

WRITING QUERIES IN SYSML V2



3DEXPERIENCE®

Sean Densford

Instructor



Sean Densford

CATIA Cyber Systems Industry Process Consultant

- ▶ MS in Systems Engineering from Johns Hopkins University
- ▶ 10 years in Electrical Engineering and Systems Engineering
- ▶ Certified professional: CSEP and OCSMP SysML V1 & V2

Tutorial Summary – Writing Queries in SysML V2

Tutorial Setup:

- ▶ 4 hour - Hands-on (75%) with lecture (25%)
- ▶ Users will have access to CATIA Magic 26xR1 OR can use the tool of their choice (I will do my best to help anyone in any tool)

Learning Objectives:

- ▷ Understand the basics of why it is useful to parse the model
- ▷ Understand the basics of SysML V2 and KerML Functions
- ▷ Understand how to count elements
- ▷ Understand how to filter elements
- ▷ Understand how to build custom table

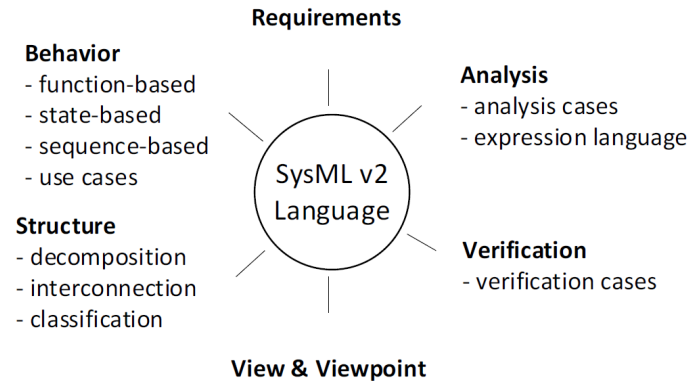
SYSTEMS MODELING LANGUAGE - BASIC INTRODUCTION



3DEXPERIENCE®

SysML – BASIC INFORMATION

- ▶ OMG Systems Modeling Language (SysML) is a graphical and textual modeling language for **specification, architecture, high-level design, analysis, verification & validation** of systems
- ▶ Originally developed by OMG and INCOSE in May 2006
- ▶ Version 2 released 2025
- ▶ Built on KerML (Kernel Modeling Language)
- ▶ SysML is a **key enabler of MBSE** for modeling large and complex systems



Key Improvements – SysML V2

Based on KerML – Kernel Modeling Language

- A modeling language for modeling languages
- No longer based on UML (Software-Centric)

Textual Notation

Standard API

- Easier to connect to other tools

Occurrences

- Time Slice and Snapshot
- 4D Modeling

Visualizations are based on View and Viewpoint

Expression Language for Analysis

More consistent implementation of Definition and Usage

- Definition is no longer required for Usage
- Easier to do quick “back of the napkin” modeling

TERMINOLOGY/CONCEPT CONSISTENCY

SysML v2	SysML v1
part / part def	part property / block
attribute / attribute def	value property / value type
port / port def	proxy port / interface block
action / action def	action / activity
state / state def	state / state machine
constraint / constraint def	constraint property / constraint block
requirement / requirement def	requirement
connection / connection def	connector / association block
view / view def	view

Part Concept

- ▶ A part can represent any level of abstraction, such as a purely logical component without implementation constraints, or a physical component with a part number, or some intermediate abstraction.
- ▶ Parts can also be used to represent different kinds of system components such as hardware components, software components, facilities, organizations, or users of a system.
- ▶ A system is modeled as a composite part, and its parts may themselves have further composite structure.

DEFINITION VS USAGE

► Definition

- ▷ A classifier that provides the blueprint of a usage element.
- ▷ Ex: part def Vehicle

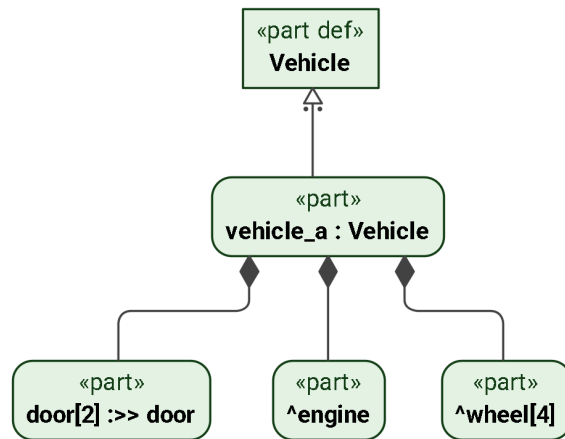
```
//Definition
part def Vehicle {
  part wheel [4];
  part door [2..4];
  part engine;
}

//Usage
part vehicle_a : Vehicle {
  part door [2] :>> door;
}
```

Textual Notation

► Usage

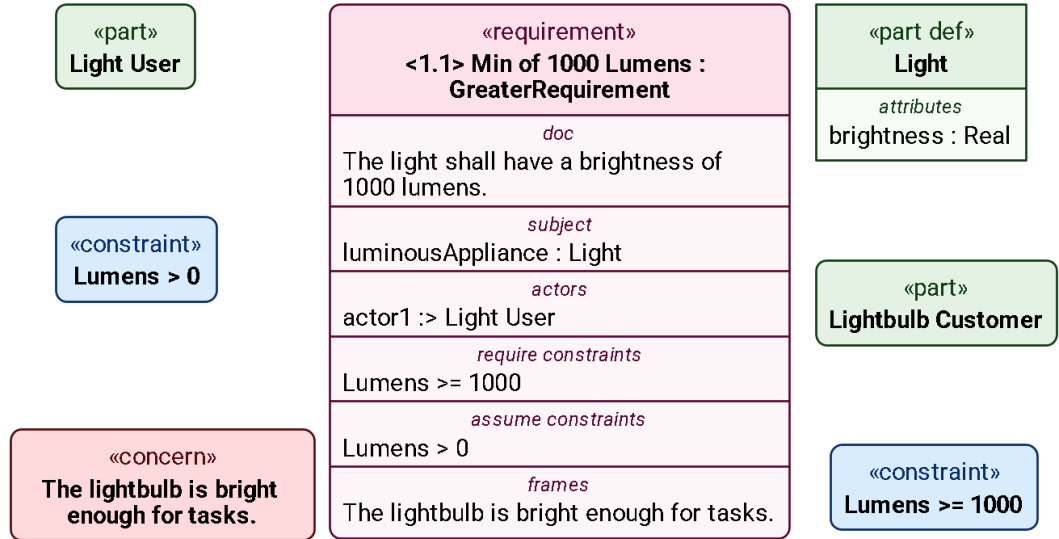
- ▷ An instance of a defined element
- ▷ Ex: part vehicle_a
 - Inherits wheel, door, and engine from Vehicle part def
 - Redefines door to be multiplicity 2 instead of 2 to 4.



