magicdraw®

Architecture Made Simple

SYSML METAMODEL

version 17.0.1
user guide

No Magic, Inc.
2011

INTRODUCTION

This document presents the MagicDraw SysML Profile structure and its representation in MagicDraw. For more information about SysML, see the latest SysML specification at http://www.omgsysml.org/.
The MagicDraw SysML Profile document lists MagicDraw SysML Profile elements in alphabetical order. The element description includes table with the following columns: attribute name, attribute type, attribute owner and sample template expression (VTL).
See the sample of the table below.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
allocatedFrom	NamedElement	Allocated	`$Allocated[i].allocatedFrom`
allocatedTo	NamedElement	Allocated	`$Allocated[i].allocatedTo`

Table – sample of MagicDraw SysML Profile element description

Attribute Name
The Attribute Name column provides name of property used in the MagicDraw SysML Profile.

Attribute Type
The Attribute Type column provides name of property's type (another MagicDraw SysML Profile element).

Attribute Owner
The Attribute Owner column provides name of property's owner in model hierarchy. Some elements properties are derived from super elements.

Sample Template Expression (VTL) for reports generation

Sample Template Expression (VTL) is the last column from the table, which gives the expression for reports generation. This expression allows to print value of the element's attribute in a report. For more information about VTL code, please see "MagicDraw Report Wizard UserGuide.pdf", "Template Variables" section.

CONTENTS

1. Actuator

An Actuator is a special external system that influences the environment of the system under development. For example a Heater assembly or a Central locking system of a car.

Base Classifier

- External system

2. Allocate

Allocate is a dependency based on UML::abstraction. It is a mechanism for associating elements of different types, or in different hierarchies, at an abstract level. Allocate is used for assessing user model consistency and directing future design activity. It is expected that an «allocate» relationship between model elements is a precursor to a more concrete relationship between the elements, their properties, operations, attributes, or sub-classes.

3. AllocateActivityPartition

AllocateActivityPartition is used to depict an «allocate» relationship on an Activity diagram. The AllocateActivityPartition is a standard UML2::ActivityPartition, with modified constraints as stated in the paragraph below.

4. Allocated

«allocated» is a stereotype that applies to any NamedElement that has at least one allocation relationship with another NamedElement. «allocated» elements may be designated by either the /from or /to end of an «allocate» dependency. The «allocated» stereotype provides a mechanism for a particular model element to conveniently retain and display the element at the opposite end of any «allocate» dependency. This stereotype provides for the properties "allocatedFrom" and "allocatedTo," which are derived from the «allocate» dependency.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
allocatedFrom	NamedElement	Allocated	`$Allocated[i].allocatedFrom`
allocatedTo	NamedElement	Allocated	`$Allocated[i].allocatedTo`

5. BasicInterval

Basic Interval distribution - value between min and max inclusive

Base Classifier

- DistributedProperty

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
max	Real	BasicInterval	`$BasicInterval[i].max`
min	Real	BasicInterval	`$BasicInterval[i].min`

6. BindingConnector

A Binding Connector is a connector which specifies that the properties at both ends of the connector have equal values. If the properties at the ends of a binding connector are typed by a DataType or ValueType, the connector specifies that the instances of the properties must hold equal values, recursively through any nested properties within the connected properties. If the properties at the ends of a binding connector are typed by a Block, the connector specifies that the instances of the properties must refer to the same block instance. As with any connector owned by a SysML Block, the ends of a binding connector may be nested within a multi-level path of properties accessible from the owning block. The NestedConnectorEnd stereotype is used to represent such nested ends just as for nested ends of other SysML connectors.

