1" in a UML2 defined representation.
Other associations as defined in Table 48.			

5.14.1.5 Other Definitions for Stereotypes Used for Mapping MOF File Information

The stereotypes and their properties used for mapping MOF file information shall be compliant with the additional definitions for their attributes and associations, as defined in Table 47 and Table 48.

Table 47 – Other Definitions for Stereotypes Used for Mapping MOF File Information

UML Attribute	**UML Infrastructure Library Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
isAbstract	Boolean	Class	Compliance value: false.
isLeaf	Boolean	Not defined in the *UML Infrastructure Specification* at this point	If supported by the UML tool then compliance value: false.
name	String	NamedElement	As defined in the table referring to this table.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property	Class	As defined in the table referring to this table.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [0]	Class	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [0]	UML *Stereotype* metaclass	Compliance value: no associated elements.

Table 48 – Other Definitions for Properties of Stereotypes Used for Mapping MOF File Information

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
aggregation	AggregationKind	Property	Compliance value: none (UML default).
isDerived	Boolean	Property	Compliance value: false (UML default).
isDerivedUnion	Boolean	Property	Compliance value: false (UML default).
isReadOnly	Boolean	Property	Compliance value: false (UML default).
isStatic	Boolean	Feature	Compliance value: false (UML default).
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	As defined in the table referring to this table.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
isOrdered	Boolean	MultiplicityElement	Compliance value: false (UML default).
isUnique	Boolean	MultiplicityElement	Compliance value: false (UML default is true).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
association	Association [0]	Property	Compliance value: no associated elements.
owningAssociation	Association [0]	Property	Compliance value: no associated elements.
datatype	DataType [0]	Property	Compliance value: no associated elements (because this property is not owned by a structured datatype).
defaultValue	ValueSpecification	Property	As defined in the table referring to this table.
redefinedProperty	Property [0]	Property	Compliance value: no associated elements.
subsettedProperty	Property [0]	Property	Compliance value: no associated elements.
associationEnd	Property [0]	Property	Compliance value: no associated elements.
qualifier	Property [0]	Property	Compliance value: no associated elements.
type	Type	TypedElement	As defined in the table referring to this table.
lowerValue	ValueSpecification	MultiplicityElement	As defined in the table referring to this table.
upperValue	ValueSpecification	MultiplicityElement	As defined in the table referring to this table.
ownedComment	Comment [*]	Element	Compliance value: no associated elements.
class (since UML 2.1)	Class	Property	Shall be owned by and associated with the *Stereotype* metaclass instance owning the property.

5.15 Stereotypes for the CIM Meta Elements

This subclause normatively defines those stereotypes that may be applied to the CIM meta elements. These stereotypes are called "Meta Element Stereotypes" in this document.

NOTE: Some of them inherit from the Qualifier Scope Stereotypes. This is how the CIM meta elements get the ability to have CIM qualifier values.

The UML profile "CIM" shall define and own the stereotypes defined in Table 49. As an extension point, these stereotypes may define additional properties, and may inherit from additional base stereotypes.

1107 **Table 49 – Meta Element Stereotypes**

Stereotype Name	Abstract	Extends UML Metaclass	Base Stereotypes and Properties
CIM_Package	No	Package (Required)	No base stereotypes required
CIM_Schema	No	Package (Optional)	No base stereotypes required
CIM_Class	No	Class (Optional)	CIM_Class_Qualifiers
CIM_Association	No	AssociationClass (Optional)	CIM_Association_Qualifiers
CIM_Indication	No	Class (Optional)	CIM_Indication_Qualifiers
CIM_Property	No	Property (Optional)	CIM_Property_Qualifiers CIM_ClassReferenceType
CIM_Reference	No	Property (Optional)	CIM_Reference_Qualifiers
CIM_Method	No	Operation (Required)	CIM_Method_Qualifiers
CIM_Parameter	No	Parameter (Optional)	CIM_Parameter_Qualifiers CIM_ClassReferenceType
CIM_Instance	No	InstanceSpecification (Optional)	No base stereotypes required
CIM_Enumeration	No	Enumeration (Optional)	No base stereotypes required

1108 Any other attributes or associations of the Meta Element Stereotypes shall be as defined in Table 50.

1109 **Table 50 – Definition of Meta Element Stereotypes**

UML Attribute	UML Infrastructure Library Type	Defined in UML Infrastructure Library Metaclass	CIM Mapping
isAbstract	Boolean	Class	As defined in the table referring to this table.
isLeaf	Boolean	Not defined in the *UML Infrastructure Specification* at this point	If supported by the UML tool then compliance value: false.
name	String	NamedElement	As defined in the table referring to this table.
visibility	VisibilityKind	NamedElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Infrastructure Library Metaclass**	**CIM Mapping**
ownedAttribute	Property [*]	Class	As defined in the table referring to this table.
ownedOperation	Operation [0]	Class	Compliance value: no associated elements.
superClass	Class [*]	Class	As defined in the table referring to this table.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.
icon	Image [*]	UML *Stereotype* metaclass	No compliance value is defined; extension point.

5.16 Other UML Metaclasses Used in the Mapping

This subclause normatively defines the usage of other UML metaclasses in addition to those defined in the previous subclauses.

5.16.1 UML *LiteralBoolean* Metaclass

The UML *LiteralBoolean* metaclass as defined in this subclause is used to represent boolean values in the following UML metaclasses:

- *defaultValue* association in UML *Property* metaclass used for CIM properties, as defined in Table 4.
- *value* association in UML *Slot* metaclass used for property values of modeled CIM instances, as defined in Table 10.
- *defaultValue* association in UML *Property* metaclass used for CIM qualifiers in Qualifier Type Stereotypes, as defined in Table 12.

Table 51 defines the mapping for the attributes and associations of the UML *LiteralBoolean* metaclass for these usages.

Table 51 – UML *LiteralBoolean* Metaclass

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
value	Boolean	LiteralBoolean	Shall be the boolean value to be represented.
name	String [0]	NamedElement	Compliance value: no value.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type [0]	TypedElement	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.16.2 UML *LiteralString* Metaclass

The UML *LiteralString* metaclass as defined in this subclause is used to represent string values in the following UML metaclasses:

- *defaultValue* association in UML *Property* metaclass used for CIM properties, as defined in Table 4.
- *value* association in UML *Slot* metaclass used for property values of modeled CIM instances, as defined in Table 10.
- *defaultValue* association in UML *Property* metaclass used for CIM qualifiers in Qualifier Type Stereotypes, as defined in Table 12.

Table 52 defines the mapping for the attributes and associations of the UML *LiteralString* metaclass for these usages.

Table 52 – UML *LiteralString* Metaclass

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
value	String [0..1]	LiteralString	Shall be the string value to be represented. If the *value* attribute has no value, the string shall be interpreted as an empty string.
name	String [0]	NamedElement	Compliance value: no value.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type [0]	TypedElement	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.16.3 UML *LiteralInteger* Metaclass

The UML *LiteralInteger* metaclass as defined in this subclause is used to represent integer values in the following UML metaclasses:

- *defaultValue* and *lowerValue* associations in UML *Property* metaclass used for CIM properties, as defined in Table 4.
- *lowerValue* association in UML *Property* metaclass used for CIM references, as defined in Table 5.
- *lowerValue* association in UML *Parameter* metaclass used for CIM parameters and CIM return values, as defined in Table 7.
- *value* association in UML *Slot* metaclass used for property values of modeled CIM instances, as defined in Table 10.
- *defaultValue* and *lowerValue* associations in UML *Property* metaclass used for CIM qualifiers in Qualifier Type Stereotypes, as defined in Table 12.

Table 53 defines the mapping for the attributes and associations of the UML *LiteralInteger* metaclass for these usages.

Table 53 – UML *LiteralInteger* Metaclass

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
value	Integer	LiteralInteger	Shall be the integer value to be represented.
name	String [0]	NamedElement	Compliance value: no value.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type [0]	TypedElement	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.16.4 UML *LiteralUnlimitedNatural* Metaclass

The UML *LiteralInteger* metaclass as defined in this subclause is used to unlimited natural values in the following UML metaclasses:

- *upperValue* association in UML *Property* metaclass used for CIM properties, as defined in Table 4.
- *upperValue* association in UML *Property* metaclass used for CIM references, as defined in Table 5.
- *upperValue* association in UML *Parameter* metaclass used for CIM parameters and CIM return values, as defined in Table 7.
- *upperValue* association in UML *Property* metaclass used for CIM qualifiers in Qualifier Type Stereotypes, as defined in Table 12.

Table 54 defines the mapping for the attributes and associations of the UML *LiteralUnlimitedNatural* metaclass for these usages.

Table 54 – UML *LiteralUnlimitedNatural* Metaclass

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
value	UnlimitedNatural	LiteralUnlimitedNatural	Shall be the unlimited natural value to be represented.
name	String [0]	NamedElement	Compliance value: no value.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type [0]	TypedElement	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.16.5 UML *LiteralNull* Metaclass

The UML *LiteralNull* metaclass as defined in this subclause is used to represent CIM NULL values in the following UML metaclasses:

- *defaultValue* association in UML *Property* metaclass used for CIM properties, as defined in Table 4,
- *value* association in UML *Slot* metaclass used for property values of modeled CIM instances, as defined in Table 10,
- *defaultValue* association in UML *Property* metaclass used for CIM qualifiers in Qualifier Type Stereotypes, as defined in Table 12.

Table 55 defines the mapping for the attributes and associations of the UML *LiteralNull* metaclass for these usages.

Table 55 – UML *LiteralNull* Metaclass

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
name	String [0]	NamedElement	Compliance value: no value.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
type	Type [0]	TypedElement	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.16.6 UML *Generalization* Metaclass

The UML *Generalization* metaclass as defined in this subclause is used by the following UML metaclasses:

- *generalization* association in UML *Class* metaclass used for CIM classes and indications,
- *generalization* association in UML *AssociationClass* metaclass used for CIM association classes.
- *generalization* association in UML *Enumeration* metaclass used for CIM *ValueMap* qualifier values.

Table 56 defines the mapping for the attributes and associations of the UML *Generalization* metaclass for these usages.

Table 56 – UML *Generalization* Metaclass

UML Attribute	UML Type	Defined in UML Metaclass	CIM Mapping
isSubstitutable	Boolean [1]	Generalization	Compliance value: true. NOTE: UML 2.1.1 does not define a default value for this attribute, OMG issue #9665 has been raised, targeting UML 2.2.
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
general	Classifier	Generalization	Shall be associated with the general *Classifier* as defined where the *Generalization* metaclass is used.
specific	Classifier	Generalization	Shall be associated with the specific *Classifier* as defined where the *Generalization* metaclass is used.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

NOTE: The color and other graphical appearance of the generalization link are not normatively defined in this document, because the UML Superstructure Specification does not allow specifying the graphical appearance. It is recommended that tools apply the DMTF conventions for the color of generalization links (i.e., blue).

5.16.7 UML *DataType* Metaclass

The UML *DataType* metaclass as defined in this subclause is used by the following metaclasses:

- *type* association in UML *Property* metaclass used for CIM properties and references,
- *type* association in UML *Parameter* metaclass used for CIM parameters.

Table 57 defines the mapping for the attributes and associations of the UML *DataType* metaclass for these usages.

Table 57 – UML *DataType* Metaclass

UML Attribute	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
isAbstract	Boolean	Classifier	Compliance value: false (UML default).
isLeaf	Boolean	RedefinableElement	Compliance value: false (UML default).
name	String	NamedElement	Shall be the name of the CIM datatype.
visibility	VisibilityKind	PackagableElement	Compliance value: public (UML default is private).
UML Association	**UML Type**	**Defined in UML Metaclass**	**CIM Mapping**
ownedAttribute	Property [0]	DataType	Compliance value: no associated elements.
ownedOperation	Operation [0]	DataType	Compliance value: no associated elements.
generalization	Generalization [0]	Classifier	Compliance value: no associated elements.
package	Package [0..1]	Classifier	Shall be owned by and associated with the UML *Package* metaclass instance representing the type library named "CIMDatatypes", as defined in 5.21.3.
redefinedClassifier	Classifier [0]	Classifier	Compliance value: no associated elements.
elementImport	ElementImport [0]	Namespace	Compliance value: no associated elements.
ownedRule	Constraint [*]	Namespace	No compliance value defined, extension point.
packageImport	PackageImport [0]	Namespace	Compliance value: no associated elements.
ownedComment	Comment [0]	Element	Compliance value: no associated elements.

5.17 Constraints

This subclause defines constraints that shall be met in order for a UML user model to be a valid CIM model. These constraints fall into three categories:

- OCL constraints for CIM – Constraints specific to the UML profile for CIM, defined using Object Constraint Language (OCL). This is the main category of constraints.
- Other constraints for CIM – Constraints specific to the UML profile for CIM, defined using normative text. This category only exists for constraints for which it was not possible to define an according OCL statement.
- Constraints for UML – general UML constraints not specific to the UML profile for CIM. They are only listed in order to show which CIM rules are already covered by UML.

A conforming implementation of the UML profile for CIM shall support all OCL constraints for CIM and should support all other constraints for CIM.

NOTE: OCL is used as a specification language in this document. Conforming implementations of the UML profile for CIM may use other OCL statements or constraint implementations using environments other than OCL as long as they produce an equivalent result.

The OCL language used in the definition of the OCL constraints for CIM is defined in the *OCL Specification*. In addition, some OCL query functions are used as defined in the *UML Superstructure Specification*. Any remaining OCL functions used in this subclause are defined in 5.17.1.

All OCL constraints for CIM specify the UML element owning the UML *Constraint* metaclass instance containing the OCL statement. In some cases, these are specific stereotypes defined in this document. In the remaining cases, these are stereotypes defined by the implementation of the UML profile for CIM and the context is then defined in terms of UML metaclasses. For example, the context "Stereotype extending *Class*" means that "`self`" in the OCL statement refers to a stereotype that extends the UML *Class* metaclass.

All OCL statements in this subclause are invariants.

NOTE 1: A note on accessing the metaclass instance to which a stereotype is applied, from within OCL constraints on such stereotypes: According to OMG, the term "`self`" in such OCL constraints refers to the applied stereotype and not to the metaclass instance to which the stereotype is applied. This means that the "`base_<metaclass>`" relationship needs to be traversed explicitly in OCL statements in order to access the metaclass instance to which the stereotype is applied. Some UML tools however define "`self`" to refer to a merge of the applied stereotype and the metaclass instance to which the stereotype is applied. This allows omitting the "`base_<metaclass>`" relationship traversal in the OCL statement. The OCL statements used in this document follow the OMG direction and use explicit traversal of the "`base_<metaclass>`" relationship.

NOTE 2: As defined in the *CIM Infrastructure Specification*, the term "CIM classes" includes CIM associations and CIM indications in addition to "normal" CIM classes, and the term "CIM properties" includes CIM references in addition to "normal" properties.