Graphical Notation

Requirement Features

- ▶ In SysML v2, there are no special requirement types for capturing Stakeholder Needs. We will use the standard requirement type
- ▶ Requirements now have multiple features including:
 - ▷ Documentation
 - ▷ Subject
 - ▷ Actors
 - ▷ Stakeholders
 - ▷ Assume and Require Constraints
 - ▷ FramedConcern (considered a require constraint of its owning requirement)



Example Requirement with All Features

RELATIONSHIP TYPES (CONT.)

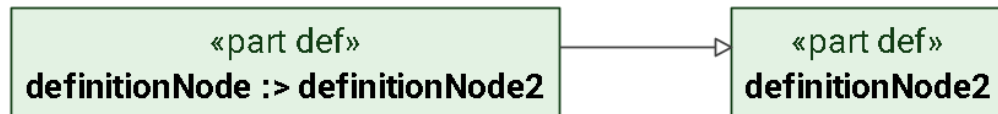
DEFINED BY = ':' | 'defined' 'by'

feature typing



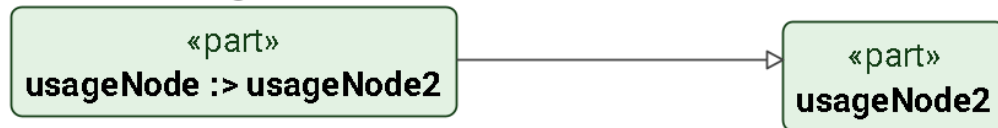
SPECIALIZES = ':>' |
'specializes' AKA
Subclassification

subclassification



SUBSETS = ':>' | 'subsets'

subsetting



Terminology Spoken from Left to Right: "Usage is Defined By Definition"

RELATIONSHIP TYPES

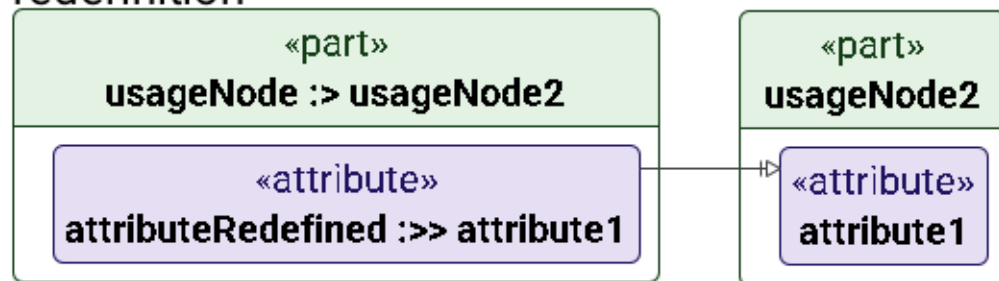
REFERENCES = '::<>' | 'references'

reference subsetting



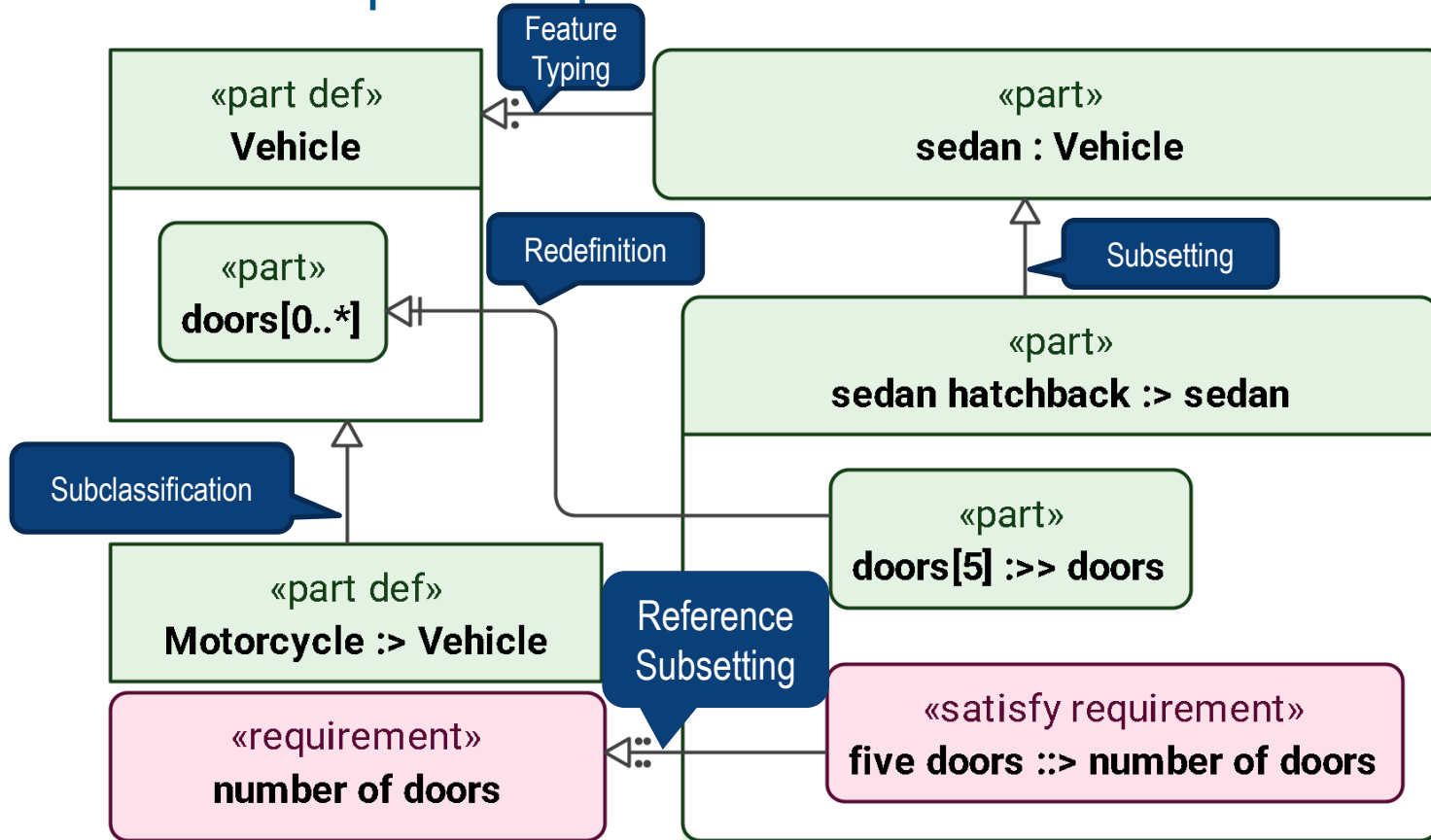
REDEFINES = '::~>' | 'redefines'