Base Classifier

- InvisibleStereotype

7. Block

A Block is a modular unit that describes the structure of a system or element. It may include both structural and behavioral features, such as properties and operations, that represent the state of the system and behavior that the system may exhibit. Some of these properties may hold parts of a system, which can also be described by blocks. A block may include a structure of connectors between its properties to indicate how its parts or other properties relate to one another. SysML blocks provide a general-purpose capability to describe the architecture of a system. They provide the ability to represent a system hierarchy, in which a system at one level is composed of systems at a more basic level. They can describe not only the connectivity relationships between the systems at any level, but also quantitative values or other information about a system. SysML does not restrict the kind of system or system element that may be described by a block. Any reusable form of description that may be applied to a system or a set of system characteristics may be described by a block. Such reusable descriptions, for example, may be applied to purely conceptual aspects of a system design, such as relationships that hold between parts or properties of a system. Connectors owned by SysML blocks may be used to define relationships between parts or other properties of the same containing block. The type of a connector or its connected ends may specify the semantic interpretation of a specific connector.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$Block[i].isEncapsulated`

8. BlockHierarchy

Block definition diagram usage for a block hierarchy - Block Hierarchy where block can be replaced by system, item, activity, etc.

9. Boundary system

A Boundary system is a special external system that serves as medium between another system and the system under development without having own interests in the communication. For example Bus system or Communication system.

Base Classifier

- External system

10. businessRequirement

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$businessRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$businessRequirement[i].DerivedFrom`
Id	String	Requirement	`$businessRequirement[i].Id`
Master	Requirement	Requirement	`$businessRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$businessRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$businessRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$businessRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$businessRequirement[i].source`
Text	String	Requirement	`$businessRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$businessRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$businessRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$businessRequirement[i].verifyMethod`

11. Complex

A Complex value type represents the mathematical concept of a complex number. A complex number consists of a real part defined by a real number, and an imaginary part defined by a real number multiplied by the square root of -1. Complex numbers are used to express solutions to various forms of mathematical equations.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
imaginaryPart	Real	Complex	`$Complex[i].imaginaryPart`

realPart	Real	Complex	`$Complex[i].realPart`

12. Conform

A Conform relationship is a dependency between a view and a viewpoint. The view conforms to the specified rules and conventions detailed in the viewpoint. Conform is a specialization of the UML dependency, and as with other dependencies the arrow direction points from the (client/source) to the (supplier/target).

13. ConnectorProperty

Connectors can be typed by association classes that are stereotyped by Block (association blocks). These connectors specify instances (links) of the association block that exist due to instantiation of the block owning or inheriting the connector. The value of a connector property on an instance of a block will be exactly those link objects that are instances of the association block typing the connector referred to by the connector property.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
connector	Connector	ConnectorProperty	`$ConnectorProperty[i].connector`

14. ConstraintBlock

A constraint block is a block that packages the statement of a constraint so it may be applied in a reusable way to constrain properties of other blocks. A constraint block typically defines one or more constraint parameters, which are bound to properties of other blocks in a surrounding context where the constraint is used. Binding connectors, as defined in Chapter 8: Blocks, are used to bind each parameter of the constraint block to a property in the surrounding context. All properties of a constraint block are constraint parameters, with the exception of constraint properties that hold internally nested usages of other constraint blocks.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$ConstraintBlock[i].isEncapsulated`

15. ConstraintProperty

A constraint property is a property of any block that is typed by a constraint block. It holds a localized usage of the constraint block. Binding connectors may be used to bind the parameters of this constraint block to other properties of the block that contains the usage.

Base Classifier

- InvisibleStereotype

16. ContextDiagram

A user defined usage of an internal block diagram, which depicts some of the top level entities in the overall enterprise and their relationships.

17. Continuous

Continuous rate is a special case of rate of flow (see Rate) where the increment of time between items approaches zero. It is intended to represent continuous flows that may correspond to water flowing through a pipe, a time continuous signal, or continuous energy flow. It is independent from UML streaming. A streaming parameter may or may not apply to continuous flow, and a continuous flow may or may not apply to streaming parameters.

Base Classifier

- Rate

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
rate	String	Rate	`$Continuous[i].rate`

18. ControlOperator

A control operator is a behavior that is intended to represent an arbitrarily complex logical operator that can be used to enable and disable other actions. When this stereotype is applied to behaviors, the behavior takes control values as inputs or provides them as outputs, that is, it treats control as data. When this stereotype is not applied, the behavior may not have a parameter typed by ControlValue. This stereotype also applies to operations with the same semantics.

