About Reusable DocGen Methods

Table of Contents

List of Tables

List of Figures

1 Expression Library

The following image and table give an overview of the current expressions in the Cookbook Expression library. The purpose of the library is to provide users with reusable expressions and that can be used wherever in their own models. Users can simply use the expression as-is, by calling it by name, can add variations to the expression library, or they can also copy and paste and adapt it for their own uses.

The following views show these expressions being used. The expressions were called by name; this means that within the context of the viewpoint method, the user simply has to put functionName() into the appropriate location in the method. Each following subview shows this function call in the viewpoint.

Figure 1. Expression Library

The above diagram shows the layout and context for all the following subviews that contain examples of the expressions being used.

_Expression Library can be found in the OpenSE Library.

Expression Name	Expression Description	

cf_com	This expression automatically creates a <u>non-editable</u> cross reference to the *val* attribute of a given element.	`let cftype : String = 'com'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'true'`
		
		`in let result : String = '')`
		
		`in result`

cf_com_editable	This expression automatically creates a <u>editable</u> cross reference to the *val* attribute of a given element.	`let cftype : String = 'com'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'false'`
		
		`in let result : String = '')`
		
		`in result`

Expression Name	Expression Description	

cf_doc	This expression automatically creates a <u>non-editable</u> cross reference to the *documentation* attribute of a given element.	`let cftype : String = 'doc'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'true'`
		
		`in let result : String = '')`
		
		`in result`

cf_doc_editable	This expression automatically creates an <u>editable</u> cross reference to the *documentation* attribute of a given element.	`let cftype : String = 'doc'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'false'`
		
		`in let result : String = '')`
		
		`in result`

cf_img	This expression automatically creates a <u>non-editable</u> cross reference to the *val* attribute of a given element.	`let cftype : String = 'img'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'true'`
		
		`in let result : String = '')`
		
		`in result`

Expression Name	**Expression Description**	

cf_img_editable	This expression automatically creates a <u>editable</u> cross reference to the *val* attribute of a given element.	`let cftype : String = 'img'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'false'`
		
		`in let result : String = '')`
		
		`in result`

cf_name	This expression automatically creates a <u>non-editable</u> cross reference to the *name* attribute of a given element.	`let cftype : String = 'name'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'true'`
		
		`in let result : String = '')`
		
		`in result`

cf_name_editable	This expression automatically creates a <u>editable</u> cross reference to the *name* attribute of a given element.	`let cftype : String = 'name'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'false'`
		
		`in let result : String = '')`
		
		`in result`

Expression Name	Expression Description	

cf_val	This expression automatically creates a non-editable cross reference to the *val* attribute of a given element.	`let cftype : String = 'val'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'true'`
		
		`in let result : String = '')`
		
		`in result`

cf_val_editable	This expression automatically creates a editable cross reference to the *val* attribute of a given element.	`let cftype : String = 'val'`
		`in let eid : String = self.ID`
		`in let title : String = self.name`
		`in let display : String = '[cf:'`
		`        .concat(title)`
		`        .concat('.')`
		`        .concat(cftype)`
		`in let editBool : String = 'false'`
		
		`in let result : String = '')`
		
		`in result`

CollectClassifiers	This expression operates on a classifier (e.g. class, actor, component, etc.) and returns all its "base classifiers" recursively - that is, all the elements which this element specializes, as well as the elements which those general elements specialize, recursively.	`Set{self.oclAsType(Element)}->closure(x | x.oclAsType(Classifi`

Expression Name	Expression Description	
CollectSpecializations()	This OCL function acts on a list of classifiers (or a single one) and returns all the specializations for that element recursively. For example, in the UML metamodel, if you selected Dependency, you get Abstraction, Usage, and Deployment which are direct specializations of Dependency, as well as all the specializations of those elements, including Substitution, Component Realization, etc.	`self.oclAsType(Classifier)->closure(r('Generalization').oclAsT`
CollectViewsFromDoc()	This expression acts on a Document and returns all of its subviews.	`self.oclAsType(Class)->closure(part.oclAsType(Property).type.c`
GetAssociationSources()	This expression acts on an element and gathers the sources of its associations.	`self.r('Association').oclAsType(Association).ownedEnd.type->ex`

GetCommentDocumentation()	This expression acts on an element and returns its comment stored in the documentation.	`if`
		`_commentOfAnnotatedElement->intersection(ownedComment)->any(tr`
		`then _commentOfAnnotatedElement->intersection(ownedComment)->a`
		`else`
		`''`
		`endif`

Expression Name	Expression Description	
GetDocsFromView()	This expression acts on a view and gathers all the documents in which it is used, both shared and composite relations.	`self.oclAsType(Element)->closure(r('Association').oclAsType(As`

GetMMSSite	This expression acts on any element and returns the corresponding MMS *site* that holds its information.	`if`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`then`
		`if self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelMan`
		`and`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`then`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`else`
		`'cae'.log('ERROR: No site found, using "cae" by default')`
		`endif`
		`else`
		`'cae'.log('ERROR: «ModelManagementSystem» is not applied to an`
		`endif`