redefinition



Terminology Spoken from Left to Right: "Usage is Defined By Definition"

Relationship Examples



BEHAVIOR MODELING

What is Behavior Modeling?

- ▶ Describes how a system will behave under various conditions

Why do We Model Behavior?

- ▶ To compare simulated/expected behavior to requirements

How do We Model Behavior?

- ▶ Exchanges between users and the system showing how the system is used to achieve specific goals – **Use Case**
- ▶ Flow of activities within a system – **Action Flow**
- ▶ System states and the transition between based on events – **State Machine**
- ▶ Interactions between different participants or components of the system over time – **Sequence View**

SYSML V2 TEXTUAL NOTATION FOR QUERYING



3DEXPERIENCE®

TEXTUAL NOTATION IN SYSML V2

SysML Libraries:

► **ScalarQuantityValue Operations**

- ▷ Basic Math: Addition, Subtraction, Multiplication, Division, and Exponents
- ▷ Basic Functions: Abs, Max, Min, Sqrt, Floor, Sum, and Product
- ▷ Basic Logic: ==, !=, <, >, and <=

► **VectorQuantityValue Operations**

- ▷ Same as above with more
- ▷ Inner, Outer, Norm, Angle, Transform

► **TensorQuantityValue Operations**

Great Start for Calculations, but Not Enough to Query the Model

TEXTUAL NOTATION IN SYSML V2

KerML Libraries:

► Function Library

▷ Base Functions

- ==, !=, ===, ToString, all, istype, hastype, meta

▷ Math Functions

- Trig, Natural, Integer, Rational, Real

▷ Sequence Functions

- Equals, Size, Includes, Excludes, Intersection

▷ Collection Functions

- Same as above + Contains

▷ Control Functions

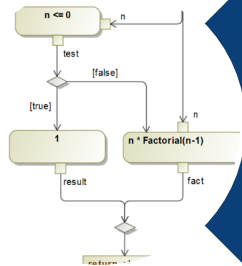
- Collect, Implies, Select, Reduce

Now We Have Everything We Need to Query

Metric Methods – Which Element Should We Use?

Large Calculations
private import Scal
calc def Area {
 in length : Real
 in width : Real;
 return : Real;
 length * width