NOTE 3: The OCL language allows traversing associations between UML metaclasses in both directions, regardless of whether or not the association ends are owned by the association or the associated class. However, OCL engines may only support traversal from association ends owned by the associated UML metaclasses, but not from ends owned by the associations themselves. The OCL in any constraints defined in this document accomodates for such OCL engines in that it only traverses association ends owned by the associated UML metaclasses.

Any ABNF rules defined in the remainder of this subclause use the conventions defined in 3.4.

5.17.1 Additional OCL Functions

For the purpose of simplifying the specification of constraints using OCL, some OCL statements use the following OCL functions in addition to the OCL syntax defined in the *OCL Specification*.

5.17.1.1 matchCimAbnf

```
String::matchCimAbnf( cimAbnfRuleName : String) : Boolean
```

Matches the target string against the ABNF rule whose name is specified in `cimAbnfRuleName` and returns `true` if the target string matches the format, else `false`. The ABNF rules referenced in `cimAbnfRuleName` shall be in the set of rules defined in this document or in the set of rules defined in the *CIM Infrastructure Specification*.

If `cimAbnfRuleName` does not reference a valid ABNF rule, the function returns `null`.

5.17.1.2 extractCimAbnf

```
String::extractCimAbnf( cimAbnfRuleName : String,
    extractedCimAbnfRuleName : String) : String
```

Extracts the substring from the target string that matches the ABNF rule named in `extractedCimAbnfRuleName` and returns the extracted substring. The target string shall match the ABNF rule named in `cimAbnfRuleName`. The ABNF rules referenced in `extractedCimAbnfRuleName` and `cimAbnfRuleName` shall be in the set of rules defined in this document or in the set of rules defined in the *CIM Infrastructure Specification*.

If the ABNF rule named in `cimAbnfRuleName` is specified with a multiplicity (i.e., using the star (*) in the ABNF), a list of the matching substrings is returned by the OCL function, in the order they are specified in the target string.

If `extractedCimAbnfRuleName` or `cimAbnfRuleName` do not reference a valid ABNF rule, or if `extractedCimAbnfRuleName` references a rule that is not used in the ABNF rule referenced in `cimAbnfRuleName`, the function returns `null`.

5.17.1.3 hexToInteger

```
String::hexToInteger() : Integer
```

Converts the target string into an integer number. The target string shall match the ABNF format:

```
   *(hexDigit hexDigit)
```

and is further constrained by the value range that can be represented using the OCL Integer datatype. If it does not match that format, or if the value range would be exceeded, the function returns `null`.

5.17.1.4 isValidCimReal32

```
Integer::isValidCimReal32() : Boolean
```

Tests whether the target integer value is a valid CIM real32 number and returns true if that is the case, else false.

5.17.1.5 isValidCimReal64

```
Integer::isValidCimReal64() : Boolean
```

Tests whether the target integer value is a valid CIM real64 number and returns true if that is the case, else false.

5.17.1.6 isValidCimString

```
String::isValidCimString() : Boolean
```

Tests for each character in the string whether it is a Unicode character from the UCS-2 character set and returns true if that is the case, else false.

5.17.1.7 isValidCimDatetime

```
String::isValidCimDatetime() : Boolean
```

Tests whether the string is a valid CIM datetime string and returns true if that is the case, else false.

5.17.1.8 isValidCimElementName

```
String::isValidCimElementName() : Boolean
```

Tests whether the string is a valid CIM element name and returns true if that is the case, else false.

The *CIM Infrastructure Specification* defines valid CIM element names as follows:

- The name of a CIM element shall consist of the following characters: "_", "A", ... , "Z", "a", ... , "z", "0", ... , "9", U+0080, ... , U+FFEF
- The name of a CIM element shall start with one of the following characters: "_", "A", ... , "Z", "a", ... , "z", U+0080, ... , U+FFEF
- The name of a CIM element shall be at least one character long.

These rules apply to the names of all CIM element types.

Some CIM element types have additional rules. For example, the name of a CIM class shall have an underscore after its schema name part. Such additional rules are validated by additional OCL constraints.

5.17.1.9 isSet

```
Element::isSet(s : Stereotype, propertyName : String) : Boolean
```

Tests whether the stereotype property has a value in the stereotype that is applied to the target element and returns true if that is the case, else false.

NOTE: This function is not testing whether the property has a non-default value, nor whether the property is non-null. An example where this distinction can be seen is an integer typed stereotype property with a default value of 0. If the property has no value set, isSet() returns false. If the property has a value of 0 set isSet() returns true. In both cases, a function that returns the value of the property returns 0.

5.17.1.10 getAppliedStereotype

```
Element::getAppliedStereotype(qualifiedName : String) : Stereotype
```

Returns the stereotype with the specified qualified name that is applied to the target element, or null if no such stereotype is applied.

The qualified name of a stereotype is its profile name, followed by "::", followed by the stereotype name.

5.17.1.11 getAllAttributes

```
Classifier::getAllAttributes() : Set(Property)
```

Returns the subset of allFeatures() that are properties. Note that this includes inherited properties.

5.17.1.12 getAllOperations

```
Classifier::getAllOperations() : Set(Operation)
```

Returns the subset of allFeatures() that are operations. Note that this includes inherited operations.

5.17.1.13 getAllSuperClasses

```
Class::getAllSuperClasses() : OrderedSet(Class)
```

Returns the set of all superclasses of the target class. The list is ordered from the direct superclasses to the most generalized superclasses.

Note that while the definition of this function covers the case of multiple superclasses, a CIM class may have at most one superclass (single inheritance).

5.17.2 OCL Constraints on Classes, Associations, Indications and their Instances

C1. A non-association CIM class shall have either the *CIM_Class* or the *CIM_Indication* stereotype applied.

Context: A stereotype extending UML *Class*

OCL:
```
self.base_Class.oclIsTypeOf(uml::AssociationClass) = false
    implies
    Bag {
       self.base_Class.getAppliedStereotype('CIM::CIM_Class'),
       self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
    }->select( s | s <> null )->size() = 1
```

C2. A CIM association shall have the *CIM_Association* stereotype applied.

Context: A stereotype extending UML *Association*

OCL:
```
self.base_Association.getAppliedStereotype('CIM::CIM_Association')
       <> null
```

C3. A CIM association shall be a UML association class (not a plain association link).

Context: A stereotype extending UML *Association*

OCL:
```
self.base_Association.oclIsTypeOf(uml::AssociationClass)
```

C4. A CIM class shall have no more than one superclass.

Context: A stereotype extending UML *Class*

OCL:
```
self.base_Class.generalization->size() <= 1
```

C5. The superclass of a normal (i.e., non-association, non-indication) CIM class shall be a normal CIM class.

Context: Stereotype *CIM_Class*

OCL:
```
self.base_Class.generalization.general->forAll( sc |
    sc.getAppliedStereotype('CIM::CIM_Class') <> null )
```

C6. The superclass of a CIM indication shall be a CIM indication.

Context: Stereotype *CIM_Indication*

OCL:
```
self.base_Class.generalization.general->forAll( sc |
    sc.getAppliedStereotype('CIM::CIM_Indication') <> null )
```

C7. The superclass of a CIM association shall be a CIM association.

Context: Stereotype *CIM_Association*

OCL:
```
self.base_AssociationClass.generalization.general->forAll( sc |
    sc.getAppliedStereotype('CIM::CIM_Association') <> null )
```

C8. A non-abstract non-indication CIM class shall have key properties.

Context: Stereotype *CIM_Class*

OCL:
```
self. base_Class.isAbstract = false
    implies
    self.base_Class
    ->union(self.base_Class.getAllSuperClasses())
```

```
        ->select( c | c.ownedMember
          ->any( p | p.getAppliedStereotype('CIM::Key')<>null)->size()>0)
        ->size() > 0
```

Context: Stereotype *CIM_Association*

OCL:
```
self.base_AssociationClass.isAbstract = false
implies
self.base_AssociationClass
->union(self.base_AssociationClass.getAllSuperClasses())
->select( c | c.ownedMember
  ->any( p | p.getAppliedStereotype('CIM::Key')<>null)->size()>0)
->size() > 0
```

C9. A CIM class with key properties shall not have a superclass with key properties.

NOTE: getAllAttributes() returns also any inherited properties

Context: A stereotype extending UML *Class*

OCL:
```
let nk : Integer = self.base_Class.ownedAttribute
  ->select( p | p.getAppliedStereotype('CIM::Key') <> null)->size()
in
nk > 0
implies
self.base_Class.getAllAttributes()->select( p |
  p.getAppliedStereotype('CIM::Key') <> null
)->size() = nk
```

C10. The name of a CIM class shall conform to the format for CIM class names defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in the *CIM Infrastructure Specification* is:

```
className = schemaName "_" IDENTIFIER;
```

In addition, the `schemaName` shall not include "_", and both `schemaName` and `IDENTIFIER` shall conform to the rules for CIM element names as defined in 5.17.1.8.

Context: A stereotype extending UML *Class*

OCL:
```
let cnsize : Integer = /* size of class name */
  self.base_Class.name.size()
in
let upos : Integer = /* position of first '_' in class name */
  Sequence { 1 .. cnsize }->any( i |
    self.base_Class.name.substring( i, i) = '_')
in
if upos.oclIsUndefined() /* no '_' found */
then false
else
  let schema : String = /* schema name of class */
    self.base_Class.name.substring( 1, upos - 1)
  in
  let sname : String = /* short class name */
    self.base_Class.name.substring( upos + 1, cnsize)
```

```
          in
          schema.isValidCimElementName() and
          sname.isValidCimElementName()
        endif
```

C11. A CIM class shall be in a package path that has *CIM_Schema* applied somewhere.

Context: A stereotype extending UML *Class*

OCL:
```
self.base_Class.allNamespaces()->
    select( c | c.getAppliedStereotype('CIM::CIM_Schema') <> null)->
    size() > 0
```

C12. The schema name of a CIM class shall be the name of the next outer package that has *CIM_Schema* applied.

Note: As with all CIM names, this comparison is done in a case-insensitive way.

Context: A stereotype extending UML *Class*

OCL:
```
let upos : Integer = /* position of first '_' in class name */
    Sequence { 1 .. self.base_Class.name.size() }->any( i |
      self.base_Class.name.substring( i, i) = '_')
in
if upos.oclIsUndefined() /* no '_' found */
then false
else
  let schema : String = /* schema name of class */
    self.base_Class.name.substring( 1, upos - 1)
  in
  schema.toUpper() = self.base_Class.allNamespaces()->select( c |
    c.getAppliedStereotype('CIM::CIM_Schema') <> null)-
      >at(1).name.toUpper()
endif
```

C13. All classes in the package subtree under a CIM schema shall have unique names.

Context: Stereotype *CIM_Schema*, extending any *Package* instance that is supposed to contain an entire CIM schema

NOTE 1: A "CIM schema" in this context is any schema, not just the CIM Schema defined by DMTF.

NOTE 2: As with all CIM names, the uniqueness is tested for in a case-insensitive way.

OCL:
```
Class.allInstances()->
    select(c | c.allNamespaces()->includes(self.base_Package))->
    isUnique(c | c.name.toUpper())
```

C14. A CIM class shall not have nested classifiers other than *Enumerations.*

Context: A stereotype extending UML *Class*

OCL:
```
self.base_Class.nestedClassifier
    ->select( c | c.oclIsTypeOf(uml::Enumeration) = false)
    ->isEmpty()
```

C15. A CIM class shall have public visibility.

Context: A stereotype extending UML *Class*

OCL: `self.base_Class.visibility = uml::VisibilityKind::public`

C16. A CIM class shall not redefine another classifier.

Context: A stereotype extending UML *Class*

OCL: `self. base_Class.redefinedClassifier->isEmpty()`

C17. A CIM association shall have the UML qualifier isDerived set to false.

Context: Stereotype *CIM_Association*

OCL: `self. base_AssociationClass.isDerived = false`

C18. A generalization shall not have a comment.

Context: A stereotype extending UML *Generalization*

OCL: `self. base_Generalization.ownedComment->isEmpty()`

C19. A generalization shall be substitutable.

Context: A stereotype extending UML *Generalization*

OCL: `self. base_Generalization.isSubstitutable`

C20. A CIM instance shall have public visibility.

Context: Stereotype *CIM_Instance*

OCL:
```
self. base_InstanceSpecification.visibility =
     uml::VisibilityKind::public
```

C21. A CIM instance shall not have UML specifications.

Context: Stereotype *CIM_Instance*

OCL: `self. base_InstanceSpecification.specification->isEmpty()`

C22. A CIM class shall not import any elements.

Context: A stereotype extending UML *Class*

OCL: `self.base_Class.elementImport->isEmpty()`

C23. A CIM class shall not import any packages.

Context: A stereotype extending UML *Class*

OCL: `self.base_Class.packageImport->isEmpty()`

5.17.3 OCL Constraints on Properties and References

P1. A CIM non-reference property (i.e., not an association end) shall have the *CIM_Property* stereotype applied.

Context: A stereotype extending UML *Property*

OCL:
```
self.base_Property.association->isEmpty()
  implies
  self.base_Property.getAppliedStereotype('CIM::CIM_Property')
     <> null
```

P2. A CIM reference (i.e., an association end) shall have the *CIM_Reference* stereotype applied.

Context: A stereotype extending UML *Property*

OCL:
```
self.base_Property.association->isEmpty() = false
implies
self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
  <> null
```

P3. A CIM property shall have only one of the stereotypes *CIM_Property* and *CIM_Reference* applied.

Context: A stereotype extending UML *Property*

OCL:
```
Bag {
  self.base_Property.getAppliedStereotype('CIM::CIM_Property'),
  self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
}->select( s | s <> null )->size() <> 2
```

Note: Other constraints validate that normal properties have CIM_Property applied and that association ends have CIM_Reference applied.

P4. The end of a CIM association shall be non-navigable.

Context: Stereotype *CIM_Reference*

OCL:
```
self.base_Property.class.getAppliedStereotype(
'CIM::CIM_Association') <> null
```

P5. The type of a CIM non-reference property shall be a CIM datatype, an Enumeration or an embedded instance.

Context: Meta Element Stereotype CIM_Property

OCL:
```
self.base_Property.type.allNamespaces()->at(1).name =
'CIMDatatypes' or
self.base_Property.type.oclIsTypeof(uml::Enumeration) or
self.base_Property.type.oclIsTypeOf(uml::Class)
```

P6. A CIM non-reference property can override only a CIM non-reference property.

Context: Stereotype *CIM_Property*

OCL:
```
self.base_Property.redefinedProperty->isEmpty() = false
implies (
  self.base_Property.redefinedProperty->size() = 1
  and (
    let rp : Property = self.base_Property.redefinedProperty->
      asSequence()->at(1)
    in
    rp.getAppliedStereotype('CIM::CIM_Property') <> null
  )
)
```

P7. The name of an overridden CIM non-reference property shall not change in the subclass.