GetMMSURL	This expression acts on any element and returns the corresponding MMS *URL* that holds its information.	`if`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`then`
		`if`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`and`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`then`
		`self.o()->flatten()->any(x|x.oclAsType(Element).s('ModelManage`
		`else`
		`'https://cae-ems.jpl.nasa.gov'.log('ERROR: No URL Found, using`
		`endif`
		`else`
		`'https://cae-ems.jpl.nasa.gov'.log('ERROR: «ModelManagementSys`
		`endif`

GetParts()	This expression acts on any class element (such as Block) and returns its owned part properties.	`oclAsType(Class).part->select(x|x.s('PartProperty')->size() =`
GetPortDirection()	This expression acts on a port and returns the port direction.	`oclAsType(Port).type.oclAsType(Class).feature->any(x|x.s('Flow`

Expression Name	Expression Description	
GetPorts()	This expression acts on any element and returns its ports.	`oclAsType(Element).ownedElement->select(x|x.oclIsKindOf(Port))`

GetUnits	This expression works on a property - it returns a string representation of the "unit" of the property. Ideally, the units are stored as the property's value's type, but if the property's value has not type, it can search using the property's type. First it looks to see if the value's type or property's type has a unit identified, if it does, it looks for the "symbol" property of the unit and returns that. If there's no "symbol" property, it returns the name of the unit. If there's no unit identified, it returns the name of the property's type or the value's type. This will fail for properties with no type and for properties whose type is not a quantity kind.	`if`
		`oclAsType(Property).defaultValue->exists(z|z.oclAsType(TypedEl`
		`then`
		`if`
		`oclAsType(Property).defaultValue.oclAsType(TypedElement).type.`
		`then`
		`if`
		`oclAsType(Property).defaultValue.oclAsType(TypedElement).type.`
		`then`
		`oclAsType(Property).defaultValue.oclAsType(TypedElement).type.`
		`else`
		`oclAsType(Property).defaultValue.oclAsType(TypedElement).type.`
		`endif`
		`else`
		`oclAsType(Property).defaultValue.oclAsType(TypedElement).type.`
		`endif`
		`else`
		`if`
		`oclAsType(Property).type.appliedStereotypeInstance.slot->any(x`
		`then`
		`if`
		`oclAsType(Property).type.appliedStereotypeInstance.slot->any(x`
		`then`
		`oclAsType(Property).type.appliedStereotypeInstance.slot->any(x`
		`else`
		`oclAsType(Property).type.appliedStereotypeInstance.slot->any(x`
		`endif`
		`else`
		`oclAsType(Property).type.oclAsType(NamedElement).name`
		`endif`
		`endif`

Expression Name	Expression Description	

GetURLFromView()	This expression acts on a view and returns its MMS URL/site as a concatenated URL string.
	Be sure to replace "Concept Description Documents" with the desired document name.
		`if self.s('View')->size() + self.s('view')->size() > 0`
		`then`
		`if`
		`self.GetDocsFromView()->flatten()->any(y|y.oclAsType(Element)-`
		`then`
		`''.concat(self.GetMMSURL()->flatten()->any(true).oclAsType(Str`
		`else`
		`''.concat(self.GetMMSURL()->flatten()->any(true).oclAsType(Str`
		`endif`
		`else`
		`if`
		`self.s('Document')->size() = 1`
		`then`
		`''.concat(self.GetMMSURL()->flatten()->any(true).oclAsType(Str`
		`else`
		`null`
		`endif`
		`endif`

GetViewID()	This expression acts on a View and returns the View ID.	`self.r('Expose').oclAsType(Dependency).source->any(x|x.oclAsTy`

Realify	This expression acts on an element of kind "LiteralString" and coverts it to "Real"	`if`
		`oclIsKindOf(LiteralString)`
		`then`
		`v().oclAsType(String).toReal().oclAsType(Real)`
		`else`
		`if`
		`oclIsKindOf(LiteralReal)`
		`then`
		`v().oclAsType(Real)`
		`else`
		`self`
		`endif`
		`endif`

RealifyNoLit()	This expression acts on an element of kind "String" (not "LiteralString") and converts it to "Real".	`if`
		`x.oclIsKindOf(String)`
		`then`
		`x.oclAsType(String).toReal()`
		`else`
		`x`
		`endif`

Expression Name	Expression Description	
Stringify	This expression acts on an element of kind "LiteralString"or "LiteralReal" and coverts it to "String".	`if oclIsKindOf(LiteralString) then v().oclAsType(String) else if oclIsKindOf(LiteralReal) then v().oclAsType(Real).toString().oclAsType(String) else self endif endif`

1.1 Cross References Expression Library

The Cross References Expression Library contains examples of how to execute creating cross references for element attributes.

Figure 2. Animals

Figure 3. Using Cross Reference Expressions VP

This method shows the different cross reference expressions being used to build a table of cross references.