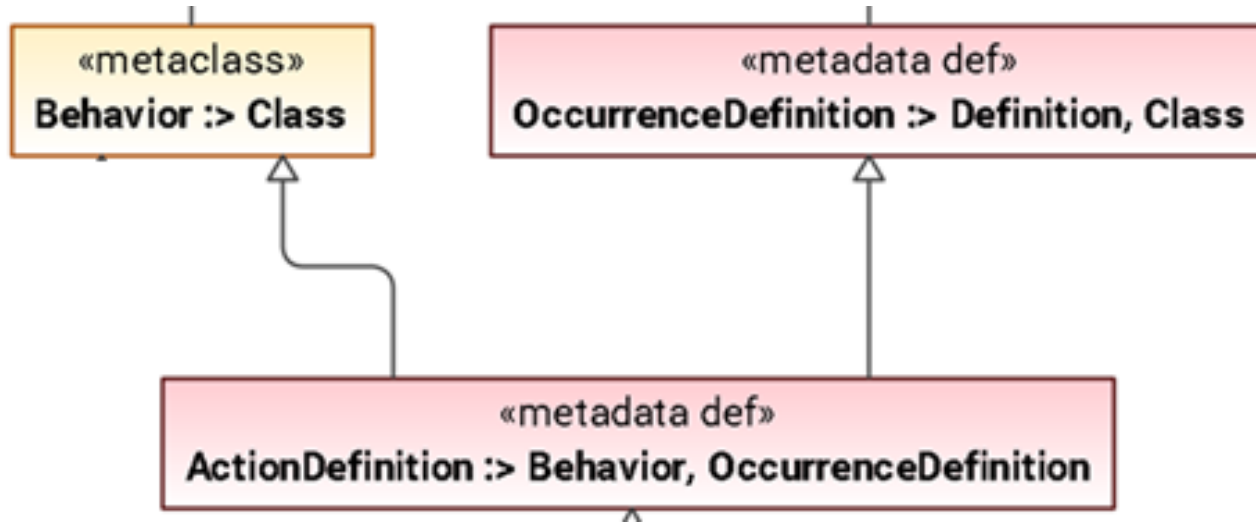
Calc Definitions



Opaque Actions

Opaque Action

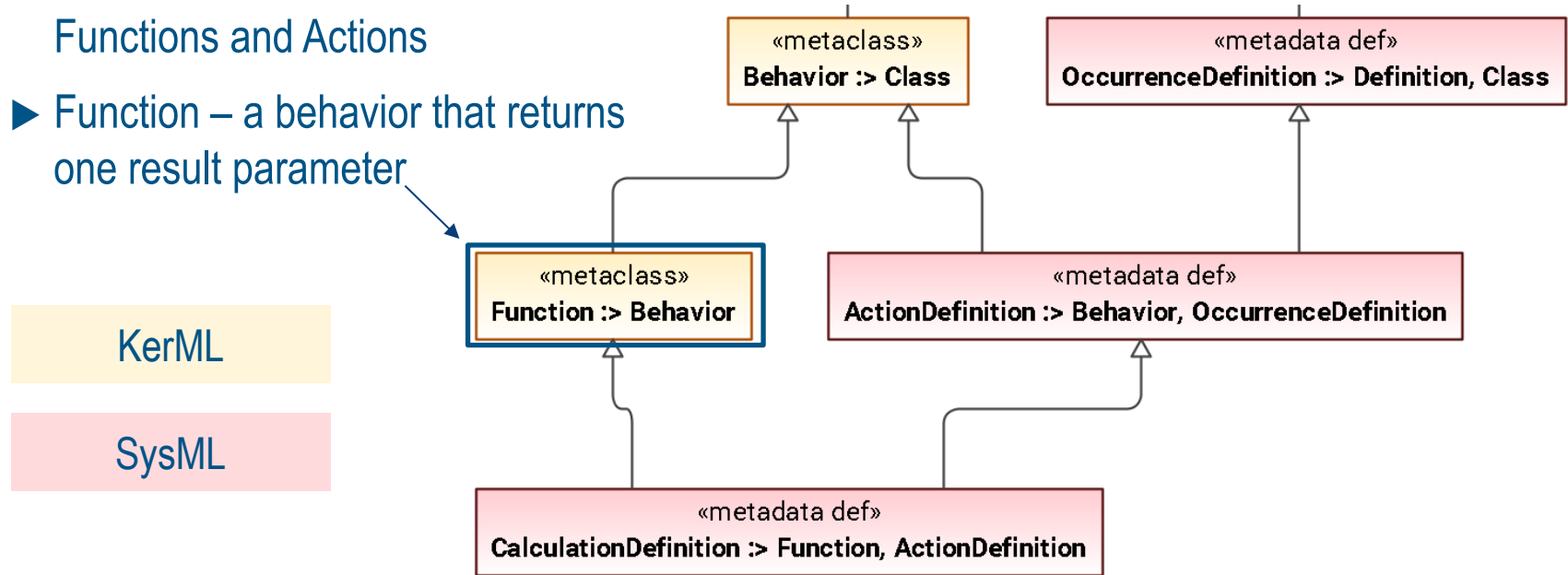
- Action Def/Usage using another language than sysml



More Suitable to Represent Behaviors

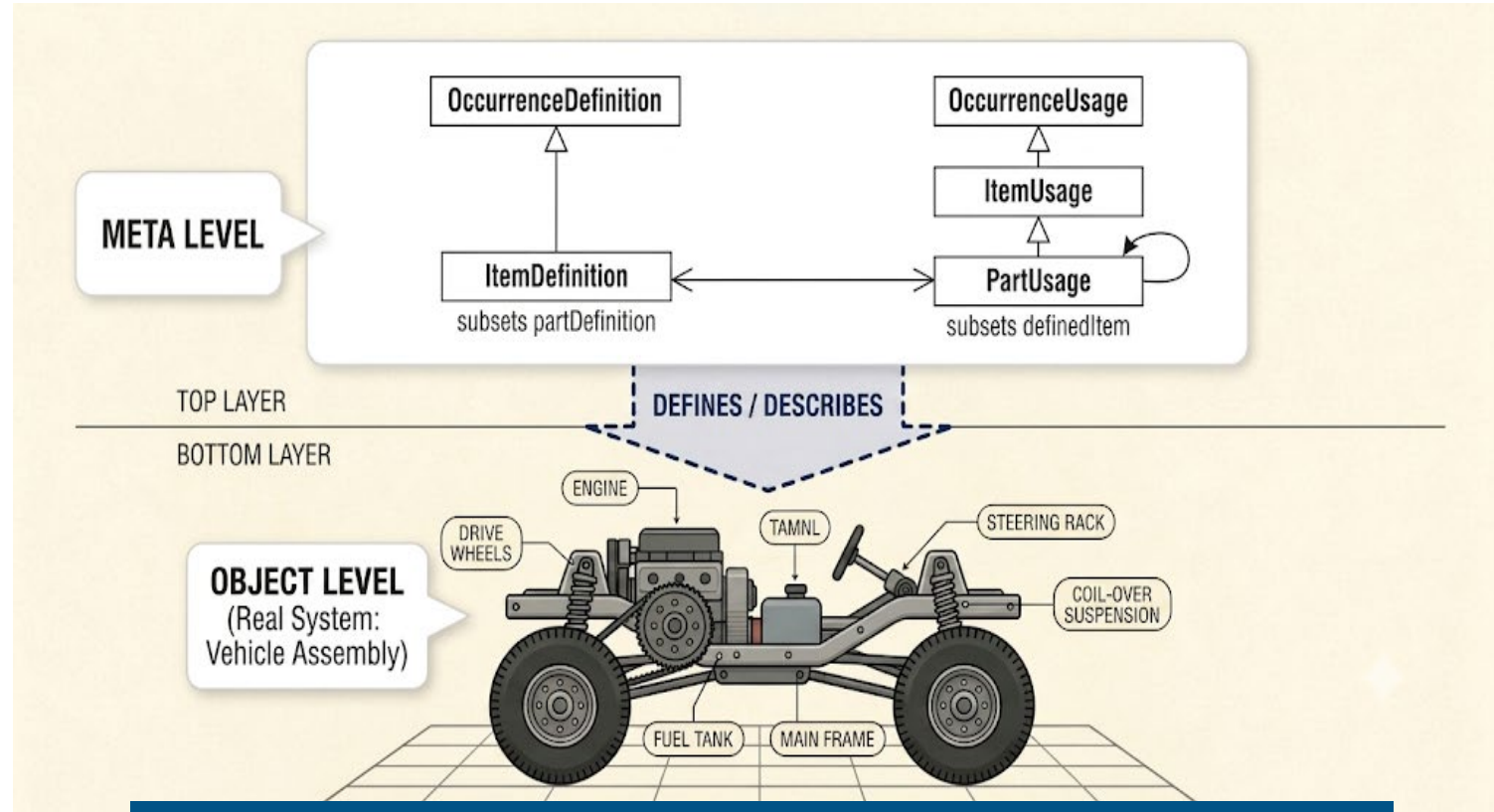
Calculation Definition - Functions

- Calculation Definitions are Functions and Actions
- Function – a behavior that returns one result parameter