NOTE: As with all CIM names, this is tested for in a case-insensitive way.

Context: Stereotype *CIM_Property*

OCL:
```
self.base_Property.redefinedProperty->isEmpty() = false
implies (
   self.base_Property.redefinedProperty->size() = 1
   and (
      let rp : Property = self.base_Property.redefinedProperty->
         asSequence()->at(1)
      in
      self.base_Property.name.toUpper() = rp.name.toUpper()
   )
)
```

P8. A CIM reference can override only a CIM reference.

Context: Stereotype *CIM_Reference*

OCL:
```
self.base_Property.redefinedProperty->isEmpty() = false
implies (
   self.base_Property.redefinedProperty->size() = 1
   and (
      let rp : Property = self.base_Property.redefinedProperty->
         asSequence()->at(1)
      in
      rp.getAppliedStereotype('CIM::CIM_Reference') <> null
   )
)
```

P9. The name of an overridden CIM reference shall not change in the subclass.

NOTE: As with all CIM names, this is tested for in a case-insensitive way.

Context: Stereotype *CIM_Reference*

OCL:
```
self.base_Property.redefinedProperty->isEmpty() = false
implies (
   self.base_Property.redefinedProperty->size() = 1
   and (
      let rp : Property = self.base_Property.redefinedProperty->
         asSequence()->at(1)
      in
      self.base_Property.name.toUpper() = rp.name.toUpper()
   )
)
```

P10. The type of a CIM reference in a subclass shall be the class referenced in the superclass or one of its subclasses.

NOTE 1: As with all CIM names, equality test for names is done in a case-insensitive way.

NOTE 2: The association ends in the derived UML AssociationClass exist regardless of whether or not an overridden CIM reference in the derived CIM association was defined.

Context: Stereotype *CIM_Reference*

OCL:
```
self.base_Property.class.superClass->isEmpty() = false
implies (
   let superp : Property =
      self.base_Property.class.superClass.getAllAttributes()->
      select( p | p.name.toUpper()
         = self.base_Property.name.toUpper())->
      asSequence()->at(1)
   in
   self.base_Property.type.conformsTo(superp.type)
)
```

P11. A CIM reference shall not reference a CIM indication.

Context: Stereotype *CIM_Reference*

OCL: `self.base_Property.type.getAppliedStereotype('CIM::CIM_Indication') = null`

P12. A CIM property shall have public visibility.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.visibility = uml::VisibilityKind::public`

P13. A CIM property shall have the UML qualifier *isDerived* set to false.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.isDerived = false`

P14. A CIM property shall have the UML qualifier *isDerivedUnion* set to false.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.isDerivedUnion = false`

P15. A CIM reference shall not be a leaf.

Context: Stereotype *CIM_Reference*

OCL: `self.base_Property.isLeaf = false`

P16. A CIM non-reference property shall not be unique.

Context: Stereotype *CIM_Property*

OCL: `self.base_Property.isUnique = false`

P17. A CIM reference shall be unique unless there are non-reference key properties defined in the same class.

Context: Stereotype *CIM_Reference*

OCL:
```
if self.base_Property.class.getAllAttributes()->select( p |
     p.getAppliedStereotype('CIM::Key') <> null
     and
     p.getAppliedStereotype('CIM::CIM_Property') <> null
     )->size() = 0
  then
     self.base_Property.isUnique = false
  else
     self.base_Property.isUnique = true
  endif
```

P18. A CIM property shall not define subsetted properties.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.subsettedProperty->isEmpty()`

P19. A CIM property shall not define UML qualifiers.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.qualifier->isEmpty()`

P20. A CIM reference shall not be static.

Context: Stereotype *CIM_Reference*

OCL: `self.base_Property.isStatic = false`

P21. A CIM reference shall not be ordered.

Context: Stereotype *CIM_Reference*

OCL: `self.base_Property.isOrdered = false`

P22. A CIM key property shall not be an array.

Context: Stereotype *CIM_Property*

OCL:
```
self.base_Property.getAppliedStereotype('CIM::Key') <> null
  implies
  self.base_Property.upper = 1
```

P23. The name of a CIM property shall conform to the format for CIM element names defined in DSP0004.

The format for CIM element names is described in 5.17.1.8.

Context: A stereotype extending UML *Property*

OCL: `self.base_Property.name.isValidCimElementName()`

5.17.4 OCL Constraints on Methods and Parameters

M1. The name of an overridden CIM method shall not change in the subclass.

NOTE: As with all CIM names, this is tested for in a case-insensitive way.

Context: Stereotype *CIM_Method*

OCL:
```
self.base_Operation.redefinedOperation->isEmpty() = false
    implies (
      self.base_Operation.redefinedOperation->size() = 1
      and (
        let ro : Operation = self.base_Operation.redefinedOperation->
          asSequence()->at(1)  /* the overridden method */
        in
        self.base_Operation.name.toUpper() = ro.name.toUpper()
              /* method name */
      )
    )
```

M2. The signature of an overridden CIM method shall not change in the subclass.

NOTE 1: The return type is part of the signature.

NOTE 2: As with all CIM names, the equality test for method names is done in a case-insensitive way.

Context: Stereotype *CIM_Method*

OCL:
```
self.base_Operation.redefinedOperation->isEmpty() = false
    implies (
      self.base_Operation.redefinedOperation->size() = 1
      and (
        let ro : Operation = self.base_Operation.redefinedOperation->
          asSequence()->at(1)  /* the overridden method */
        in
        self.base_Operation.type = ro.type  /* method return type */
        and
        self.base_Operation.ownedParameter->size() =
          ro.ownedParameter->size()  /* number of parameters */
        and
        Set {1 .. ro.ownedParameter->size() }->forAll( i |
          self.base_Operation.ownedParameter->asOrderedSet()->
            at(i).name.toUpper() = ro.ownedParameter->asOrderedSet()->
            at(i).name.toUpper()  /* parameter name */
          and
          self.base_Operation.ownedParameter->asOrderedSet()->
            at(i).type = ro.ownedParameter->asOrderedSet()->
            at(i).type  /* parameter type */
        )
      )
    )
```

M3. A CIM method shall not be a query.

Context: Stereotype *CIM_Method*

OCL: `self. base_Operation.isQuery = false`

M4. A CIM method shall have public visibility.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.visibility = uml::VisibilityKind::public`

M5. A CIM method shall not be a leaf.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.isLeaf = false`

M6. A CIM method shall not use body constraints.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.bodyCondition->isEmpty()`

M7. A CIM method shall not define exceptions.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.raisedException->isEmpty()`

M8. A CIM method shall not import any elements.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.elementImport->isEmpty()`

M9. A CIM method shall not import any packages.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.packageImport->isEmpty()`

M10. An in/out/inout parameter of a CIM method shall have the *CIM_Parameter* stereotype applied.

Context: A stereotype extending UML *Parameter*

OCL:
```
self.base_Parameter.direction <>uml::ParameterDirectionKind::return
    implies
    self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter') <> null
```

M11. A CIM method return value shall not have the *CIM_Parameter* stereotype applied.

Context: A stereotype extending UML *Parameter*

OCL:
```
self.base_Parameter.direction = uml::ParameterDirectionKind::return
    implies
    self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter') = null
```

M12. The type of a CIM parameter shall be a CIM datatype, an Enumeration or an embedded instance.

Context: Stereotype *CIM_Parameter*

OCL:
```
self.base_Parameter.type.allNamespaces()->at(1).name = 'CIMDatatypes' or
    self.base_Parameter.type.oclIsTypeOf(uml::Enumeration) or
    self.base_Parameter.type.oclIsTypeOf(uml::Class)
```

M13. The type of a CIM method return value shall be a CIM datatype, an Enumeration or an embedded instance.

NOTE: Returning CIM references is not allowed in CIM.

Context: A stereotype extending UML *Parameter*

OCL:
```
self.base_Parameter.direction = uml::ParameterDirectionKind::return
    implies
    self.base_Parameter.type.allNamespaces()->at(1).name = 'CIMDatatypes' or
    self.base_Parameter.type.oclIsTypeOf(uml::Enumeration) or
    self.base_Parameter.type.oclIsTypeOf(uml::Class)
```

M14. A CIM method return value shall not be an array.

Context: A stereotype extending UML *Parameter*

OCL:
```
self.base_Parameter.direction = uml::ParameterDirectionKind::return
    implies
    self.base_Parameter.upper = 1
```

M15. A CIM parameter shall not have a default value.

Context: Stereotype *CIM_Parameter*

OCL: `self.base_Parameter.defaultValue->isEmpty()`

M16. A CIM parameter shall have public visibility.

Context: Stereotype *CIM_Parameter*

OCL: `self.base_Parameter.visibility = uml::VisibilityKind::public`

M17. A CIM parameter shall not be unique.

Context: Stereotype *CIM_Parameter*

OCL: `self.base_Parameter.isUnique = false`

M18. The name of a CIM method shall conform to the format for CIM element names defined in DSP0004.

The format for CIM element names is described in 5.17.1.8.

Context: Stereotype *CIM_Method*

OCL: `self.base_Operation.name.isValidCimElementName()`

M19. The name of a CIM parameter shall conform to the format for CIM element names defined in DSP0004.

The format for CIM element names is described in 5.17.1.8.

Context: Stereotype *CIM_Parameter*

OCL: `self.base_Parameter.name.isValidCimElementName()`

5.17.5 OCL Constraints on Qualifiers

Q1. A CIM qualifier shall have a CIM datatype.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.type.allNamespaces()->at(1).name = 'CIMDatatypes'`

Q2. A CIM qualifier type stereotype shall have public visibility.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.visibility = uml::VisibilityKind::public`

Q3. A CIM qualifier type stereotype shall be abstract.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.isAbstract`

Q4. A CIM qualifier type stereotype shall not be a leaf.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.isLeaf = false`

Q5. A CIM qualifier type stereotype shall not own UML operations.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.ownedOperation->isEmpty()`

Q6. A CIM qualifier type stereotype shall not have a superclass.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.superClass->isEmpty()`

Q7. A CIM qualifier type stereotype shall not import any elements.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.elementImport->isEmpty()`

Q8. A CIM qualifier type stereotype shall not import any packages.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.packageImport->isEmpty()`

Q9. A CIM qualifier type stereotype shall not have a comment.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.ownedComment->isEmpty()`

Q10. A CIM qualifier type stereotype shall have one property defined.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.class.ownedAttribute->size() = 1`

Q11. A CIM qualifier type stereotype property shall not be an association end.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.owningAssociation->isEmpty()`

Q12. A CIM qualifier type stereotype property shall have public visibility.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.visibility = uml::VisibilityKind::public`

Q13. A CIM qualifier type stereotype property shall have the UML qualifier *isDerived* set to false.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isDerived = false`

Q14. A CIM qualifier type stereotype property shall have the UML qualifier *isDerivedUnion* set to false.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isDerivedUnion = false`

Q15. A CIM qualifier type stereotype property shall not be read-only.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isReadOnly = false`

Q16. A CIM qualifier type stereotype property shall not be static.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isStatic = false`

Q17. A CIM qualifier type stereotype property shall not be a leaf.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isLeaf = false`

Q18. A CIM qualifier type stereotype property shall be unique.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.isUnique`

Q19. A CIM qualifier type stereotype property of array type shall be ordered.

Context: Stereotype *CIM_QualifierType*

OCL:
```
self.base_Property.upper > 1
    implies
    self.base_Property.isOrdered = true
```

Q20. A CIM qualifier type stereotype property shall not be owned by a structured datatype.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.datatype->isEmpty()`

Q21. A CIM qualifier type stereotype property shall not define subsetted properties.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.subsettedProperty->isEmpty()`

Q22. A CIM qualifier type stereotype property shall not redefine another property.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.redefinedElement->isEmpty()`

Q23. A CIM qualifier type stereotype property shall not define UML qualifiers.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.qualifier->isEmpty()`

Q24. The number of entries in the *BitValues* and *BitMap* qualifier arrays shall match.

Context: Stereotype *CIM_Property*

OCL: `self.BitMap->size() = self.BitValues->size()`

Context: Stereotype *CIM_Method*

OCL: `self.BitMap->size() = self.BitValues->size()`

Context: Stereotype *CIM_Parameter*

OCL: self.BitMap->size() = self.BitValues->size()

Q25. The *BitMap* qualifier is applicable only to a CIM integer datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'BitMap')
implies
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'BitMap')
implies
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'BitMap')
implies
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q26. The *BitValues* qualifier is applicable only to a CIM integer datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'BitValues')
implies
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'BitValues')
implies
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
    self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'BitValues')
implies
Set { /* the integer datatypes */
    'CIMDatatypes::uint8',
    'CIMDatatypes::uint16',
    'CIMDatatypes::uint32',
    'CIMDatatypes::uint64',
    'CIMDatatypes::sint8',
    'CIMDatatypes::sint16',
    'CIMDatatypes::sint32',
    'CIMDatatypes::sint64'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q27. The *Counter* qualifier is applicable only to a CIM unsigned integer datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
    self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Counter')
implies
Set { /* the unsigned integer datatypes */
    'CIMDatatypes::uint8',
    'CIMDatatypes::uint16',
    'CIMDatatypes::uint32',
    'CIMDatatypes::uint64'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
    self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'Counter')
implies
Set { /* the unsigned integer datatypes */
    'CIMDatatypes::uint8',
    'CIMDatatypes::uint16',
    'CIMDatatypes::uint32',
    'CIMDatatypes::uint64'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'Counter')
implies
Set { /* the unsigned integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q28. A value of the *Deprecated* qualifier shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format for non-NULL values of the *Deprecated* qualifier as defined in the *CIM Infrastructure Specification* and DSP0004 is:

```
DeprecatedFormat = className [ [ embeddedInstancePath ]
        "." elementSpec ];
elementSpec = propertyName |
        methodName "(" [parameterName *("," parameterName)] ")";
embeddedInstancePath = 1*( "." propertyName );
```

where the non-terminals on the right hand side of these ABNF rules are defined in Appendix A of DSP0004.

Context: Stereotype *CIM_Class*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Association*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Indication*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Property*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Reference*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Method*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Context: Stereotype *CIM_Parameter*

OCL: `self.Deprecated.matchCimAbnf('DeprecatedFormat')`

Q29. If stereotype property *Deprecated* has a value, the marker stereotype *Deprecated* shall be applied.