19. ControlValue

The ControlValue enumeration is a type for treating control values as data and for UML control pins. It can be used as the type of behavior and operation parameters, object nodes, and attributes, and so on. The possible runtime values are given as enumeration literals. Modelers can extend the enumeration with additional literals, such as suspend, resume, with their own semantics.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
disable	Enumeration Literal	ControlValue	`$ControlValue[i].disable`
enable	Enumeration Literal	ControlValue	`$ControlValue[i].enable`

20. Copy

A Copy relationship is a dependency between a supplier requirement and a client requirement that specifies that the text of the client requirement is a read-only copy of the text of the supplier requirement.

Base Classifier

- trace

21. DeriveReqt

A DeriveReqt relationship is a dependency between two requirements in which a client requirement can be derived from the supplier requirement. As with other dependencies, the arrow direction points from the derived (client) requirement to the (supplier) requirement from which it is derived.

Base Classifier

- trace

22. designConstraint

Requirement that specifies a constraint on the implementation of the system or system part, such as the system must use a commercial off the shelf component.

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$designConstraint[i].Derived`
DerivedFrom	Requirement	Requirement	`$designConstraint[i].DerivedFrom`
Id	String	Requirement	`$designConstraint[i].Id`
Master	Requirement	Requirement	`$designConstraint[i].Master`
RefinedBy	NamedElement	Requirement	`$designConstraint[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$designConstraint[i].risk`
SatisfiedBy	NamedElement	Requirement	`$designConstraint[i].SatisfiedBy`
source	String	extendedRequirement	`$designConstraint[i].source`
Text	String	Requirement	`$designConstraint[i].Text`
TracedTo	NamedElement	Requirement	`$designConstraint[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$designConstraint[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$designConstraint[i].verifyMethod`

23. Diagram Description

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)

Completion status	String	Diagram Description	`$DiagramDescription[i].Completion status`
Description	String	Diagram Description	`$DiagramDescription[i].Description`
Reference	Element	Diagram Description	`$DiagramDescription[i].Reference`
Version	String	Diagram Description	`$DiagramDescription[i].Version`

24. diagramUsage

SysML also introduces the concept of a diagram usage. This represents a unique usage of a particular diagram type, such as a context diagram as a usage of an block definition diagram, internal block diagram, or use case diagram. The diagram usage can be identified in the header above the diagramKind as «diagramUsage».

25. Discrete

Discrete rate is a special case of rate of flow (see Rate) where the increment of time between items is non-zero.

Base Classifier

- Rate

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
rate	String	Rate	`$Discrete[i].rate`

26. DistributedProperty

DistributedProperty is a stereotype of Property used to apply a probability distribution to the values of the property. Specific distributions should be defined as subclasses of the DistributedProperty stereotype with the operands of the distributions represented by properties of those stereotype subclasses.

Base Classifier

- InvisibleStereotype

27. Domain

A Domain block represents an entity, a concept, a location, or a person from the real-world domain. A domain block is part of the system knowledge.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$Domain[i].isEncapsulated`

28. effbd

Enhanced Functional Flow Block Diagrams (EFFBD) are a widely-used systems engineering diagram, also called a behavior diagram. Most of its functionality is a constrained use of UML activities. EFFBD specifies that the activity conforms to the constraints necessary for EFFBD.

29. Environmental effect

An Environmental effect is an influence on the system from the environment without communicating with it directly. For example Temperature or Humidity.

30. Essential

31. extendedRequirement

A mix-in stereotype that contains generally useful attributes for requirements

Base Classifier

- Requirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$extendedRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$extendedRequirement[i].DerivedFrom`
Id	String	Requirement	`$extendedRequirement[i].Id`
Master	Requirement	Requirement	`$extendedRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$extendedRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$extendedRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$extendedRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$extendedRequirement[i].source`
Text	String	Requirement	`$extendedRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$extendedRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$extendedRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$extendedRequirement[i].verifyMethod`

32. External

An External block is a block that represents an actor. It facilitates a more detailed modeling of actors like ports or internal structure.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$External[i].isEncapsulated`

33. External system

An External system is a system that interacts with the system under development. For example an Information server or a Monitoring system.