Could Represent Math or Behavior

Meta Level Vs Object Level Casting



Why Cast? To Access Different Properties of an Element

OPERATOR MAPPING – IN CLASS EXAMPLES

Table 5. Operator Mapping

Operator	Library Function	Description	Model-Level Evaluable?
all	BaseFunctions::'all'	Type extent	No
istype	BaseFunctions::'istype'	All argument values are directly or indirectly instances of a type	Yes
hastype	BaseFunctions::'hastype'	All argument values are directly instances of a type	Yes
@	BaseFunctions::'@'	Any argument value is directly or indirectly an instance of a type	Yes
@@	BaseFunctions::'@@'	Any argument value is directly or indirectly an instance of a metaclass	Yes

Is/Hastype eval to true on a null (empty list)

SysML Name Rules

► Basic Name

- ▷ Initial character
 - ▶ Must be Lowercase, Uppercase, or an Underscore
- ▷ Remaining characters
 - ▶ May be any of the above or any digit
- ▷ Reserved words may not be used as a name see spec 8.2.2.1.2 for a list

► Unrestricted Name

- ▷ Must be non-empty sequence of characters surrounded by single quotes
- ▷ Backspace, tab, or newline characters must be escaped
- ▷ Escape character is a backslash

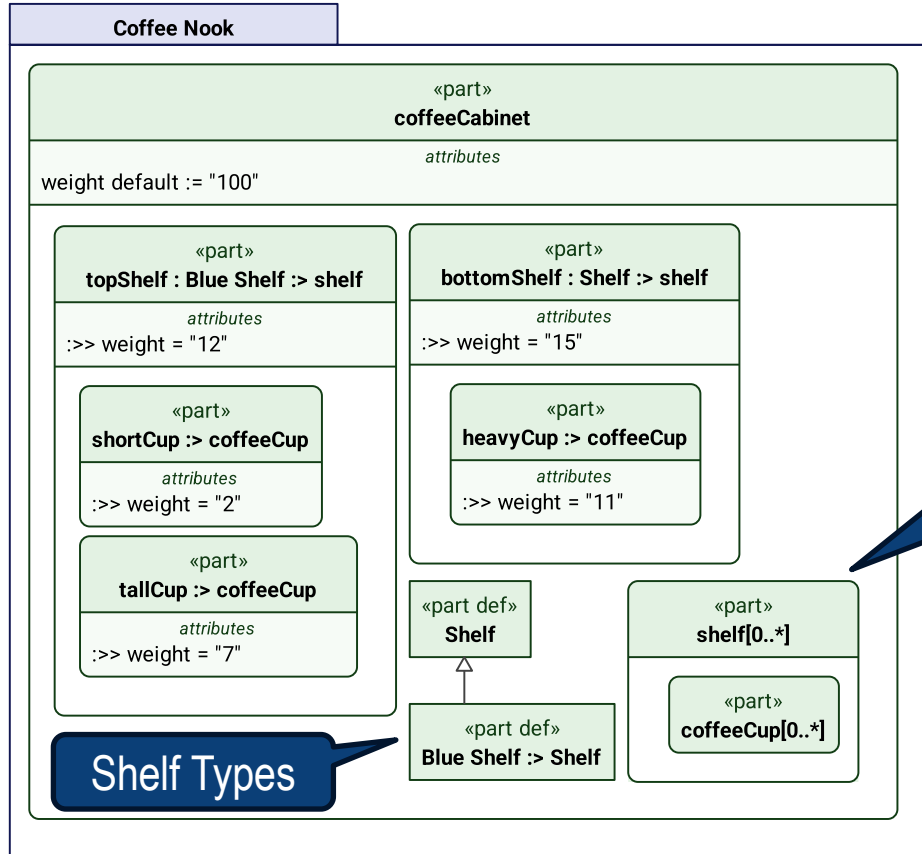
Examples:

Plane
power_120V
_power120V

Examples:

'-1'
'Power Adapter'
'On/Off Switch'
'Ångström'

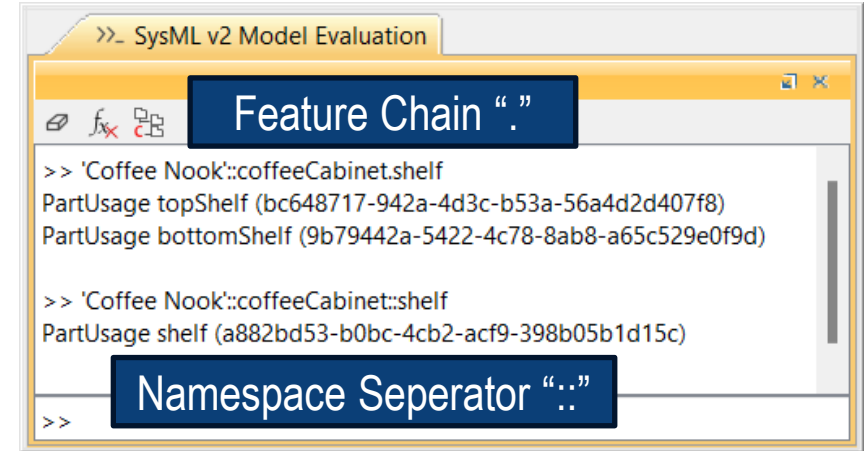
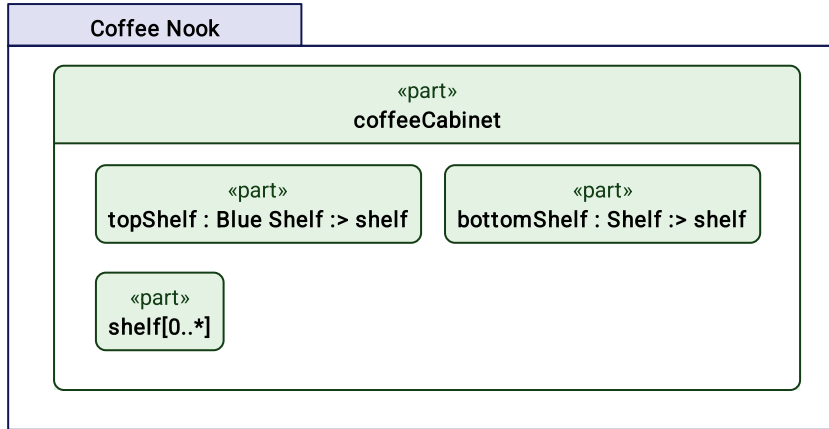
Sample Architecture – Coffee Nook



Coffee Nook contains a coffeeCabinet, a top and bottom shelf, and some coffee mugs all with their own weights

coffeeCabinet has some generic shelf and coffeeCup used to subset its shelves. Could be stored in a library

NAMESPACE SEPERATOR "::" VS FEATURE CHAIN "."