Context: Stereotype *CIM_Class* and *CIM_Indication*

OCL:
```
self.Deprecated->size() > 0
    implies
    self.base_Class.getAppliedStereotype('CIM::Deprecated') <> null
```

Context: Stereotype *CIM_Association*

OCL:
```
self.Deprecated->size() > 0
    implies
    self.base_AssociationClass.getAppliedStereotype('CIM::Deprecated') <> null
```

Context: Stereotypes *CIM_Property* and *CIM_Reference*

OCL:
```
self.Deprecated->size() > 0
    implies
    self.base_Property.getAppliedStereotype('CIM::Deprecated') <> null
```

Context: Stereotype *CIM_Method*

OCL:
```
self.Deprecated->size() > 0
    implies
    self.base_Operation.getAppliedStereotype('CIM::Deprecated') <> null
```

Context: Stereotype *CIM_Parameter*

OCL:
```
self.Deprecated->size() > 0
    implies
    self.base_Parameter.getAppliedStereotype('CIM::Deprecated') <> null
```

Q30. If the *Deprecated* qualifier is specified on the class owning a property, it should also be specified on that property.

Context: Stereotypes *CIM_Property* and *CIM_Reference*

OCL:
```
self.base_Property.class.getAppliedStereotype('CIM::Deprecated') <> null
    implies
    self.base_Property.getAppliedStereotype('CIM::Deprecated') <> null
```

Q31. If the *Deprecated* qualifier is specified on the class owning a method, it should also be specified on that method.

Context: Stereotype *CIM_Method*

OCL:
```
self.base_Operation.class.getAppliedStereotype('CIM::Deprecated') <> null
    implies
    self.base_Operation.getAppliedStereotype('CIM::Deprecated') <> null
```

Q32. The *DN* qualifier is applicable only to a CIM string datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
       self.base_Property.getAppliedStereotype('CIM::CIM_Property')
    in
    self.base_Property.isSet(s,'DN')
    implies
    Set { /* the string datatypes */
```

```
            'CIMDatatypes::string'
        }->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
            self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
        in
        self.base_Operation.isSet(s,'DN')
        implies
        Set { /* the string datatypes */
            'CIMDatatypes::string'
        }->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
            self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
        in
        self.base_Parameter.isSet(s,'DN')
        implies
        Set { /* the string datatypes */
            'CIMDatatypes::string'
        }->includes(self.base_Parameter.type.qualifiedName)
```

Q33. The *EmbeddedObject* qualifier is applicable only to a CIM string datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
            self.base_Property.getAppliedStereotype('CIM::CIM_Property')
        in
        self.base_Property.isSet(s,'EmbeddedObject')
        implies
        Set { /* the string datatypes */
            'CIMDatatypes::string'
        }->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
            self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
        in
        self.base_Operation.isSet(s,'EmbeddedObject')
        implies
        Set { /* the string datatypes */
            'CIMDatatypes::string'
        }->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
            self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
        in
        self.base_Parameter.isSet(s,'EmbeddedObject')
```

```
    implies
    Set { /* the string datatypes */
       'CIMDatatypes::string'
    }->includes(self.base_Parameter.type.qualifiedName)
```

Q34. A CIM class shall not be abstract and have an *Exception* qualifier with an effective value of true.

Context: Stereotypes *CIM_Class* and *CIM_Indication*

OCL: `not (self.base_Class.isAbstract and self.Exception)`

Q35. The *Gauge* qualifier is applicable only to a CIM unsigned integer datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Gauge')
implies
Set { /* the unsigned integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'Gauge')
implies
Set { /* the unsigned integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'Gauge')
implies
Set { /* the unsigned integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q36. A value of an entry in the *MappingStrings* qualifier array shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format for non-NULL values of the *MappingStrings* qualifier as defined in DSP0004 is:

```
MappingStringsFormat = mib_format | oid_format | general_format | mif_format;
mib_format = "MIB" "." mib_naming_authority "|" mib_name "." mib_variable_name;
mib_naming_authority = 1*(stringChar);
mib_name = 1*(stringChar);
mib_variable_name = 1*(stringChar);
oid_format = "OID" "." oid_naming_authority "|" oid_protocol_name "." oid;
oid_naming_authority = 1*(stringChar);
oid_protocol_name = 1*(stringChar);
oid = 1*(stringChar);
general_format = general_format_fullname "|" general_format_mapping;
general_format_fullname = general_format_name "." general_format_defining_body;
general_format_name = 1*(stringChar);
general_format_defining_body = 1*(stringChar);
general_format_mapping = 1*(stringChar);
mif_format = mif_attribute_format | mif_group_format;
mif_attribute_format = "MIF" "." mif_class_string "." mif_attribute_id;
mif_group_format = "MIF" "." mif_class_string;
mif_class_string = mif_defining_body "|" mif_specific_name "|" mif_version;
mif_defining_body = 1*(stringChar);
mif_specific_name = 1*(stringChar);
mif_version = 3(decimalDigit);
mif_attribute_id = positiveDecimalDigit *decimalDigit;
```

where the non-terminals on the right hand side of these ABNF rules are defined in Appendix A of DSP0004.

Context: Stereotypes *CIM_Class, CIM_Association, CIM_Indication, CIM_Property, CIM_Reference, CIM_Method, CIM_Parameter*

OCL:
```
self.MappingStrings->forAll( qv | qv.matchCimAbnf( 'MappingStringsFormat' )
)
```

Q37. The *MaxLen* qualifier is applicable only to a CIM string datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'MaxLen')
implies
Set { /* the string datatypes */
     'CIMDatatypes::string'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'MaxLen')
implies
Set { /* the string datatypes */
   'CIMDatatypes::string'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'MaxLen')
implies
Set { /* the string datatypes */
   'CIMDatatypes::string'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q38. The *MinLen* qualifier is applicable only to a CIM string datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'MinLen')
implies
Set { /* the string datatypes */
   'CIMDatatypes::string'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'MinLen')
implies
Set { /* the string datatypes */
   'CIMDatatypes::string'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'MinLen')
implies
Set { /* the string datatypes */
```

```
            'CIMDatatypes::string'
        }->includes(self.base_Parameter.type.qualifiedName)
```

Q39. The *MaxValue* qualifier is applicable only to a CIM numeric datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
            self.base_Property.getAppliedStereotype('CIM::CIM_Property')
        in
        self.base_Property.isSet(s,'MaxValue')
        implies
        Set { /* the numeric datatypes */
            'CIMDatatypes::uint8',
            'CIMDatatypes::uint16',
            'CIMDatatypes::uint32',
            'CIMDatatypes::uint64',
            'CIMDatatypes::sint8',
            'CIMDatatypes::sint16',
            'CIMDatatypes::sint32',
            'CIMDatatypes::sint64',
            'CIMDatatypes::real32',
            'CIMDatatypes::real64'
        }->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
            self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
        in
        self.base_Operation.isSet(s,'MaxValue')
        implies
        Set { /* the numeric datatypes */
            'CIMDatatypes::uint8',
            'CIMDatatypes::uint16',
            'CIMDatatypes::uint32',
            'CIMDatatypes::uint64',
            'CIMDatatypes::sint8',
            'CIMDatatypes::sint16',
            'CIMDatatypes::sint32',
            'CIMDatatypes::sint64',
            'CIMDatatypes::real32',
            'CIMDatatypes::real64'
        }->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
            self.base_Parameter. getAppliedStereotype('CIM::CIM_Parameter')
        in
        self.base_Parameter.isSet(s,'MaxValue')
        implies
        Set { /* the numeric datatypes */
```

```
            'CIMDatatypes::uint8',
            'CIMDatatypes::uint16',
            'CIMDatatypes::uint32',
            'CIMDatatypes::uint64',
            'CIMDatatypes::sint8',
            'CIMDatatypes::sint16',
            'CIMDatatypes::sint32',
            'CIMDatatypes::sint64',
            'CIMDatatypes::real32',
            'CIMDatatypes::real64'
         }->includes(self.base_Parameter.type.qualifiedName)
```

Q40. The *MinValue* qualifier is applicable only to a CIM numeric datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'MinValue')
implies
Set { /* the numeric datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::real32',
   'CIMDatatypes::real64'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'MinValue')
implies
Set { /* the numeric datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::real32',
   'CIMDatatypes::real64'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'MinValue')
implies
Set { /* the numeric datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::real32',
   'CIMDatatypes::real64'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q41. A value of the *ModelCorrespondence* qualifier shall conform to the format defined in DSP0004.

The format defined in DSP0004 is:

```
className [ *( "." ( propertyName |  referenceName ) ) [ "." methodName [ "("
      parameterName ")"] ] ]
```

Note that this format definition is currently flawed for multiple reasons:

- It allows only one parameter for a method.
- It does not describe in DSP0004 that the concatenation of multiple properties is meant to denote elements in embedded properties.
- It allows further elements to be specified after a reference as if the reference were an embedded property, which does not make sense.
- It does not allow distinguishing between methods and properties when they have the same name (a rare case, admittedly).

Therefore, this document is using the following improved format definition, which is intended to be incorporated into DSP0004 at some point. Represented in ABNF according to 3.4, the format is:

```
ModelCorrespondenceFormat = className [ [ embeddedPropertyNames ]
  "." ( propertyName | referenceName |
        methodName "(" [ parameterName *("," parameterName) ] ")" )
  ];
embeddedPropertyNames = 1*( "." propertyName );
```

where the remaining rules on the right hand side of these ABNF rules are defined in DSP0004, Appendix A.

Context: Stereotypes *CIM_Class*, *CIM_Association*, *CIM_Indication*, *CIM_Property*, *CIM_Reference*, *CIM_Method* and *CIM_Parameter*

OCL:
```
self.ModelCorrespondence->forAll( qv |
     qv.matchCimAbnf( 'ModelCorrespondenceFormat' )
```

Q42. The CIM elements referenced in a *ModelCorrespondence* qualifier value shall exist.

NOTE: As with all CIM names, equality tests are done in a case-insensitive way.

Context: Stereotypes *CIM_Class, CIM_Association, CIM_Indication, CIM_Property, CIM_Reference, CIM_Method, CIM_Parameter*

OCL:
```
let className : String = self.ModelCorrespondence.extractCimAbnf(
   'ModelCorrespondenceFormat', 'className' )
in
let propName : String = self.ModelCorrespondence.extractCimAbnf(
   'ModelCorrespondenceFormat', 'propertyName' )
in
let refName : String = self.ModelCorrespondence.extractCimAbnf(
   'ModelCorrespondenceFormat', 'referenceName' )
in
let methName : String = self.ModelCorrespondence.extractCimAbnf(
   'ModelCorrespondenceFormat', 'methodName' )
in
let parmNames : OrderedSet(String) =
   self.ModelCorrespondence.extractCimAbnf('ModelCorrespondenceFormat',
      'parameterName' )
in
let embPropNames : OrderedSet(String) =
   self.ModelCorrespondence.extractCimAbnf('ModelCorrespondenceFormat',
      'embeddedPropertyNames').extractCimAbnf( 'embeddedPropertyNames',
      'propertyName')
in
let c : Class = Class.allInstances()->select( _c | _c.name.toUpper() =
   className.toUpper())
in
   c <> null
and
   propName <> null implies
      c.getAllAttributes()->select( p | p.name.toUpper() =
         propName.toUpper()) <> null)
and
   refName <> null implies
      c.getAllAttributes()->select( p | p.name.toUpper() =
         refName.toUpper()) <> null)
and
   methName <> null implies
      c.getAllOperations()->select( m | m.name.toUpper() =
         methName.toUpper())->select( m | m.ownedParameter.name
         ->asOrderedSet().toUpper() = parmNames.toUpper())
   )
)
```

Q43. The *NullValue* qualifier is applicable only to CIM string and integer datatypes.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'NullValue')
implies
```

```
      Set { /* the string and integer datatypes */
         'CIMDatatypes::uint8',
         'CIMDatatypes::uint16',
         'CIMDatatypes::uint32',
         'CIMDatatypes::uint64',
         'CIMDatatypes::sint8',
         'CIMDatatypes::sint16',
         'CIMDatatypes::sint32',
         'CIMDatatypes::sint64',
         'CIMDatatypes::string',
         'CIMDatatypes::char16'
      }->includes(self.base_Property.type.qualifiedName)
```

Q44. A value of the *NullValue* qualifier when used with an integer datatype shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 for this case is:

```
NullValueFormatForIntegers = [ "+" | "-" ] 1*(decimalDigit);
```

where the rule on the right hand side of this ABNF rule is defined in Appendix A of DSP0004, and the value range of this format is constrained by the integer datatype of the CIM element to which this qualifier is applied. For example, the value range for this qualifier when applied to a CIM element with a `uint8` datatype would be 0..127.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
         self.base_Property.getAppliedStereotype('CIM::CIM_Property')
      in
      self.base_Property.isSet(s,'NullValue')
      and
      Set { /* the integer datatypes */
         'CIMDatatypes::uint8',
         'CIMDatatypes::uint16',
         'CIMDatatypes::uint32',
         'CIMDatatypes::uint64',
         'CIMDatatypes::sint8',
         'CIMDatatypes::sint16',
         'CIMDatatypes::sint32',
         'CIMDatatypes::sint64'
      }->includes(self.base_Property.type.qualifiedName)
      implies
      self.NullValue.matchCimAbnf( 'NullValueFormatForIntegers' )
```

Q45. A value of the *Propagated* qualifier shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
PropagatedFormat = [className "."] IDENTIFIER;
```

where the rules on the right hand side of this ABNF rule are defined in Appendix A of DSP0004.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
         self.base_Property.getAppliedStereotype('CIM::CIM_Property')
      in
      self.base_Property.isSet(s,'Propagated')
```

```
implies
self.Propagated.matchCimAbnf( 'PropagatedFormat' )
```

Q46. The property referenced in a Propagated qualifier value shall be exposed by the specified class.

NOTE: As with all CIM names, equality tests are done in a case-insensitive way.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Propagated')
implies
let className : String = self.Propagated.extractCimAbnf(
   'PropagatedFormat', 'className' )
in
let propName : String = self.Propagated.extractCimAbnf(
   'PropagatedFormat', 'IDENTIFIER' )
in
let myc : Class = /* class receiving the propagated property */
   self.oclAsType(CIM::CIM_Property).base_Property.class
in
let c : Class = /* class that owns the propagated property */
   if className = null
   then
      myc /* default if no class was specified */
   else
      Class.allInstances()->select( c | c.name.toUpper() =
         className.toUpper())
         ->asOrderedSet()->at(1)
   endif
in
let p : Property = /* the propagated property */
   c.getAllAttributes()->select( p | p.name.toUpper() = propName.toUpper())
in
   c <> null and p <> null
```

Q47. A property qualified with *Propagated* shall be a key property.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Propagated')
implies
self.base_Property.getAppliedStereotype('CIM::Key') <> null
```

Q48. The property referenced in a *Propagated* qualifier value shall have the same datatype as the qualified property.