34. FlowDirection

FlowDirection is an enumeration type that defines literals used for specifying input and output directions. FlowDirection is used by flow properties to indicate if a property is an input or an output with respect to its owner.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
in	Enumeration Literal	FlowDirection	`$FlowDirection[i].in`
inout	Enumeration Literal	FlowDirection	`$FlowDirection[i].inout`
out	Enumeration Literal	FlowDirection	`$FlowDirection[i].out`

35. FlowPort

A FlowPort is an interaction point through which input and/or output of items such as data, material, or energy may flow. This enables the owning block to declare which items it may exchange with its environment and the interaction points through which the exchange is made. We distinguish between atomic flow port and a nonatomic flow port. Atomic flow ports relay items that are classified by a single Block, ValueType, DataType, or Signal classifier. A nonatomic flow port relays items of several types as specified by a FlowSpecification. Flow ports and associated flow specifications define "what can flow" between the block and its environment, whereas item flows specify "what does flow" in a specific usage context. Flow ports relay items to their owning block or to a connector that connects them with their owner's internal parts (internal connector).

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
direction	FlowDirection	FlowPort	`$FlowPort[i].direction`
isAtomic	Boolean	FlowPort	`$FlowPort[i].isAtomic`
isConjugated	Boolean	FlowPort	`$FlowPort[i].isConjugated`

36. FlowProperty

A FlowProperty signifies a single flow element that can flow to/from a block. A flow property's values are either received from or transmitted to an external block. Flow properties are defined directly on blocks or flow specifications that are those specifications which type the flow ports. Flow properties enable item flows across connectors connecting parts of the corresponding block types, either directly (in case of the property is defined on the block) or via flowPorts. For Block, Data Type, and Value Type properties, setting an "out" FlowProperty value of a block usage on one end of a connector will result in assigning the same value of an "in" FlowProperty of a block usage at the other end of the connector, provided the flow properties are matched. Flow properties of type Signal imply sending and/or receiving of a signal usage. An "out" FlowProperty of type Signal means that the owning Block may broadcast the signal via connectors and an "in" FlowProperty means that the owning block is able to receive the Signal.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
direction	FlowDirection	FlowProperty	`$FlowProperty[i].direction`

37. FlowSpecification

A FlowSpecification specifies inputs and outputs as a set of flow properties. A flow specification is used by flow ports to specify what items can flow via the port.

38. functionalRequirement

Requirement that specifies an operation or behavior that a system, or part of a system, must perform.

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$functionalRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$functionalRequirement[i].DerivedFrom`
Id	String	Requirement	`$functionalRequirement[i].Id`
Master	Requirement	Requirement	`$functionalRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$functionalRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$functionalRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$functionalRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$functionalRequirement[i].source`
Text	String	Requirement	`$functionalRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$functionalRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$functionalRequirement[i].VerifiedBy`

verifyMethod	VerificationMethodKind	extendedRequirement	`$functionalRequirement[i].verifyMethod`

39. interfaceRequirement

Requirement that specifies the ports for connecting systems and system parts and the optionally may include the item flows across the connector and/or Interface constraints.

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$interfaceRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$interfaceRequirement[i].DerivedFrom`
Id	String	Requirement	`$interfaceRequirement[i].Id`
Master	Requirement	Requirement	`$interfaceRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$interfaceRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$interfaceRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$interfaceRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$interfaceRequirement[i].source`
Text	String	Requirement	`$interfaceRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$interfaceRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$interfaceRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$interfaceRequirement[i].verifyMethod`

40. Interval

Interval distribution - unknown probability between min and max

Base Classifier

- BasicInterval

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
max	Real	BasicInterval	`$Interval[i].max`
min	Real	BasicInterval	`$Interval[i].min`

41. ItemFlow

An ItemFlow describes the flow of items across a connector or an association. It may constrain the item exchange between blocks, block usages, or flow ports as specified by their flow properties. For example, a pump connected to a tank: the pump has an “out” flow property of type Liquid and the tank has an *"in" FlowProperty of type Liquid. To signify that only water flows between the pump and the tank, we can specify an ItemFlow of type Water on the connector.*

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
itemProperty	Property	ItemFlow	`$ItemFlow[i].itemProperty`

42. moe

A measure of effectiveness (moe) represents a parameter whose value is critical for achieving the desired mission cost effectiveness.