'Coffee Nook'::coffeeCabinet.shelf

'Coffee Nook'::coffeeCabinet::shelf

"::" will get contained elements, "." will iterate through them

MULTI-STEP QUERY IN V1

Goal: Get all the coffeeCup weights in the cabinet

ibd [Block] Coffee Cabinet[Coffee Cabinet]

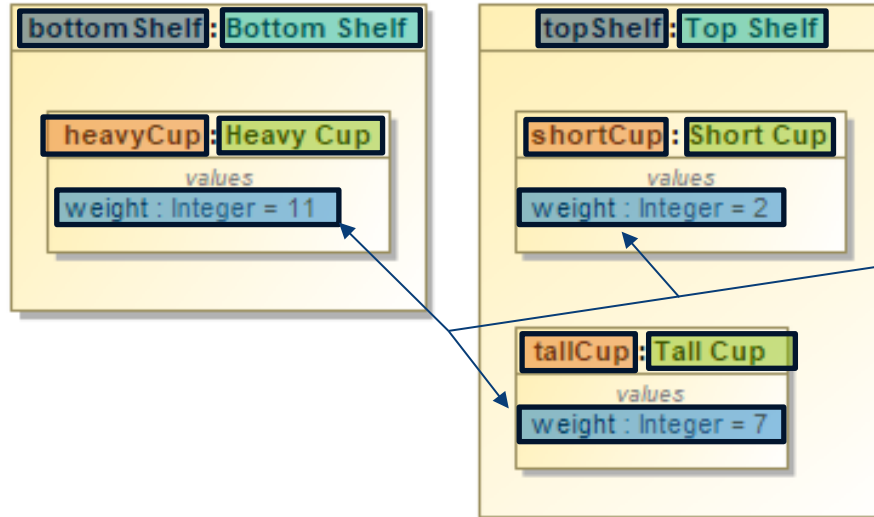


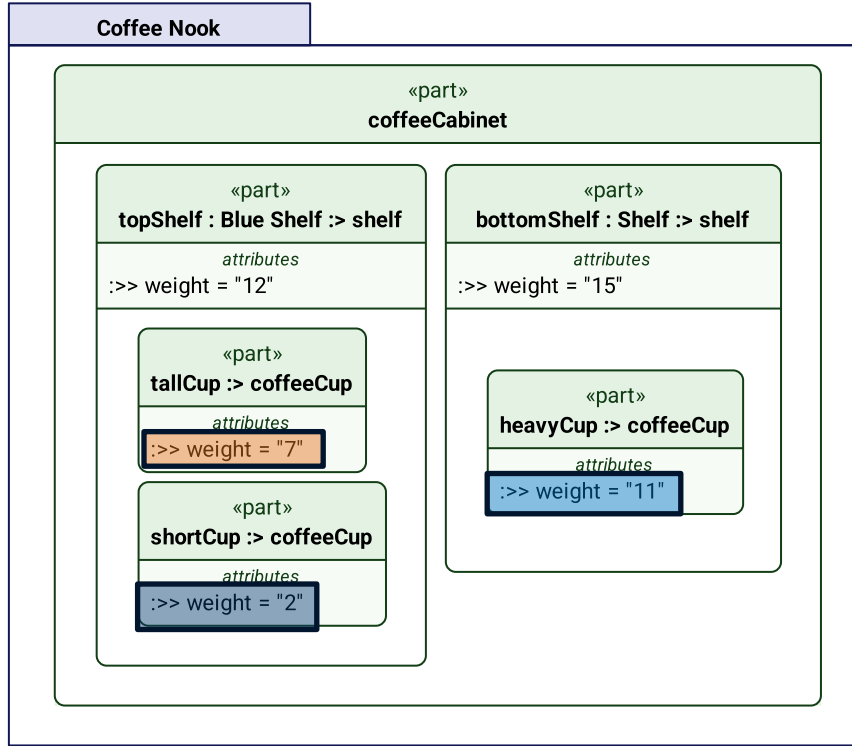
Table [Model] Model[Coffee Cabinet Table]

#	Name	Nested Owned Elements	Nested Owned Elements Values
1	P bottomShelf	P heavyCup : Heavy Cup	11
2	P topShelf	P tallCup : Tall Cup P shortCup : Short Cup	7 2

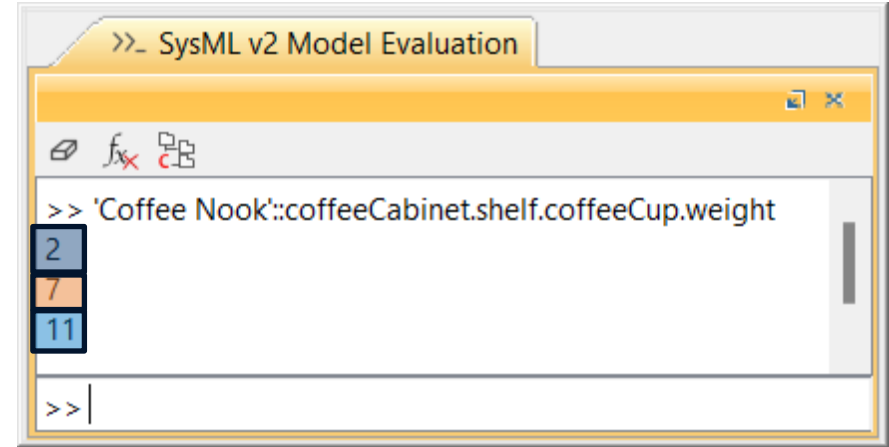
Metaclass or Stereotype	Property
P PartProperty [Property]	Type
Block [Class]	Owned Element
P PartProperty [Property]	Type
Block [Class]	Owned Element
V ValueProperty [Property]	Default Value