NOTE: As with all CIM names, equality tests are done in a case-insensitive way.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
```

```
        in
        self.base_Property.isSet(s,'Propagated')
        implies
        let className : String = self.Propagated.extractCimAbnf(
           'PropagatedFormat', 'className' )
        in
        let propName : String = self.Propagated.extractCimAbnf(
           'PropagatedFormat', 'IDENTIFIER' )
        in
        let myc : Class = /* class receiving the propagated property */
           self.oclAsType(CIM::CIM_Property).base_Property.class
        in
        let c : Class = /* class that owns the propagated property */
           if className = null
           then
              myc /* default if no class was specified */
           else
              Class.allInstances()->select( c | c.name.toUpper() =
                 className.toUpper())
                 ->asOrderedSet()->at(1)
           endif
        in
        let p : Property = /* the propagated property */
           c.getAllAttributes()->select( p | p.name.toUpper() = propName.toUpper())
        in
           p.type = self.oclAsType(CIM::CIM_Property).base_Property.type
```

Q49. The class of a property qualified with the *Propagated* qualifier and the class of the property referenced in that *Propagated* qualifier value shall have an association defined between them which is qualified as weak on the reference referencing the property qualified with the *Propagated* qualifier.

NOTE: As with all CIM names, equality tests are done in a case-insensitive way.

Context: Stereotype *CIM_Property*

OCL:
```
        let s : Stereotype =
           self.base_Property.getAppliedStereotype('CIM::CIM_Property')
        in
        self.base_Property.isSet(s,'Propagated')
        implies
        let className : String = self.Propagated.extractCimAbnf(
           'PropagatedFormat', 'className' )
        in
        let propName : String = self.Propagated.extractCimAbnf(
           'PropagatedFormat', 'IDENTIFIER' )
        in
        let myc : Class = /* class receiving the propagated property */
           self.oclAsType(CIM::CIM_Property).base_Property.class
        in
        let c : Class = /* class that owns the propagated property */
           if className = null
```

```
              then
                 myc /* default if no class was specified */
              else
                 Class.allInstances()->select( c | c.name.toUpper() =
                    className.toUpper())
                    ->asOrderedSet()->at(1)
              endif
           in
           let acs : Set(AssociationClass) = /* associations between them */
              AssociationClass.allInstances()->select( ac |
                 ac.ownedEnd.class = Set {c, myc} )
           in
           acs->select( ac |
              ac.ownedEnd.getAppliedStereotype('CIM::CIM_Reference').
                 oclAsType(CIM::CIM_Reference).Weak = true)->size() = 1
```

Q50. A CIM class shall not be abstract and have a *Terminal* qualifier with an effective value of true.

Context: Stereotypes *CIM_Class* and *CIM_Indication*

OCL: `not (self.base_Class.isAbstract and self.Terminal)`

Context: Stereotypes *CIM_Association*

OCL: `not (self.base_AssociationClass.isAbstract and self.Terminal)`

Q51. The number of entries in the *Values* and *ValueMap* qualifier arrays shall match if both are defined.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'ValueMap')
and self.base_Property.isSet(s,'Values')
implies
self.ValueMap->size() = self.Values->size()
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Property.isSet(s,'ValueMap')
and self.base_Property.isSet(s,'Values')
implies
self.ValueMap->size() = self.Values->size()
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Property.isSet(s,'ValueMap')
and self.base_Property.isSet(s,'Values')
implies
```

```
        self.ValueMap->size() = self.Values->size()
```

Q52. The *ValueMap* qualifier is applicable only to a CIM string or integer datatype.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'ValueMap')
implies
Set { /* the integer and string datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::string',
   'CIMDatatypes::char16'
}->includes(self.base_Property.type.qualifiedName)
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'ValueMap')
implies
Set { /* the integer and string datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::string',
   'CIMDatatypes::char16'
}->includes(self.base_Operation.type.qualifiedName)
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'ValueMap')
implies
Set { /* the integer and string datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64',
   'CIMDatatypes::string',
   'CIMDatatypes::char16'
}->includes(self.base_Parameter.type.qualifiedName)
```

Q53. A value of an entry in the *ValueMap* qualifier array when used with an integer datatype shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
ValueMapFormat = [integerValue] ".." [integerValue];
```

where the rule on the right hand side of this ABNF rule is defined in Appendix A of DSP0004, and the value range of both occurrences of `integerValue` is constrained by the integer datatype of the CIM element to which this qualifier is applied.

Context: Stereotypes *CIM_Property*, *CIM_Method* and *CIM_Parameter*

OCL:
```
self.ValueMap->forAll( qv | qv.matchCimAbnf( 'ValueMapFormat' )
)
```

Q54. The value of a *Version* qualifier array shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
VersionFormat = 1*(decimalDigit) "." 1*(decimalDigit)
                "." 1*(decimalDigit);
```

where the rule on the right hand side of this ABNF rule is defined in Appendix A of DSP0004.

Context: Stereotypes *CIM_Class*, *CIM_Association* and *CIM_Indication*,

OCL:
```
self.Version.matchCimAbnf( 'VersionFormat' )
```

Q55. A CIM association shall have no more than one reference qualified with the *Weak* qualifier.

Context: Stereotype *CIM_Association*

OCL:
```
self.base_AssociationClass.ownedEnd->select( p |
    p.isSet(p.getAppliedStereotype('CIM::CIM_Reference'),'Weak')
)->size() <= 1
```

Q56. The value of a *PropertyUsage* qualifier shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
PropertyUsageFormat = "UNKNOWN" | "OTHER" | "CURRENTCONTEXT" |
                      "DESCRIPTIVE" | "CAPABILITY" |
                      "CONFIGURATION" | "STATE" | "METRIC";
```

Context: Stereotype *CIM_Property*

OCL:
```
self.PropertyUsage <> null
implies
Bag {
    'UNKNOWN', 'OTHER', 'CURRENTCONTEXT',
    'DESCRIPTIVE', 'CAPABILITY',
    'CONFIGURATION', 'STATE', 'METRIC'
```

```
        }->includes(self.PropertyUsage.toUpper())
```

Q57. The *Syntax* qualifier and the *SyntaxType* qualifier shall be used together.

Context: Stereotypes *CIM_Property*, *CIM_Reference*, *CIM_Method* and *CIM_Parameter*

OCL: `self.Syntax <> null xor self.SyntaxType = null`

Q58. The value of a *TriggerType* qualifier on a CIM non-indication class shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
TriggerTypeFormat1 = "CREATE" | "DELETE" | "UPDATE" | "ACCESS";
```

Context: Stereotype *CIM_Class*

OCL:
```
self.TriggerType <> null
    implies
    Bag {
       'CREATE', 'DELETE', 'UPDATE', 'ACCESS'
    }->includes(self.TriggerType.toUpper())
```

Context: Stereotype *CIM_Association*

OCL:
```
self.TriggerType <> null
    implies
    Bag {
       'CREATE', 'DELETE', 'UPDATE', 'ACCESS'
    }->includes(self.TriggerType.toUpper())
```

Q59. The value of a *TriggerType* qualifier on a CIM property shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
TriggerTypeFormat2 = "UPDATE" | "ACCESS";
```

Context: Stereotype *CIM_Property*

OCL:
```
self.TriggerType <> null
    implies
    Bag {
       'UPDATE', 'ACCESS'
    }->includes(self.TriggerType.toUpper())
```

Context: Stereotype *CIM_Reference*

OCL:
```
self.TriggerType <> null
    implies
    Bag {
       'UPDATE', 'ACCESS'
    }->includes(self.TriggerType.toUpper())
```

Q60. The value of a *TriggerType* qualifier on a CIM method shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
TriggerTypeFormat2 = "BEFORE" | "AFTER";
```

Context: Stereotype *CIM_Method*

OCL:
```
self.TriggerType <> null
implies
Bag {
   'BEFORE', 'AFTER'
}->includes(self.TriggerType.toUpper())
```

Q61. The value of a *TriggerType* qualifier on a CIM indication shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
TriggerTypeFormat3 = "THROWN";
```

Context: Stereotype *CIM_Indication*

OCL:
```
self.TriggerType <> null
implies
Bag {
   'THROWN'
}->includes(self.TriggerType.toUpper())
```

Q62. A value of an entry in the *UnsupportedValues* qualifier array when used with an integer datatype shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
UnsupportedValuesFormat = [integerValue] ".." [integerValue];
```

where the rule on the right hand side of this ABNF rule is defined in Appendix A of DSP0004, and the value range of both occurrences of `integerValue` is constrained by the integer datatype of the CIM element to which this qualifier is applied.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'UnsupportedValues')
and
Set { /* the integer datatypes */
   'CIMDatatypes::uint8',
   'CIMDatatypes::uint16',
   'CIMDatatypes::uint32',
   'CIMDatatypes::uint64',
   'CIMDatatypes::sint8',
   'CIMDatatypes::sint16',
   'CIMDatatypes::sint32',
   'CIMDatatypes::sint64'
}->includes(self.base_Property.type.qualifiedName)
implies
   self.UnsupportedValues->forAll( qv |
      qv.matchCimAbnf( ' UnsupportedValuesFormat' )
)
```

Q63. The *Deprecated* qualifier shall not be used with the *Experimental* qualifier

Context: Stereotype *CIM_Class* and *CIM_Indication*

OCL:
```
not
```

```
          (self.base_Class.getAppliedStereotype('CIM::Deprecated')
             <> null
          and self.Experimental
       )
```

Context: Stereotype *CIM_Association*

OCL:
```
not (
          self.base_AssociationClass.getAppliedStereotype('CIM::Deprecated')
             <> null
          and self.Experimental
       )
```

Context: Stereotypes *CIM_Property* and *CIM_Reference*

OCL:
```
not (
          self.base_Property.getAppliedStereotype('CIM::Deprecated')
             <> null
          and self.Experimental
       )
```

Context: Stereotype *CIM_Method*

OCL:
```
not (
          self.base_Operation.getAppliedStereotype('CIM::Deprecated')
             <> null
          and self.Experimental
       )
```

Context: Stereotype *CIM_Parameter*

OCL:
```
not (
          self.base_Parameter.getAppliedStereotype('CIM::Deprecated')
             <> null
          and self.Experimental
       )
```

Q64. The *EmbeddedObject* qualifier shall not be used with the *Key* qualifier.

Context: Stereotypes *CIM_Property*

OCL:
```
not (
          self.EmbeddedObject and
             self.base_Property.getAppliedStereotype('CIM::Key') <> null
       )
```

Q65. A property containing an embedded instance by value shall not be qualified with the *Key* qualifier.

Context: Stereotypes *CIM_Property*

OCL:
```
let s : Stereotype =
          self.base_Property.getAppliedStereotype('CIM::CIM_Property')
       in
       not (
          self.base_Property.type.oclIsTypeOf(uml::Class) and
          self.base_Property.getAppliedStereotype('CIM::Key') <> null
       )
```

Q66. The name of a CIM qualifier type shall conform to the format for CIM element names defined in DSP0004.

The format for CIM element names is described in 5.17.1.8.

Context: Stereotype *CIM_QualifierType*

OCL: `self.base_Property.name.isValidCimElementName()`

Q67. The value of the *ArrayType* stereotype property shall be consistent with the value of the *isOrdered* attribute.

Context: Stereotype *CIM_Property*

OCL:
```
if self.base_Property.isOrdered
then
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Indexed
else
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Ordered
or
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Bag
endif
```

Context: Stereotype *CIM_Parameter*

OCL:
```
if self.base_Parameter.isOrdered
then
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Indexed
else
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Ordered
or
   self.ArrayType = CIM::CIM_Qualifier_ArrayType_Enum::Bag
endif
```

Q68. The value of the *ClassReferenceType* stereotype property of a CIM property of class type shall be *ByValue.*

Context: Stereotype *CIM_Property*

OCL:
```
self.base_Property.type.oclIsTypeOf(uml::Class)
implies
self.ClassReferenceType = CIM::CIM_ClassReferenceType_Enum::ByValue
```

Q69. The value of the *ClassReferenceType* stereotype property of a CIM parameter of class type shall be *ByReference* or *ByValue.*

Context: Stereotype *CIM_Parameter*

OCL:
```
self.base_Parameter.type.oclIsTypeOf(uml::Class)
implies (
   self.ClassReferenceType = CIM::CIM_ClassReferenceType_Enum::ByValue or
   self.ClassReferenceType = CIM::CIM_ClassReferenceType_Enum::ByReference
)
```

Q70. The value of the *ClassReferenceType* stereotype property of a CIM method with a return value of class type shall be *ByReference* or *ByValue.*

Context: Stereotype *CIM_Method*

OCL:
```
let rv : OrderedSet(Parameter) =
   self.base_Operation.ownedParameter->select( p |
            p.direction = uml::ParameterDirectionKind::return)
      in
      rv->size() = 1 and
      rv->at(1).type.oclIsTypeOf(uml::Class)
      implies (
         self.ClassReferenceType = CIM::CIM_ClassReferenceType_Enum::ByValue or
         self.ClassReferenceType = CIM::CIM_ClassReferenceType_Enum::ByReference
      )
```

5.17.6 OCL Constraints on Datatypes

T1. The value range of the CIM datatypes uint8, uint16, uint32, and uint64 shall be 0 to 2**N-1 where N is 8, 16, 32 and 64, respectively.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::uint8' implies
self.base_Slot.value.integerValue()->forAll( iv | iv >= 0 and iv <= 255
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::uint16' implies
self.base_Slot.value.integerValue()->forAll( iv | iv >= 0 and iv <= 32767
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::uint32' implies
self.base_Slot.value.integerValue()->forAll( iv | iv >= 0 and iv <= 2**32-1
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::uint64' implies
self.base_Slot.value.integerValue()->forAll( iv | iv >= 0 and iv <= 2**64-1
)
```

T2. The value range of the CIM datatypes sint8, sint16, sint32, and sint64 shall be -2**(N-1) to 2**(N-1)-1 where N is 8, 16, 32 and 64, respectively.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::sint8' implies
self.base_Slot.value.integerValue()->forAll( iv |
   iv >= -128 and iv <= 127
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::sint16' implies
self.base_Slot.value.integerValue()->forAll( iv |
   iv >= -32768 and iv <= 32767
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::sint32' implies
self.base_Slot.value.integerValue()->forAll( iv |
   iv >= -2**31 and iv <= 2**31-1
```

```
    )
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::sint64' implies
self.base_Slot.value.integerValue()->forAll( iv |
   iv >= -2**64 and iv <= 2**64-1
)
```

T3. The value range of the CIM datatypes real32 and real64 shall be the value range specified for IEEE 4-byte and 8-byte floating point values, respectively, as defined in IEEE Standard 754.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::real32' implies
self.base_Slot.value.integerValue()->forAll( iv | iv.isValidCimReal32()
)
```

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::real64' implies
self.base_Slot.value.integerValue()->forAll( iv | iv.isValidCimReal64()
)
```

T4. The value range of the CIM datatype char16 shall be one Unicode character from the UCS-2 range.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::char16' implies
self.base_Slot.value.stringValue()->forAll( sv | sv.size() <= 1 and
   sv.isValidCimString()
)
```

T5. The value range of the CIM datatype string shall be an ordered array of Unicode characters from the UCS-2 range.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::string' implies
self.base_Slot.value.stringValue()->forAll( sv | sv.isValidCimString()
)
```