43. NestedConnectorEnd

The NestedConnectorEnd stereotype of UML ConnectorEnd extends a UML ConnectorEnd so that the connected property may be identified by a multi-level path of accessible properties from the block that owns the connector.

Base Classifier

- InvisibleStereotype

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
propertyPath	Property	NestedConnectorEnd	`$NestedConnectorEnd[i].propertyPath`

44. NoBuffer

When this stereotype is applied to object nodes, tokens arriving at the node are discarded if they are refused by outgoing edges, or refused by actions for object nodes that are input pins. This is typically used with fast or continuously flowing data values, to prevent buffer overrun, or to model transient values, such as electrical signals. For object nodes that are the target of continuous flows, «nobuffer» and «overwrite» have the same effect. The stereotype does not override UML token offering semantics; it just indicates what happens to the token when it is accepted. When the stereotype is not applied, the semantics are as in UML, specifically, tokens arriving at an object node that are refused by outgoing edges, or action for input pins, are held until they can leave the object node.

45. nonStreaming

Used for activities that accept inputs only when they start, and provide outputs only when they finish.

46. Normal

Normal distribution - constant probability between min and max

Base Classifier

- DistributedProperty

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
mean	Real	Normal	`$Normal[i].mean`
standardDerivation	Real	Normal	`$Normal[i].standardDerivation`

47. objectiveFunction

An objective function (aka optimization or cost function) is used to determine the overall value of an alternative in terms of weighted criteria and/or moe's.

48. Optional

When the «optional» stereotype is applied to parameters, the lower multiplicity must be equal to zero. This means the parameter is not required to have a value for the activity or any behavior to begin or end execution. Otherwise, the lower multiplicity must be greater than zero, which is called "required."

49. Overwrite

When the «overwrite» stereotype is applied to object nodes, a token arriving at a full object node replaces the ones already there (a full object node has as many tokens as allowed by its upper bound). This is typically used on an input pin with an upper bound of 1 to ensure that stale data is overridden at an input pin. For upper bounds greater than one, the token replaced is the one that would be the last to be selected according to the ordering kind for the node. For FIFO ordering, this is the most recently added token, for LIFO it is the least recently added token. A null token removes all the tokens already there. The number of tokens replaced is equal to the weight of the incoming edge, which defaults to 1. For object nodes that are the target of continuous flows, «overwrite» and «nobuffer» have the same effect. The stereotype does not override UML token offering semantics, just indicates what happens to the token when it is accepted. When the stereotype is not applied, the semantics is as in UML, specifically, tokens arriving at object nodes do not replace ones that are already there.

50. ParticipantProperty

The Block stereotype extends Class, so it can be applied to any specialization of Class, including Association Classes. These are informally called "association blocks." An association block can own properties and connectors, like any other block. Each instance of an association block can link together instances of the end classifiers of the association. To refer to linked objects and values of an instance of an association block, it is necessary for the modeler to specify which (participant) properties of the association block identify the instances being linked at which end of the association. The value of a participant property on an instance (link) of the association block is the value or object at the end of the link corresponding to this end of the association.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
end	Property	ParticipantProperty	`$ParticipantProperty[i].end`

51. performanceRequirement

Requirement that quantitatively measures the extent to which a system, or a system part, satisfies a required capability or condition.

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$performanceRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$performanceRequirement[i].DerivedFrom`
Id	String	Requirement	`$performanceRequirement[i].Id`
Master	Requirement	Requirement	`$performanceRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$performanceRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$performanceRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$performanceRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$performanceRequirement[i].source`
Text	String	Requirement	`$performanceRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$performanceRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$performanceRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$performanceRequirement[i].verifyMethod`

52. physicalRequirement

Requirement that specifies physical characteristics and/or physical constraints of the system, or a system part.