Table 2. <>

Cf_Name_NonEditable	Cf_Name_Editable	Cf_Doc_NonEditable	Cf_Doc_Editable	Cf_Val_NonEditable	Cf_Val_Editable
[cf:Cat.name]	[cf:Cat.name]	[cf:Cat.doc]	[cf:Cat.doc]	[cf:noise.val] [cf:ocl.val]	[cf:noise.val] [cf:ocl.val]
[cf:Duck.name]	[cf:Duck.name]	[cf:Duck.doc]	[cf:Duck.doc]	[cf:noise.val] [cf:ocl.val]	[cf:noise.val] [cf:ocl.val]
[cf:Cow.name]	[cf:Cow.name]	[cf:Cow.doc]	[cf:Cow.doc]	[cf:noise.val] [cf:ocl.val]	[cf:noise.val] [cf:ocl.val]
[cf:Frog.name]	[cf:Frog.name]	[cf:Frog.doc]	[cf:Frog.doc]	[cf:noise.val] [cf:ocl.val]	[cf:noise.val] [cf:ocl.val]
[cf:Dog.name]	[cf:Dog.name]	[cf:Dog.doc]	[cf:Dog.doc]	[cf:noise.val] [cf:ocl.val]	[cf:noise.val] [cf:ocl.val]

1.2 Collect Expression Library

The Collect Expression Library contains examples of how to execute identified functions.

The first table shows how "CollectClassifiers" and "CollectSpecializations" can be used on elements that have been generalized to show the base classifiers and the specializations of exposed element. These relationships can be seen in cf name([cf:Specialized Animals BDD.name]) does not exist .

The second table shows how "CollectViewsFromDoc" can be used to find all the subviews of an exposed document.

Figure 4. Specialized Animals Inheritance

Figure 5. Using Collection Functions Expressions VP

Table 3. Classifiers and Specializations

Name	CollectClassifiers	CollectSpecializations
Wild Cat	Cat	Lion Jaguar
House Dog	Dog	Husky Golden Retriever
House Cat	Cat	Siamese Cat Main Coon
Wild Dog	Dog	Wolf Dog African Wild Dog
Siamese Cat	House Cat Cat	
Main Coon	House Cat Cat	
Lion	Cat Wild Cat	
Jaguar	Cat Wild Cat	
Wolf Dog	Wild Dog Dog	
African Wild Dog	Wild Dog Dog	
Golden Retriever	Dog House Dog	
Husky	Dog House Dog	

1.3 GET Functions Expression Library

The GET Functions Expression Library contains examples of how to execute GET functions for specific exposed elements.

The table applies "GetAssociationSources", "GetDocsFromView", GetComments", "GetMMSSite", and "GetMMSUrl" to the children views of Expression Library

Figure 6. Using GET Functions Expressions VP

Table 5. GET Functions Expressions

Name	GetAssociationSources	GetDocsFromView()	GetComments	GetMMSSite	GetMMSURL
Cross References Expression Library	Expression Library	About Reusable DocGen Methods	The Cross References Expression Library contains examples of how to execute creating cross references for element attributes.	jpl-common-sysml-model-library	https://opencae.jpl.nasa.gov

Name	GetAssociationSources	GetDocsFromView()	GetComments	GetMMSSite	GetMMSURL
GET Functions Expression Library	Expression Library	About Reusable DocGen Methods	The GET Functions Expression Library contains examples of how to execute GET functions for specific exposed elements. The table applies "GetAssociationSources", "GetDocsFromView", GetComments", "GetMMSSite", and "GetMMSUrl" to the children views of Expression Library	jpl-common-sysml-model-library	https://opencae.jpl.nasa.gov
Collect Expression Library	Expression Library	About Reusable DocGen Methods	The Collect Expression Library contains examples of how to execute identified functions. The first table shows how "CollectClassifiers" and "CollectSpecializations" can be used on elements that have been generalized to show the base classifiers and the specializations of exposed element. These relationships can be seen in cf name([cf:Specialized Animals BDD.name]) does not exist . The second table shows how "CollectViewsFromDoc" can be used to find all the subviews of an exposed document.	jpl-common-sysml-model-library	https://opencae.jpl.nasa.gov
Typing Functions Expression Library	Expression Library	About Reusable DocGen Methods	The Cross References Expression Library contains examples of how to execute "typing" expressions to convert LiteralReal, LiteralString, String, and Real types into a wanted state. In this case, the "danger level" ValueProperty was used for demonstration.	jpl-common-sysml-model-library	https://opencae.jpl.nasa.gov

1.4 Typing Functions Expression Library

The Cross References Expression Library contains examples of how to execute "typing" expressions to convert LiteralReal, LiteralString, String, and Real types into a wanted state. In this case, the "danger level" ValueProperty was used for demonstration.

Figure 7. Specialized Animals Inheritance

Table 6. Typing Functions

Realify	RealifyNoLit	Stringify
5.0	5	5
7.5	7.5	7.5
5.0	5.0	5.0
9.0	9.0	9.0