T6. The value range of the CIM datatype datetime shall be a string of 24 Unicode characters as defined in DSP0004.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::datetime' implies
self.base_Slot.value.stringValue()->forAll( sv | sv.isValidCimDatetime()
)
```

T7. The format of a value of a CIM::octetstring datatype shall conform to the format defined in DSP0004.

Represented in ABNF according to 3.4, the format defined in DSP0004 is:

```
OctetstringFormat = "0x" 4*(hexDigit hexDigit);
```

where the rule on the right hand side of this ABNF rule is defined in Appendix A of DSP0004, and the first 4 octets of the octet string (8 hexadecimal digits in the text encoding) are the number of octets in the represented octet string in big endian format with the length portion included in the octet count.

NOTE: This rule covers octetstrings with both variants of CIM datatypes, array of string and array of uint8.

Context: A stereotype extending UML *Slot*

OCL:
```
self.base_Slot.definingFeature.type.qualifiedName =
   'CIMDatatypes::octetstring' implies
self.base_Slot.value.stringValue()->forAll( sv |
   sv.matchCimAbnf( 'OctetstringFormat' ) and
   sv.size() = sv.substring(3,10).hexToInteger() * 2 + 2
)
```

5.17.7 OCL Constraints for Values of Directly Mapped Qualifiers

The OCL constraints in this subclause verify that directly mapped qualifiers do not have a value in the corresponding property of their qualifier type stereotype.

NOTE: The *ArrayType* qualifier is mapped to the stereotype property in addition to UML constructs, therefore it is not included in this subclause.

These OCL constraints should be surfaced with less severity than the others.

W1. Stereotype property *Abstract* shall not have a value – use the UML *isAbstract* attribute instead

Context: Stereotype *CIM_Class*

OCL:
```
let s : Stereotype = self.base_Class.getAppliedStereotype('CIM::CIM_Class')
in
self.base_Class.isSet(s,'Abstract') = false
```

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Abstract') = false
```

Context: Stereotype *CIM_Indication*

OCL:
```
let s : Stereotype =
   self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
in
self.base_Class.isSet(s,'Abstract') = false
```

W2. Stereotype property *Association* shall not have a value – already indicated by using UML *AssociationClass.*

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Association') = false
```

W3. Stereotype property *Aggregation* shall not have a value – use the UML aggregation indicator instead.

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Aggregation') = false
```

W4. Stereotype property *Composition* shall not have a value – use the UML aggregation indicator instead.

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Composition') = false
```

W5. Stereotype property *Aggregate* shall not have a value – use the UML aggregation indicator instead.

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
in
self.base_Property.isSet(s,'Aggregate') = false
```

W6. Stereotype property *Description* shall not have a value – use the documentation field instead.

Context: Stereotype *CIM_Class*

OCL:
```
let s : Stereotype =
   self.base_Class.getAppliedStereotype('CIM::CIM_Class')
in
self.base_Class.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Indication*

OCL:
```
let s : Stereotype =
   self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
in
self.base_Class.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
in
```

```
    self.base_Property.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'Description') = false
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'Description') = false
```

W7. Stereotype property *Experimental* shall not have a value – use the marker stereotype *Experimental* instead.

Context: Stereotype *CIM_Class*

OCL:
```
let s : Stereotype = self.base_Class.getAppliedStereotype('CIM::CIM_Class')
in
self.base_Class.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Indication*

OCL:
```
let s : Stereotype =
   self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
in
self.base_Class.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
in
self.base_Property.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'Experimental') = false
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
     self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
  in
  self.base_Parameter.isSet(s,'Experimental') = false
```

W8. Stereotype property *In* shall not have a value – use the UML direction indicator instead.

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
     self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
  in
  self.base_Parameter.isSet(s,'In') = false
```

W9. Stereotype property *Out* shall not have a value – use the UML direction indicator instead.

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
     self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
  in
  self.base_Parameter.isSet(s,'Out') = false
```

W10. Stereotype property *Key* shall not have a value – use the marker stereotype *Key* instead.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Property')
  in
  self.base_Property.isSet(s,'Key') = false
```

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
  in
  self.base_Property.isSet(s,'Key') = false
```

W11. Stereotype property *Min* shall not have a value – use the UML association multiplicity instead.

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
  in
  self.base_Property.isSet(s,'Min') = false
```

W12. Stereotype property *Max* shall not have a value – use the UML association multiplicity instead.

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
      self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
  in
  self.base_Property.isSet(s,'Max') = false
```

W13. Stereotype property *ClassConstraint* shall not have a value – use UML user model constraints instead.

Context: Stereotype *CIM_Class*

OCL:
```
let s : Stereotype = self.base_Class.getAppliedStereotype('CIM::CIM_Class')
  in
  self.base_Class.isSet(s,'ClassConstraint') = false
```

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
   self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
in
self.base_AssociationClass.isSet(s,'ClassConstraint') = false
```

Context: Stereotype *CIM_Indication*

OCL:
```
let s : Stereotype =
   self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
in
self.base_Class.isSet(s,'ClassConstraint') = false
```

W14. Stereotype property *MethodConstraint* shall not have a value – use UML user model constraints instead.

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'MethodConstraint') = false
```

W15. Stereotype property *PropertyConstraint* shall not have a value – use UML user model constraints instead.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'PropertyConstraint') = false
```

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
in
self.base_Property.isSet(s,'PropertyConstraint') = false
```

W16. Stereotype property *Octetstring* shall not have a value – use the type CIMDatatypes::octetstring instead.

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'Octetstring') = false
```

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Octetstring') = false
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
```

```
     self.base_Parameter.isSet(s,'Octetstring') = false
```

W17. Stereotype property *Override* shall not have a value – use the UML redefinition capabilities instead.

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
     self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
   in
   self.base_Operation.isSet(s,'Override') = false
```

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Property')
   in
   self.base_Property.isSet(s,'Override') = false
```

Context: Stereotype *CIM_Reference*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Reference')
   in
   self.base_Property.isSet(s,'Override') = false
```

W18. Stereotype property *Static* shall not have a value – use the UML static indicator instead.

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
     self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
   in
   self.base_Operation.isSet(s,'Static') = false
```

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
     self.base_Property.getAppliedStereotype('CIM::CIM_Property')
   in
   self.base_Property.isSet(s,'Static') = false
```

W19. Stereotype property *Terminal* shall not have a value – use the UML leaf indicator instead.

Context: Stereotype *CIM_Class*

OCL:
```
let s : Stereotype = self.base_Class.getAppliedStereotype('CIM::CIM_Class')
   in
   self.base_Class.isSet(s,'Terminal') = false
```

Context: Stereotype *CIM_Association*

OCL:
```
let s : Stereotype =
     self.base_AssociationClass.getAppliedStereotype('CIM::CIM_Association')
   in
   self.base_AssociationClass.isSet(s,'Terminal') = false
```

Context: Stereotype *CIM_Indication*

OCL:
```
let s : Stereotype =
     self.base_Class.getAppliedStereotype('CIM::CIM_Indication')
   in
   self.base_Class.isSet(s,'Terminal') = false
```

W20. Stereotype property *Write* shall not have a value – use the UML read-only indicator instead.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'Write') = false
```

W21. Stereotype property *EmbeddedInstance* shall not have a value – use a class as a type instead.

Context: Stereotype *CIM_Property*

OCL:
```
let s : Stereotype =
   self.base_Property.getAppliedStereotype('CIM::CIM_Property')
in
self.base_Property.isSet(s,'EmbeddedInstance') = false
```

Context: Stereotype *CIM_Method*

OCL:
```
let s : Stereotype =
   self.base_Operation.getAppliedStereotype('CIM::CIM_Method')
in
self.base_Operation.isSet(s,'EmbeddedInstance') = false
```

Context: Stereotype *CIM_Parameter*

OCL:
```
let s : Stereotype =
   self.base_Parameter.getAppliedStereotype('CIM::CIM_Parameter')
in
self.base_Parameter.isSet(s,'EmbeddedInstance') = false
```

5.17.8 Other OCL Constraints

O1. A comment shall not have further comments.

Context: A stereotype extending UML *Comment*

OCL: `self.base_Comment.ownedComment->isEmpty()`

O2. A CIM constraint shall have public visibility.

Context: A stereotype extending UML *Constraint*

OCL: `self.base_Constraint.visibility = uml::VisibilityKind::public`

O3. A CIM constraint shall not have a name.

Context: A stereotype extending UML *Constraint*

OCL:
```
self.base_Constraint.name = null
or
self.base_Constraint.name = ''
```

O4. A CIM constraint shall not have a comment.

Context: A stereotype extending UML *Constraint*

OCL: `self.base_Constraint.ownedComment->isEmpty()`

O5. The constraint expression of a CIM constraint shall have public visibility.

Context: A stereotype extending UML *Constraint*

OCL: `self.base_Constraint.specification.visibility = uml::VisibilityKind::public`

O6. The constraint expression of a CIM constraint shall not have a name.

Context: A stereotype extending UML *Constraint*

OCL:
```
self.base_Constraint.specification.name = null
or
self.base_Constraint.specification.name = ''
```

O7. The constraint expression of a CIM constraint shall not have a comment.

Context: A stereotype extending UML *Constraint*

OCL:
```
self.base_Constraint.specification.ownedComment->isEmpty()
```

O8. The constraint expression of a CIM constraint shall use OCL as a language.

Context: A stereotype extending UML *Constraint*

OCL:
```
let lang : OrderedSet(String) = self.base_Constraint.specification.
    oclAsType(uml::OpaqueExpression).language
in
lang->isEmpty() /* OCL is implied if no language specified */
or
( lang->size() = 1 and lang->at(1) = 'OCL')
```

O9. A *LiteralSpecification* shall not have a name.

Context: A stereotype extending UML *LiteralSpecification*

OCL:
```
self.base_LiteralSpecification.name = null
or
self.base_LiteralSpecification.name = ''
```

O10. A *LiteralSpecification* shall have public visibility.

Context: A stereotype extending UML *LiteralSpecification*

OCL:
```
self.base_LiteralSpecification.visibility = uml::VisibilityKind::public
```

O11. A *LiteralSpecification* shall not reference a type.

Context: A stereotype extending UML *LiteralSpecification*

OCL:
```
self.base_LiteralSpecification.type->isEmpty()
```

O12. A *LiteralSpecification* shall not have a comment.

Context: A stereotype extending UML *LiteralSpecification*

OCL:
```
self.base_LiteralSpecification.ownedComment->isEmpty()
```

O13. An *Enumeration* shall not be abstract.

Context: Stereotype *CIM_Enumeration*

OCL:
```
self.base_Enumeration.isAbstract = false
```

O14. An *Enumeration* shall not be a leaf.

Context: Stereotype *CIM_Enumeration*

OCL:
```
self.base_Enumeration.isLeaf = false
```

O15. An *Enumeration* shall be defined nested in a *Class.*

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.owner.oclIsTypeOf(uml::Class)`

O16. An *Enumeration* shall be used as a type for a property, operation parameter, or operation return value defined in the *Class* defining the *Enumeration.*

Context: Stereotype *CIM_Enumeration*

OCL:
```
let enumParms : Sequence(Parameter) =
   self.base_Enumeration.owner.oclAsType(uml::Class)
      .ownedOperation.ownedParameter
         ->select( p | p.type = self.base_Enumeration)
in
let enumProps : OrderedSet(Property) =
   self.base_Enumeration.owner.oclAsType(uml::Class).ownedAttribute
      ->select( p | p.type = self.base_Enumeration)
in
enumProps->size() = 1 xor
enumParms->size() = 1
```

O17. An *Enumeration* shall have public visibility.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.visibility = uml::VisibilityKind::public`

O18. An *Enumeration* shall not own UML attributes.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.ownedAttribute->isEmpty()`

O19. An *Enumeration* shall not own UML operations.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.ownedOperation->isEmpty()`

O20. An *Enumeration* shall not have superclasses other than either one CIM datatype or one other enumeration.

Context: Stereotype *CIM_Enumeration*

OCL:
```
self.base_Enumeration.generalization->isEmpty() = false
implies
self.base_Enumeration.generalization->size() = 1 and (
   self.base_Enumeration.generalization->asOrderedSet()
      ->at(1).general.namespace.name = 'CIMDatatypes'  or
   self.base_Enumeration.generalization->asOrderedSet()
      ->at(1).general.oclIsTypeOf(uml::Enumeration)
)
```

O21. An *Enumeration* shall not redefine another classifier.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.redefinedClassifier->isEmpty()`

O22. An *Enumeration* shall not import any elements.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.elementImport->isEmpty()`

O23. An *Enumeration* shall not import any packages.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.packageImport->isEmpty()`

O24. An *Enumeration* shall not own any UML comments.

Context: Stereotype *CIM_Enumeration*

OCL: `self.base_Enumeration.ownedComment->isEmpty()`

O25. An *Enumeration* shall have its *IsValuesDefined* or its *IsValueMapDefined* stereotype property set to true.

Context: Stereotype *CIM_Enumeration*

OCL: `self.IsValuesDefined or self.IsValueMapDefined`

O26. An Enumeration used as a type for a CIM property shall have the name: <property-name> '_Enum'

Context: Stereotype *CIM_Property*

OCL:
```
self.base_Property.type.oclIsTypeOf(uml::Enumeration)
implies
self.base_Property.type.name =
  self.base_Property.name
    .concat('_Enum')
```

O27. An Enumeration used as a type for a CIM method return value shall have the name: <method-name> '_Enum'

Context: A stereotype extending UML *Parameter*

OCL:
```
self.base_Parameter.type.oclIsTypeOf(uml::Enumeration) and
self.base_Parameter.direction = uml::ParameterDirectionKind::return
implies
self.base_Parameter.type.name =
  self.base_Parameter.operation.name
    .concat('_Enum')
```

O28. An Enumeration used as a type for a CIM parameter shall have the name: <method-name> '_' <parameter-name> '_Enum'

Context: Stereotype *CIM_Parameter*

OCL:
```
self.base_Parameter.type.oclIsTypeOf(uml::Enumeration) and
self.base_Parameter.direction <>uml::ParameterDirectionKind::return
implies
self.base_Parameter.type.name =
  self.base_Parameter.operation.name
    .concat('_')
    .concat(self.base_Parameter.name)
    .concat('_Enum')
```

O29. An *EnumerationLiteral* in an *Enumeration* with its *IsValuesDefined* stereotype property set to false shall have a value that equals its name.