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$physicalRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$physicalRequirement[i].DerivedFrom`
Id	String	Requirement	`$physicalRequirement[i].Id`
Master	Requirement	Requirement	`$physicalRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$physicalRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$physicalRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$physicalRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$physicalRequirement[i].source`
Text	String	Requirement	`$physicalRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$physicalRequirement[i].TracedTo`

VerifiedBy	NamedElement	Requirement	`$physicalRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$physicalRequirement[i].verifyMethod`

53. Probability

When the «probability» stereotype is applied to edges coming out of decision nodes and object nodes, it provides an expression for the probability that the edge will be traversed. These must be between zero and one inclusive, and add up to one for edges with same source at the time the probabilities are used.
When the «probability» stereotype is applied to output parameter sets, it gives the probability the parameter set will be given values at runtime. These must be between zero and one inclusive, and add up to one for output parameter sets of the same behavior at the time the probabilities are used.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
probability	String	Probability	`$Probability[i].probability`

54. Problem

A Problem documents a deficiency, limitation, or failure of one or more model elements to satisfy a requirement or need, or other undesired outcome. It may be used to capture problems identified during analysis, design, verification, or manufacture and associate the problem with the relevant model elements. Problem is a stereotype of comment and may be attached to any other model element in the same manner as a comment.

55. PropertySpecificType

The PropertySpecificType stereotype should automatically be applied to the classifier which types a property with a propertyspecific type. This classifier can contain definitions of new or redefined features which extend the original classifier referenced by the property-specific type.

56. QuantityKind

A QuantityKind (formerly called 'Dimension') is a kind of quantity that may be stated by means of defined units. For example, the quantity kind of length may be measured by units of meters, kilometers, or feet. QuantityKind is defined as a stereotype of InstanceSpecification, but it uses this metaclass only to define supporting elements for ValueType definitions. (The reuse of InstanceSpecification to define another metaclass is similar to the EnumerationLiteral metaclass in UML.) The only valid use of a QuantityKind instance is to be referenced by the "quantityKind" property of a ValueType or Unit stereotype.

57. Rate

When the «rate» stereotype is applied to an activity edge, it specifies the expected value of the number of objects and values that traverse the edge per time interval, that is, the expected value rate at which they leave the source node and arrive at the target node. It does not refer to the rate at which a value changes over time. When the stereotype is applied to a parameter, the parameter must be streaming, and the stereotype gives the number of objects or *values that flow in or out of the parameter per time interval while the behavior or operation is executing. Streaming is a characteristic of UML behavior parameters that supports the input and output of items while a behavior is executing, rather than only when the behavior starts and stops. The flow may be continuous or discrete. The «rate» stereotype has a rate property of type InstanceSpecification. The values of this property must be instances of classifiers stereotyped by «valueType» or «distributionDefinition». In particular, the denominator for units used in the rate property must be time units.*

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
rate	String	Rate	`$Rate[i].rate`

58. Rationale

A Rationale documents the justification for decisions and the requirements, design, and other decisions. A Rationale can be attached to any model element including relationships. It allows the user, for example, to specify a rationale that may reference more detailed documentation such as a trade study or analysis report. Rationale is a stereotype of comment and may be attached to any other model element in the same manner as a comment.

59. Real

A Real value type represents the mathematical concept of a real number. A Real value type may be used to type values that hold continuous quantities, without committing a specific representation such as a floating point data type with restrictions on precision and scale.