In V1, many steps are required to get these values

MULTI-STEP QUERY IN V2



Goal: Get all the coffeeCup weights in the cabinet



`'Coffee Nook'::coffeeCabinet.shelf.coffeeCup.weight`

Example Query:

Using Feature Chain We can Iterate All coffeeCups and Get Their Weight

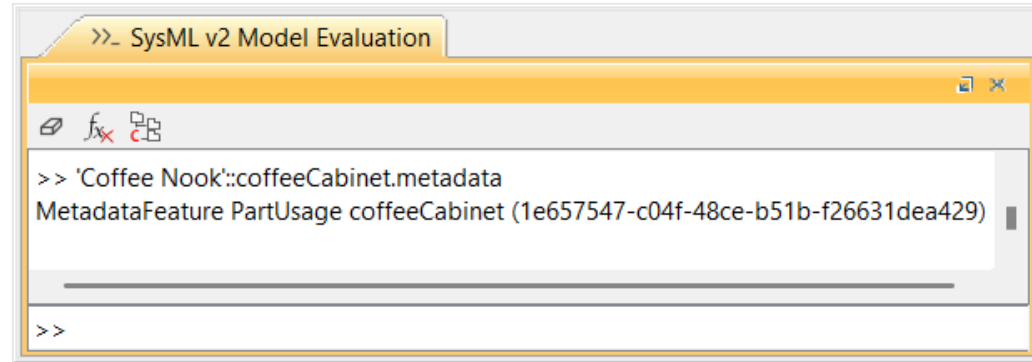
METADATA ACCESS EXPRESSION: “.METADATA” - META LEVEL

Coffee Nook

«part»

coffeeCabinet

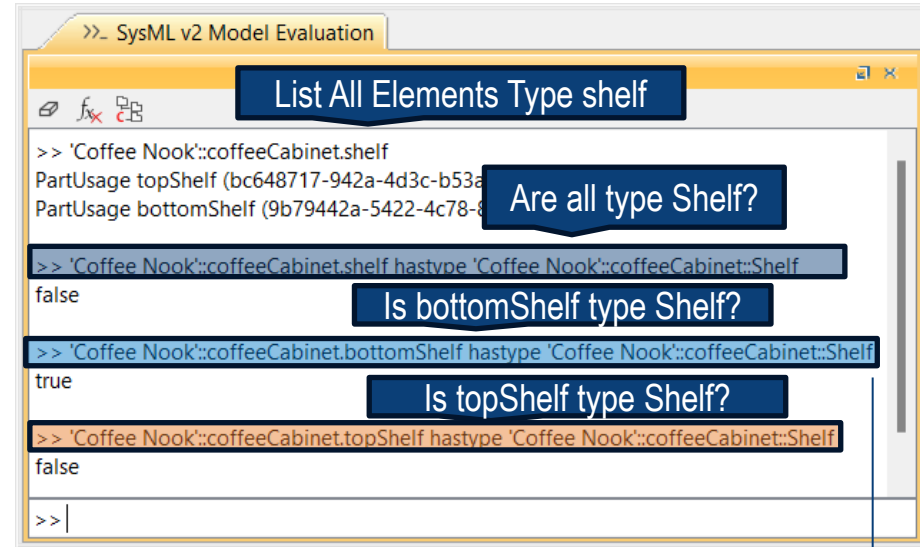
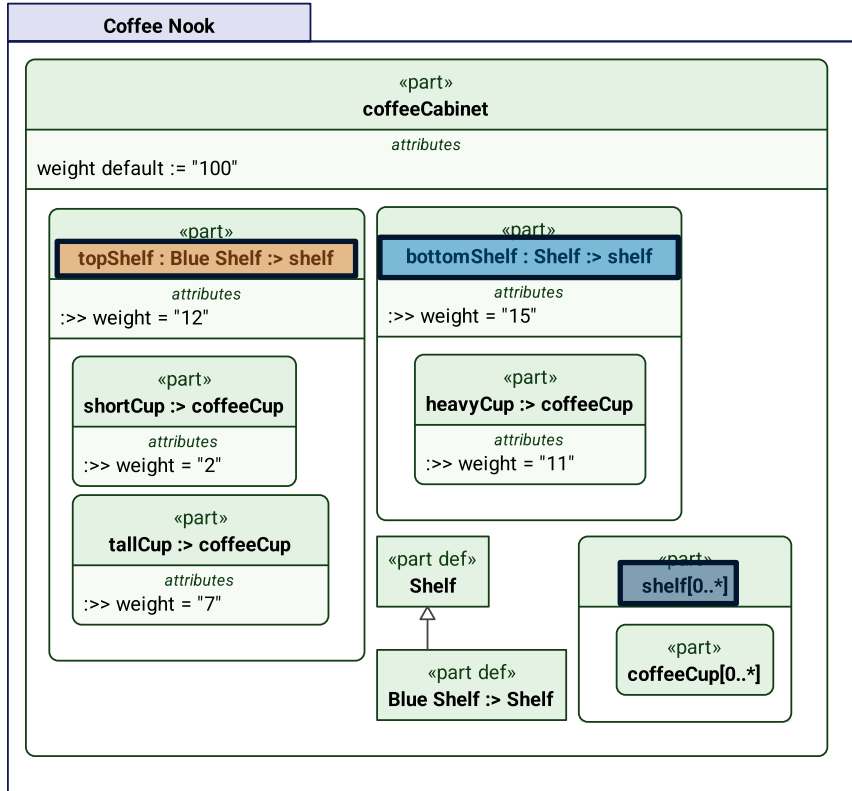
Since this has no metadata, it only returns the reflective metadata feature PartUsage (always the last element returned)