Context: A stereotype extending UML *EnumerationLiteral*

OCL:
```
self.base_EnumerationLiteral.enumeration
        .getAppliedStereotype('CIM::CIM_Enumeration')
        .oclAsType(CIM::CIM_Enumeration).IsValuesDefined = false
    implies (
        let s : ValueSpecification =
            self.base_EnumerationLiteral.specification
        in
        if s.oclIsTypeOf(uml::LiteralString) then
            self.base_EnumerationLiteral.name =
                s.oclAsType(uml::LiteralString).value
        else
            self.base_EnumerationLiteral.name =
                s.oclAsType(uml::LiteralInteger).value
        endif
    )
```

O30. An *EnumerationLiteral* in an *Enumeration* with its *IsValueMapDefined* stereotype property set to false shall have a value that equals its position in the *Enumeration.*

Context: A stereotype extending UML *EnumerationLiteral*

OCL:
```
self.base_EnumerationLiteral.enumeration
        .getAppliedStereotype('CIM::CIM_Enumeration')
        .oclAsType(CIM::CIM_Enumeration).IsValueMapDefined = false
    implies (
        let i : Integer =
            self.base_EnumerationLiteral.specification
                .oclAsType(uml::LiteralInteger).value
        in
            self.base_EnumerationLiteral.enumeration
                .ownedLiteral->asOrderedSet()->at(i)
                    = self.base_EnumerationLiteral
    )
```

O31. An *EnumerationLiteral* shall have public visibility.

Context: A stereotype extending UML *EnumerationLiteral*

OCL: `self.base_EnumerationLiteral.visibility = uml::VisibilityKind::public`

O32. An *EnumerationLiteral* shall not have UML slots.

Context: A stereotype extending UML *EnumerationLiteral*

OCL: `self.base_EnumerationLiteral.slot->isEmpty()`

O33. An *EnumerationLiteral* shall have one *LiteralInteger* or *LiteralString* as a value.

Context: A stereotype extending UML *EnumerationLiteral*

OCL:
```
let s : ValueSpecification = self.base_EnumerationLiteral.specification
    in
    s->size() = 1 and (
        s.oclIsTypeOf(uml::LiteralInteger) or
        s.oclIsTypeOf(uml::LiteralString)
```

)

5.17.9 Other Constraints

This subclause lists constraints that cannot be expressed in OCL.

N1. A CIM qualifier with inheritance flavor *ToSubclass DisableOverride* shall not be overridden.

5.17.10 Constraints Covered by UML

This subclause lists CIM constraints for which the CIM related profiles do not need to define specific OCL constraints because these situations are handled by constraints defined by UML, and so a UML tool should be supporting them already.

U1. A CIM property may by owned by only one CIM class.

U2. A CIM method may by owned by only one CIM class.

U3. The CIM class overriding a CIM property shall be a subclass of the class defining the property.

U4. The CIM class overriding a CIM method shall be a subclass of the class defining the method.

U5. The name of a property defined in a CIM class shall be unique within the defining class.

U6. The name of a method defined in a CIM class shall be unique within the defining class.

U7. The name of a CIM method parameter shall be unique within the defining method.

U8. A CIM association that inherits another association shall have the same arity.

U9. The arity of a CIM association shall be at least two.

U10. The default value of a CIM property shall match the type of the property.

U11. The name of a CIM qualifier type shall be unique within the owning profile.

U12. A CIM reference shall be owned by a CIM association.

U13. A CIM reference shall not be an array.

U14. The CIM element referenced in an Override qualifier value shall exist in one of the superclasses.

U15. The constraint expression of a CIM OCL constraint shall be of type UML Boolean.

U16. The Min qualifier shall not be NULL.

U17. The Octetstring qualifier is applicable only to CIM string array and uint8 array datatypes.

5.18 Extension Points

This subclause defines extension points that may be used by conforming implementations of the UML profile for CIM in order to support additional functionality.

5.18.1 Attributes and Associations in UML Metaclasses Used for the Mapping

The UML metaclasses used in the mapping define several extension points at the level of attributes or associations of such metaclasses. In the mapping tables in this document, they are all marked with the text "extension point". These extension points are further defined here:

- *name* attribute in *InstanceSpecification* metaclass used for modeled CIM instances. This allows defining a name for modeled CIM instances in UML although this is not supported in CIM MOF.
- *ownedComment* association in *InstanceSpecification* metaclass used for modeled CIM instances. This allows defining comments or descriptions on modeled CIM instances. Note that such comments or descriptions will not be represented in CIM MOF.
- *ownedComment* association in *Slot* metaclass used for property values of modeled CIM instances. This allows defining comments or descriptions on property values of modeled CIM instances. Note that such comments or descriptions will not be represented in CIM MOF.
- *icon* association in *Stereotype* metaclass used as Qualifier Type Stereotype. This allows usage of a specific icon for qualifier types. Note that such icons will not be represented in CIM MOF.
- *icon* association in *Stereotype* metaclass used as Meta Element Stereotypes. This allows usage of a specific icon for each CIM meta element. Note that such icons will not be represented in CIM MOF.
- *ownedComment* association in *Property* metaclass used for definitional aspects of CIM qualifier value within Qualifier Type Stereotype. This allows defining comments or descriptions on qualifier types. Note that such comments or descriptions will not be represented in CIM MOF.
- *name* attribute in *Constraint* metaclass used for OCL related constraint qualifiers. This allows giving these OCL constraints a name to distinguish them. Note that such names will not be represented in CIM MOF.
- *ownedRule* association in *Enumeration* metaclass used for used for *Values* and *ValueMap* qualifiers. This allows defining additional constraints on such enumerations. Note that such additional constraints will not be represented in CIM MOF.
- *ownedComment* association in *EnumerationLiteral* metaclass used for pair of *Values* and *ValueMap* qualifier values. This allows defining comments or descriptions on each pair of *Values* and *ValueMap* qualifier values. Note that such comments or descriptions will not be represented in CIM MOF.
- *ownedRule* association in *DataType* metaclass used for CIM datatypes. This allows defining additional constraints on such datatypes (for example value ranges or patterns). Note that such additional constraints will not be represented in CIM MOF.

5.18.2 Additional Metamodel Constraints

Conforming implementations of the UML profile for CIM may define additional constraints in the UML profiles "CIM" and "CIMQualifierType" and in the UML type library "CIMDatatypes", as long as such constraints are supported by DSP0004.

5.18.3 Additional Stereotypes

Conforming implementations of the UML profile for CIM may define additional stereotypes on any UML metaclasses in the UML profiles "CIM" and "CIMQualifierType". Those stereotypes may own properties.

Conforming implementations of the UML profile for CIM may define additional properties on any stereotypes defined in this document.

If such property values cannot be represented in CIM MOF, their values will be lost when exporting the UML model to CIM MOF.

5.19 Compatibility Considerations

There are two kinds of compatibility of future versions of this document:

- Compatibility with respect to the UML profile for CIM

A future release of this document is said to be downward compatible with respect to the UML profile for CIM if it does not mandate any changes in the definition of the UML profiles and type libraries defined in 5.21.

- Compatibility with respect to user models using the UML profile for CIM

  A future release of this document is said to be downward compatible with respect to user models using the UML profile for CIM if it does not mandate any changes in the definition of the user model, except for possibly referencing updated versions of the UML profiles and type libraries defined in 5.21.

Any versions of this document that have the same major version number shall be compatible to each other with respect to user models using the UML profile for CIM and should be compatible to each other with respect to the UML profile for CIM.

This definition of compatibility allows minor version updates of this document to allow elements to be associated with UML metaclass association ends that currently are required to have no elements associated. For example, the *packageImport* association of the UML metaclass *Class* — which currently is required to have no associated elements — might be used for some purpose in the future.

This may cause the UML profiles and type libraries for CIM to be updated in order to support future minor version updates of this document, and in order to deal with user models being transferred from other UML tools that already have updated their UML profiles and type libraries.

5.20 Limitations

This subclause lists any limitations of the mapping of CIM to UML, from a perspective of CIM.

- The arity of aggregations and compositions is limited to two, while CIM allows their arity to be larger than two. Note that for associations that are not aggregations or compositions, any arity is supported.

  Reason: UML 2 restricts the arity of aggregations and compositions to two.

- The definition of CIM flavors on qualifier values is not supported. Note that CIM flavors used as part of qualifier type declarations are supported.

  Reason: Flavors on qualifier values are intended to be removed from CIM, are not used in the current CIM Schema, and would have introduced additional complexity.

- Information about MOF files is mapped in a limited way.

  Any `#pragma` directives are represented only from the point of view of their values relevant for CIM classes and CIM instances. This allows reconstructing semantically equivalent CIM MOF files from the UML user model, but information such as where exactly a #pragma was defined in the original CIM MOF file will not be represented in the UML user model. Also, the include hierarchy of CIM MOF files is not represented in the UML user model.

- CIM elements qualified with *EmbeddedObject* are treated like ordinary strings instead of mapping their types to classes.

  Reason: It is not known in the user model whether the CIM element represents an instance or a class. The qualifier *EmbeddedInstance* has been introduced because of this, and it should be used in the CIM Schema instead of *EmbeddedObject*, where applicable.

- There is no mapping for CIM Triggers.

  Reason: CIM Triggers, while defined in the CIM metamodel, are not used in the CIM Schema, nor is it clear how they would be used.

5.21 Definition of UML Profiles and Type Libraries

Beginning with UML 2.0, UML profiles are represented by the UML *Profile* metaclass, so they have become a runtime entity that can be modified dynamically as new qualifiers are added.

In this document, the term "UML profile" is used for the instances of the UML *Profile* metaclass, and for the files that are the physical representation of the UML Profile in some format. If a specific notion of "UML profile" is referred to, appropriate qualifications are used.

Note that a UML type library is a normal UML user model that happens to contain type definitions and therefore is called "type library".

This subclause normatively defines the existence and contents of the UML profiles and UML type libraries that represent the UML Profile for CIM. It references UML constructs defined in other subclauses when defining the contents.

5.21.1 Profile "CIM"

There shall be a UML profile named "CIM".

A UML user model that intends to become a CIM model shall apply the UML profile "CIM".

It is left to the UML user model whether the UML profile "CIM" is applied strictly or non-strictly.

The UML profile "CIM" shall define the following elements:

- All meta element stereotypes, as defined in 5.15.
- All qualifier scope stereotypes, as defined in 5.13.2.
- All qualifier type stereotypes including any enumerations they use, as defined in 5.13.2. Note that these stereotypes depend on the qualifier type declarations in CIM, so this causes the UML profile "CIM" to be modified during the act of importing CIM qualifier type declarations.
- All marker stereotypes, as defined in the subclauses of 5.13.4.
- Any relations between these elements, as defined in this document.

The UML profile "CIM" shall import the UML type library "CIMDatatypes".

5.21.2 Profile "CIMQualifierType"

There shall be a UML profile named "CIMQualifierType". This UML profile shall not be applied to any UML user models. It shall be applied non-strictly to the UML profile "CIM".

The UML profile "CIMQualifierType" shall define the following elements:

- The stereotype *CIM_QualifierType*, as defined in Table 13.
- The stereotype *CIM_QualifierInheritanceRule*, as defined in Table 14.
- Any relations between these elements, as defined in this document.

5.21.3 Type Library "CIMDatatypes"

There shall be a UML type library named "CIMDatatypes". It shall have the stereotype "modelLibrary" applied.

A UML user model that intends to become a CIM model shall import the UML type library "CIMDatatypes".

The UML type library "CIMDatatypes" shall define the following elements:

- The *DataType* metaclass instances defined in Table 8.

ANNEX A
(informative)

UML Metamodel Diagrams

The diagrams in this appendix show the UML metaclasses involved in this document.

Figure 3 shows the stereotype inheritance hierarchy resulting from the mapping of CIM qualifiers. The stereotypes shown as an example are only those for CIM classes and CIM properties, and only a subset of the qualifier types.