60. Requirement

A requirement specifies a capability or condition that must (or should) be satisfied. A requirement may specify a function that a system must perform or a performance condition that a system must satisfy. Requirements are used to establish a contract between the customer (or other stakeholder) and those responsible for designing and implementing the system.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$Requirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$Requirement[i].DerivedFrom`
Id	String	Requirement	`$Requirement[i].Id`
Master	Requirement	Requirement	`$Requirement[i].Master`
RefinedBy	NamedElement	Requirement	`$Requirement[i].RefinedBy`
SatisfiedBy	NamedElement	Requirement	`$Requirement[i].SatisfiedBy`
Text	String	Requirement	`$Requirement[i].Text`
TracedTo	NamedElement	Requirement	`$Requirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$Requirement[i].VerifiedBy`

61. RequirementRelated

This stereotype is used to add properties to those elements that are related to requirements via the various dependencies.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Refines	Requirement	RequirementRelated	`$RequirementRelated[i].Refines`
Satisfies	Requirement	RequirementRelated	`$RequirementRelated[i].Satisfies`
TracedFrom	Requirement	RequirementRelated	`$RequirementRelated[i].TracedFrom`

62. RiskKind

1) High indicates an unacceptable level of risk,
2) Medium indicates an acceptable level of risk, and
3) Low indicates a minimal level of risk or no risk

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
High	Enumeration Literal	RiskKind	`$RiskKind[i].High`
Low	Enumeration Literal	RiskKind	`$RiskKind[i].Low`
Medium	Enumeration Literal	RiskKind	`$RiskKind[i].Medium`

63. Satisfy

A Satisfy relationship is a dependency between a requirement and a model element that fulfills the requirement. As with other dependencies, the arrow direction points from the satisfying (client) model element to the (supplier) requirement that is satisfied.

Base Classifier

- trace

64. Sensor

A Sensor is a special external system that forwards information from the environment to the system under development. For example a Temperature sensor.

Base Classifier

- External system

65. streaming

Used for activities that can accept inputs or provide outputs after they start and before they finish.

66. Subsystem

A Subsystem is a - typically large - encapsulated block within a larger system.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$Subsystem[i].isEncapsulated`

67. SwimLaneDiagram

Activity diagram usage with swim lanes.

68. System

A System is an artificial artifact consisting of blocks that pursue a common goal that cannot be achieved by the system's individual elements. A block can be software, hardware, a person, or an arbitrary unit.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$System[i].isEncapsulated`

69. System context

A System context element is a virtual container that includes the entire system and its actors.

Base Classifier

- Block

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
isEncapsulated	Boolean	Block	`$Systemcontext[i].isEncapsulated`

70. System process

71. TestCase

A test case is a method for verifying a requirement is satisfied.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Verifies	Requirement	TestCase	`$TestCase[i].Verifies`

72. Uniform

Uniform distribution - constant probability between min and max

Base Classifier

- BasicInterval

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
max	Real	BasicInterval	`$Uniform[i].max`
min	Real	BasicInterval	`$Uniform[i].min`

73. Unit

A Unit is a quantity in terms of which the magnitudes of other quantities that have the same quantity kind can be stated. A unit often relies on precise and reproducible ways to measure the unit. For example, a unit of length such as meter may be specified as a multiple of a particular wavelength of light. A unit may also specify less stable or precise ways to express some value, such as a cost expressed in some currency, or a severity rating measured by a numerical scale.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
quantityKind	QuantityKind	Unit	`$Unit[i].quantityKind`

74. usabilityRequirement

Base Classifier

- extendedRequirement

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Derived	Requirement	Requirement	`$usabilityRequirement[i].Derived`
DerivedFrom	Requirement	Requirement	`$usabilityRequirement[i].DerivedFrom`
Id	String	Requirement	`$usabilityRequirement[i].Id`
Master	Requirement	Requirement	`$usabilityRequirement[i].Master`
RefinedBy	NamedElement	Requirement	`$usabilityRequirement[i].RefinedBy`
risk	RiskKind	extendedRequirement	`$usabilityRequirement[i].risk`
SatisfiedBy	NamedElement	Requirement	`$usabilityRequirement[i].SatisfiedBy`
source	String	extendedRequirement	`$usabilityRequirement[i].source`
Text	String	Requirement	`$usabilityRequirement[i].Text`
TracedTo	NamedElement	Requirement	`$usabilityRequirement[i].TracedTo`
VerifiedBy	NamedElement	Requirement	`$usabilityRequirement[i].VerifiedBy`
verifyMethod	VerificationMethodKind	extendedRequirement	`$usabilityRequirement[i].verifyMethod`

75. User system

An User system is a special external system that serves as medium between a user and the system without having own interests in the communication. For example Input device or Display.