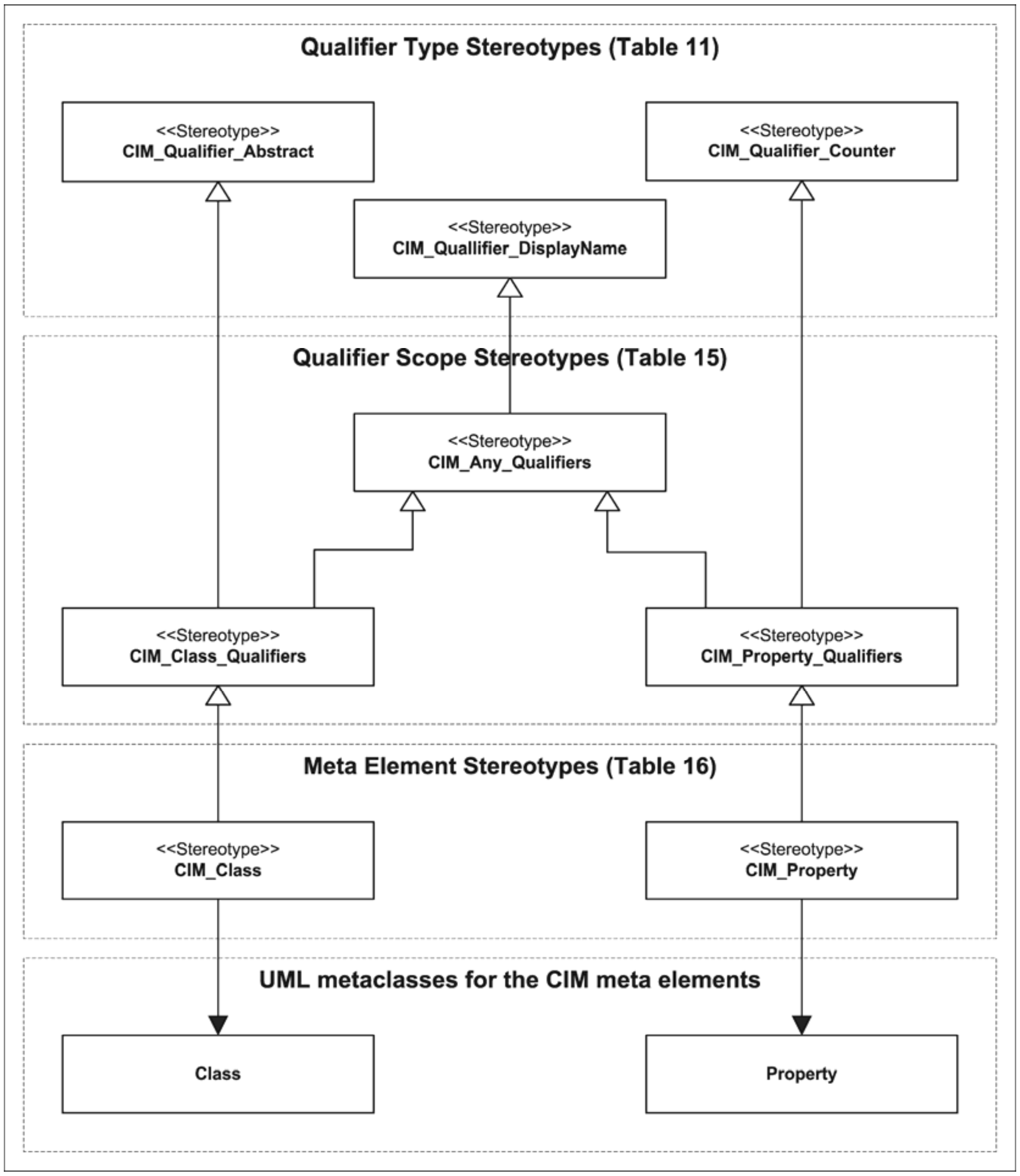
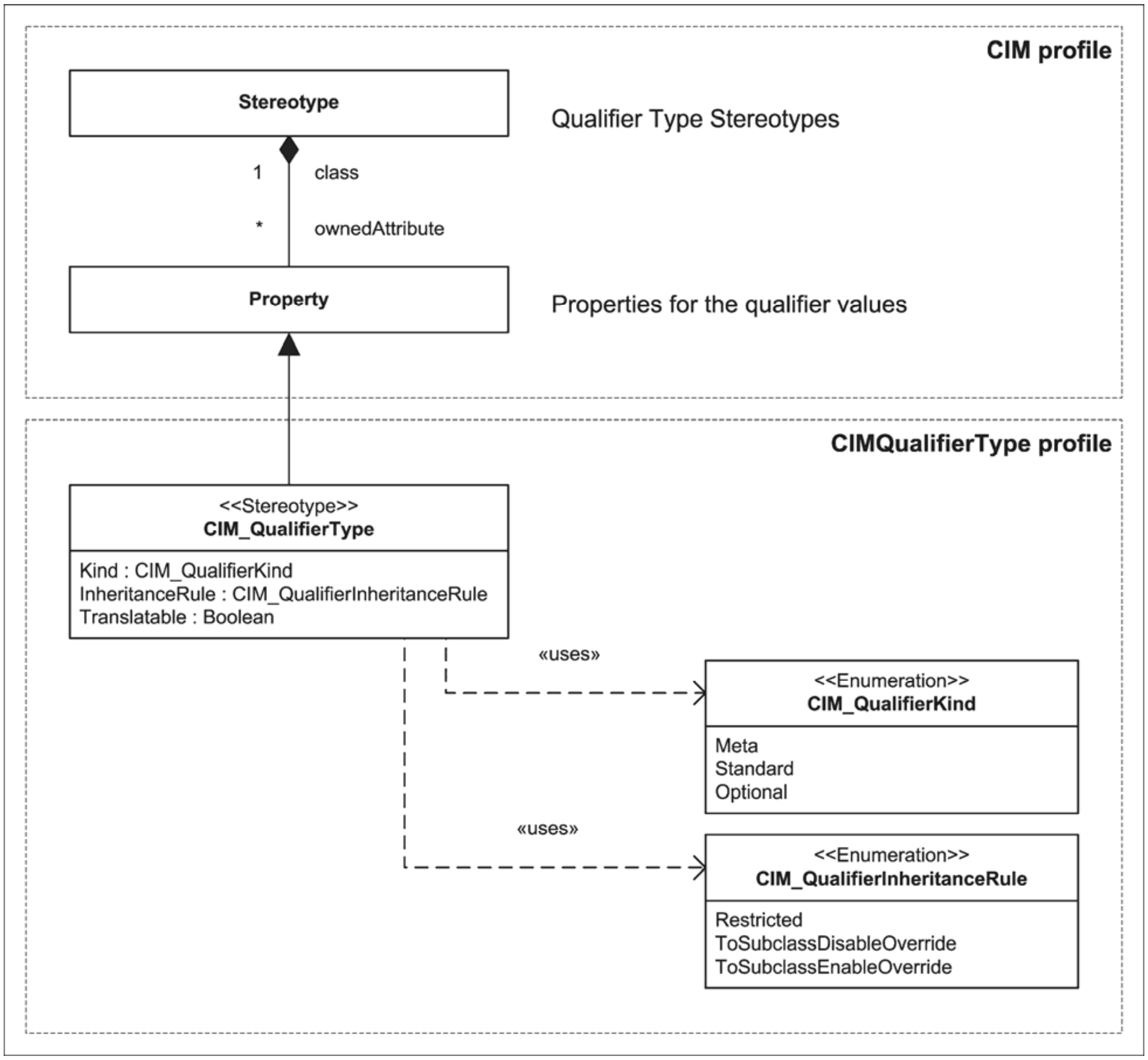

Qualifier Type Stereotypes (Table 11)
<<Stereotype>> CIM_Qualifier_Abstract
<<Stereotype>> CIM_Qualifier_Counter
<<Stereotype>> CIM_Quallifier_DisplayName
Qualifier Scope Stereotypes (Table 15)
<<Stereotype>> CIM_Any_Qualifiers
<<Stereotype>> CIM_Class_Qualifiers
<<Stereotype>> CIM_Property_Qualifiers
Meta Element Stereotypes (Table 16)
<<Stereotype>> CIM_Class
<<Stereotype>> CIM_Property
UML metaclasses for the CIM meta elements
Class
Property

Figure 3 – Inheritance of Qualifier Related Stereotypes

3545 Figure 4 shows the contents of the UML profile "CIMQualifierType" as defined in 5.21.2.



3546

3547 **Figure 4 – Contents of UML Profile "CIMQualifierType"**

Figure 5 shows a high level view of the UML metaclasses involved in representing a normal (non-association, non-indication) CIM class. It does not show the UML metaclass instances for lower level elements such as DataTypes, Comments or Constraints. Between the elements it includes, it shows all UML meta-associations that are compositions but leaves away some non-compositions. Note that the elements labeled with "<<Stereotype>>" are stereotype definitions.

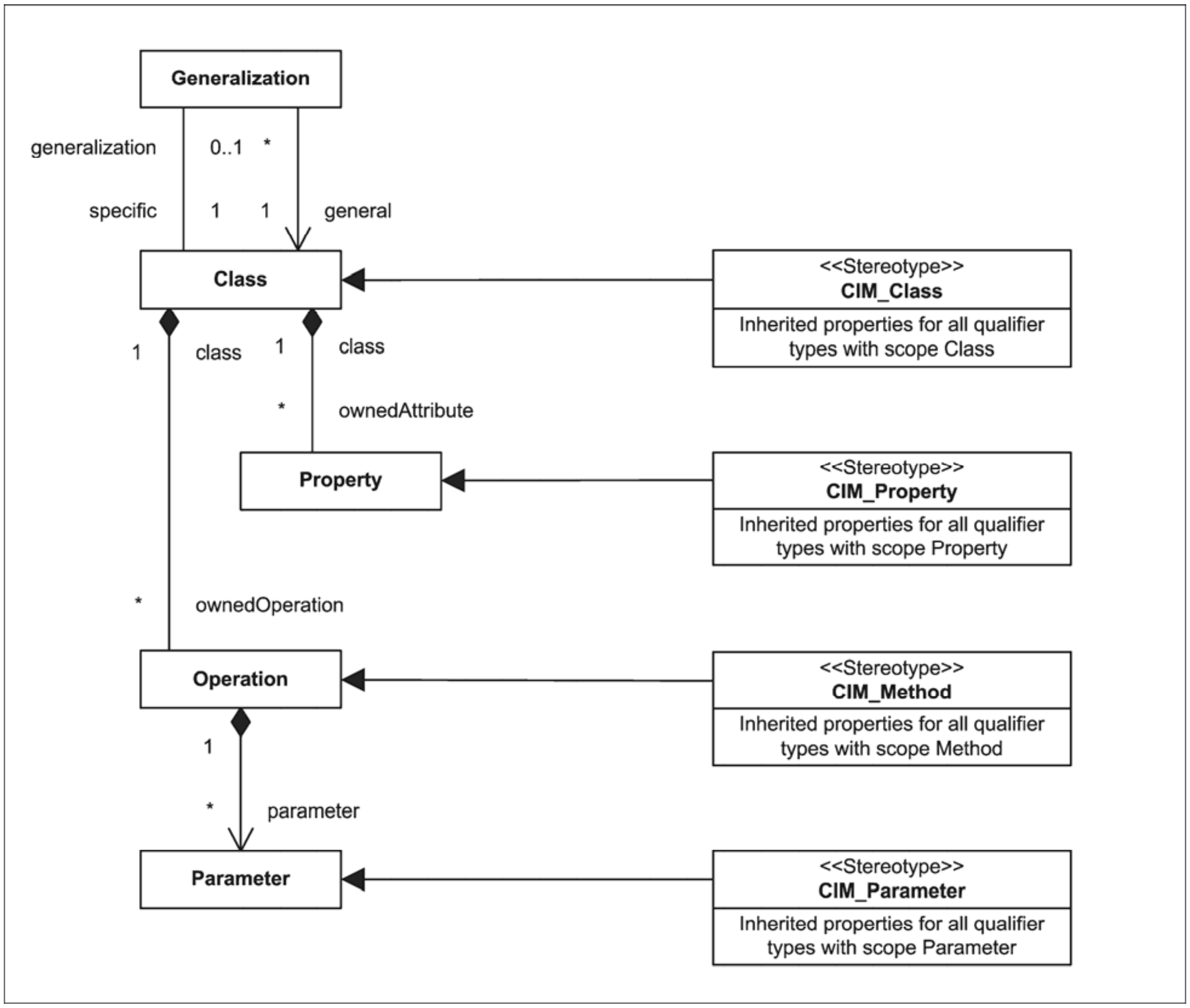

Generalization
generalization
0..1
*
specific
1
1
general
Class
<<Stereotype>>
CIM_Class
Inherited properties for all qualifier types with scope Class
1
class
1
class
*
ownedAttribute
Property
<<Stereotype>>
CIM_Property
Inherited properties for all qualifier types with scope Property
*
ownedOperation
Operation
<<Stereotype>>
CIM_Method
Inherited properties for all qualifier types with scope Method
1
*
parameter
Parameter
<<Stereotype>>
CIM_Parameter
Inherited properties for all qualifier types with scope Parameter

Figure 5 – UML Metaclasses Representing a CIM Class (high level)

Figure 6 shows the UML metaclass instances representing a simple CIM class specified in the CIM MOF fragment shown in Figure 7. Again, some lower level elements have been left out for clarity. Note that the elements labeled with "<<Stereotype>>" are applied stereotypes, i.e., they are not stereotype definitions.

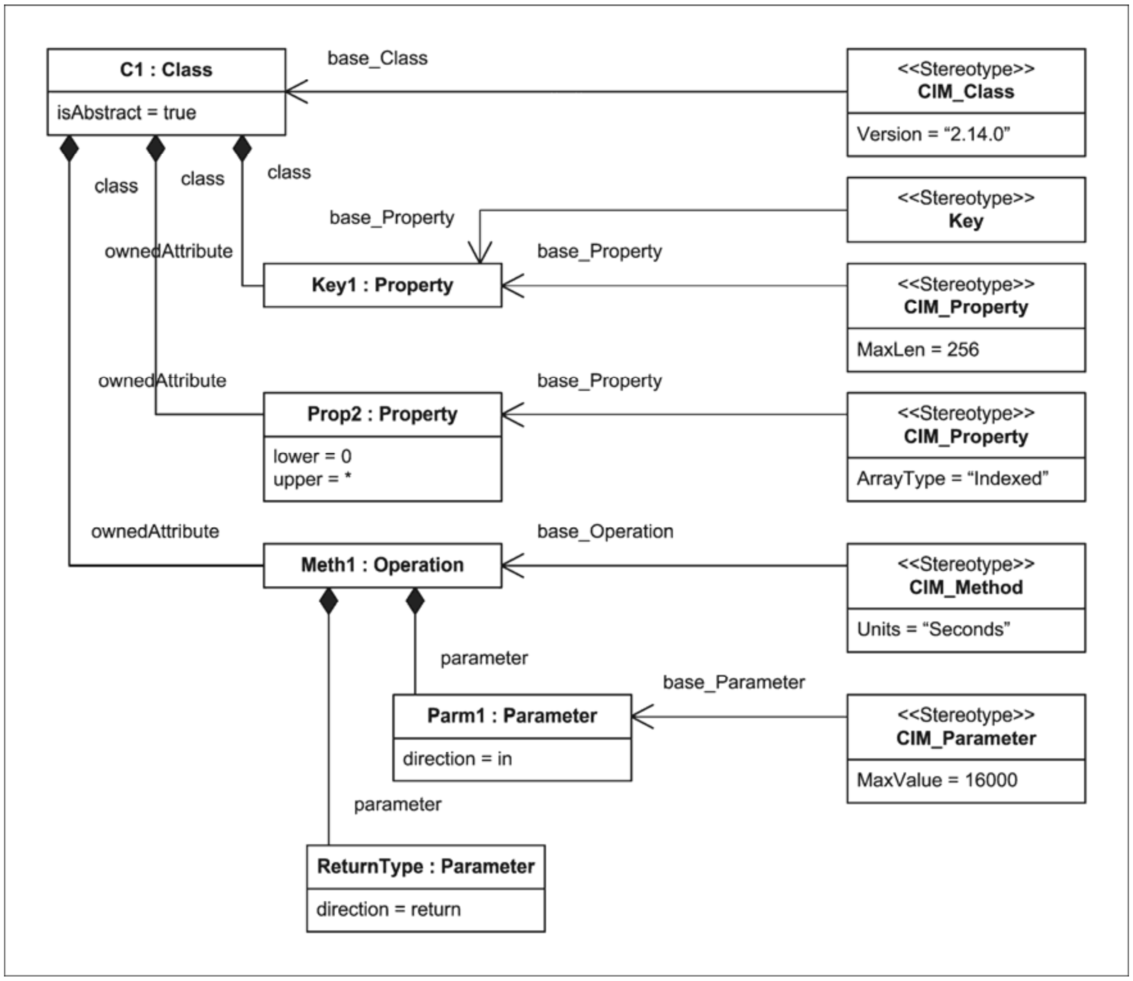


Figure 6 – UML Metaclass Instances for a Simple CIM Class

```
[Abstract, Version (“2.14.0”)]
class XMP_C1 {

     [Key, MaxLen(256)]
     string Key1;

     [ArrayType(“Indexed”)]
     string Prop2 [];
```

Figure 7 – CIM MOF Definition of a Simple CIM Class

ANNEX B (informative)

Change Log

Version	Date	Author	Description
1.0.0	08/11/2009		DMTF Standard Release

Bibliography

This section contains references to useful documents which are not normatively used by this document.

OMG *MOF 2.0 XMI Mapping Specification, Version 2.1*, http://www.omg.org/cgi-bin/doc?formal/05-09-01

OMG *MOF Core Specification, Version 2.0*, http://www.omg.org/cgi-bin/doc?formal-06-01-01

OMG *MOF 2.0 Query / Views / Transformations Specification*, http://www.omg.org/cgi-bin/doc?ptc/2005-11-01