Base Classifier

- External system

76. ValueType

A ValueType defines types of values that may be used to express information about a system, but cannot be identified as the target of any reference. Since a value cannot be identified except by means of the value itself, each such value within a model is independent of any other, unless other forms of constraints are imposed. Value types may be used to type properties, operation parameters, or potentially other elements within SysML. SysML defines ValueType as a stereotype of UML DataType to establish a more neutral term for system values that may never be given a concrete data representation.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
quantityKind	QuantityKind	ValueType	`$ValueType[i].quantityKind`
unit	Unit	ValueType	`$ValueType[i].unit`

77. VerdictKind

Type of a return parameter of a TestCase must be VerdictKind, consistent with the UML Testing Profile.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
error	Enumeration Literal	VerdictKind	`$VerdictKind[i].error`
fail	Enumeration Literal	VerdictKind	`$VerdictKind[i].fail`
inconclusive	Enumeration Literal	VerdictKind	`$VerdictKind[i].inconclusive`
pass	Enumeration Literal	VerdictKind	`$VerdictKind[i].pass`

78. VerificationMethodKind

1) Analysis indicates that verification will be performed by technical evaluation using mathematical representations, charts, graphs, circuit diagrams, data reduction, or representative data. Analysis also includes the verification of requirements under conditions, which are simulated or modeled; where the results are derived from the analysis of the results produced by the model,
2) Demonstration indicates that verification will be performed by operation, movement or adjustment of the item under specific conditions to perform the design functions without recording of quantitative data. Demonstration is typically considered the least restrictive of the verification types,
3) Inspection indicates that verification will be performed by examination of the item, reviewing descriptive documentation, and comparing the appropriate characteristics with a predetermined standard to determine conformance to requirements without the use of special laboratory equipment or procedures, and
4) Test indicates that verification will be performed through systematic exercising of the applicable item under appropriate conditions with instrumentation to measure required parameters and the collection, analysis, and evaluation of quantitative data to show that measured parameters equal or exceed specified requirements.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
Analysis	Enumeration Literal	VerificationMethodKind	`$VerificationMethodKind[i].Analysis`
Demonstration	Enumeration Literal	VerificationMethodKind	`$VerificationMethodKind[i].Demonstration`
Inspection	Enumeration Literal	VerificationMethodKind	`$VerificationMethodKind[i].Inspection`
Test	Enumeration Literal	VerificationMethodKind	`$VerificationMethodKind[i].Test`

79. Verify

A Verify relationship is a dependency between a requirement and a test case or other model element that can determine whether a system fulfills the requirement. As with other dependencies, the arrow direction points from the (client) element to the (supplier) requirement.

Base Classifier

- trace

80. View

A View is a representation of a whole system or subsystem from the perspective of a single viewpoint. Views are allowed to import other elements including other packages and other views that conform to the viewpoint.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
viewPoint	Viewpoint	View	`$View[i].viewPoint`

81. Viewpoint

A Viewpoint is a specification of the conventions and rules for constructing and using a view for the purpose of addressing a set of stakeholder concerns. The languages and methods for specifying a view may reference languages and methods in another viewpoint. They specify the elements expected to be represented in the view, and may be formally or informally defined. For example, the security viewpoint may require the security requirements, security functional and physical architecture, and security test cases.

Attribute Name	Attribute Type	Attribute Owner	Sample Template Expression (VTL)
concerns	String	Viewpoint	`$Viewpoint[i].concerns`
languages	String	Viewpoint	`$Viewpoint[i].languages`
methods	String	Viewpoint	`$Viewpoint[i].methods`
purpose	String	Viewpoint	`$Viewpoint[i].purpose`
stakeholders	String	Viewpoint	`$Viewpoint[i].stakeholders`