'Coffee Nook'::coffeeCabinet.metadata

Returns Metadata Features of the Named Element

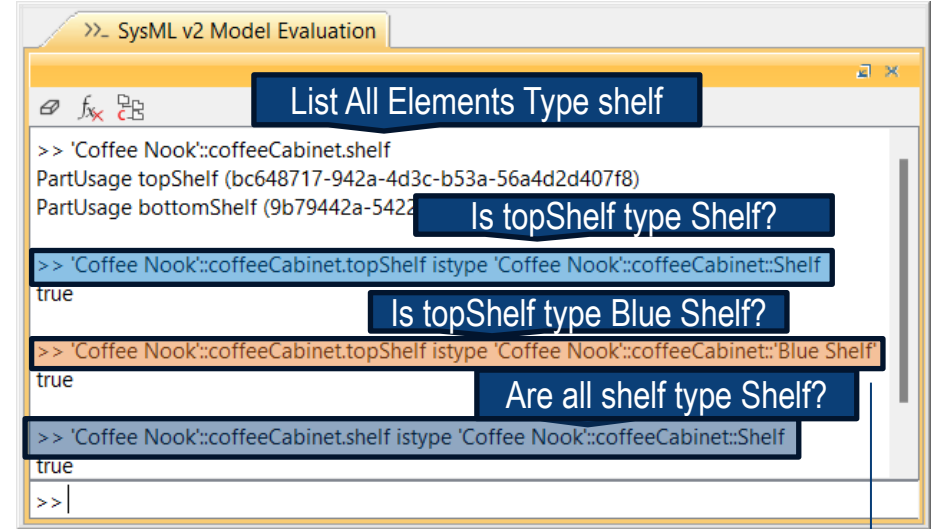
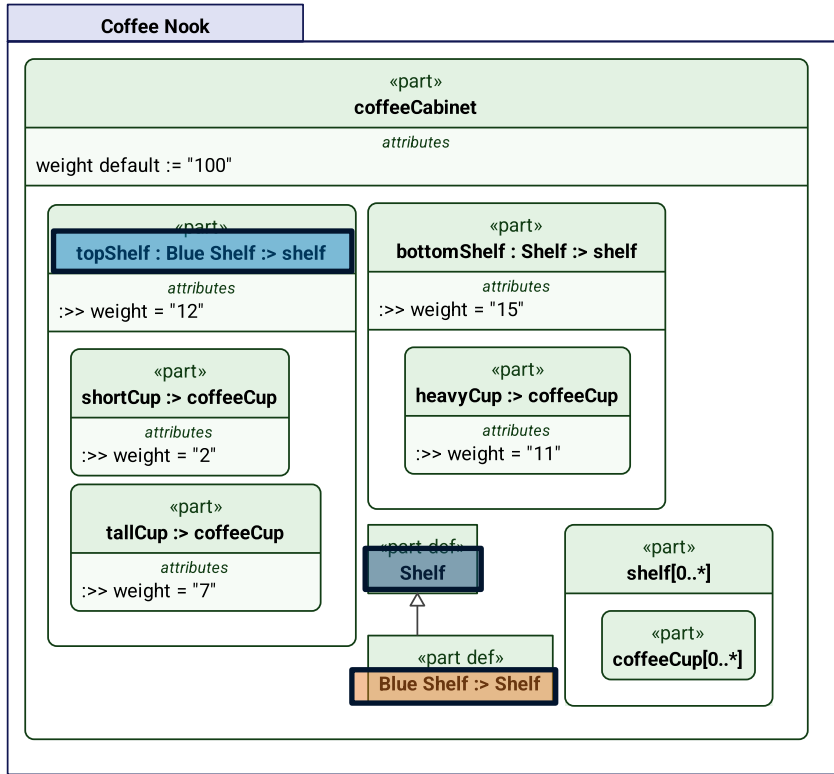
CLASSIFICATION TEST OPERATOR: HASTYPE – ALL



'Coffee Nook'::coffeeCabinet.bottomShelf
hastype 'Coffee Nook'::coffeeCabinet::Shelf

Hastype Checks if All are Direct Instances; Returns Boolean; Subsets not Accepted

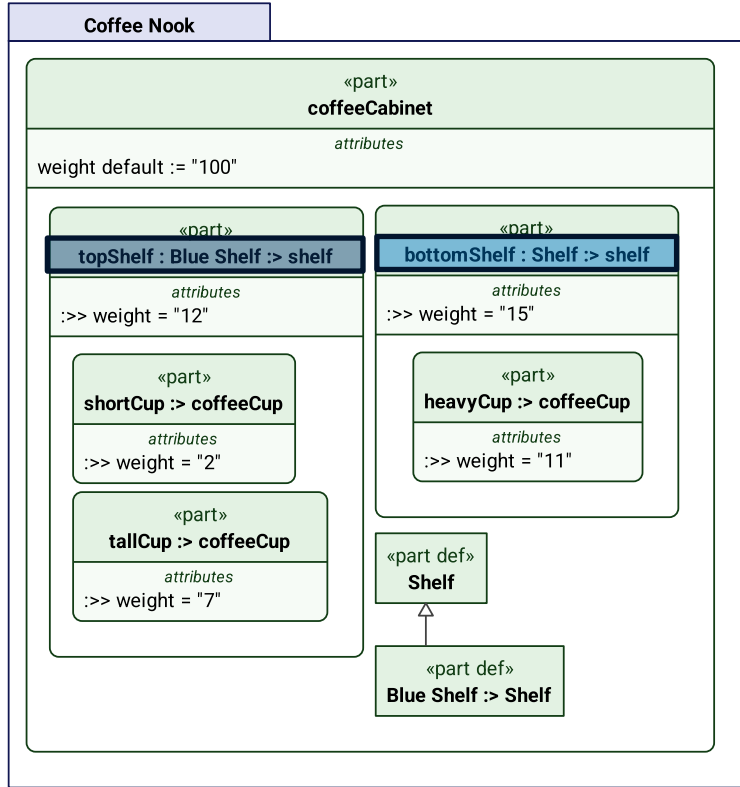
CLASSIFICATION TEST OPERATOR: ISTYPE - ALL



'Coffee Nook'::coffeeCabinet.topShelf istype
'Coffee Nook'::coffeeCabinet::'Blue Shelf'

Istype Checks if All are Direct or Indirect Instances; Returns Boolean; Subsets not Accepted

CAST OPERATOR: "AS" - OBJECT LEVEL



```
>> SysML v2 M

Also to be Used to Access Properties
in More Complex Queries

>> 'Coffee Nook'::coffeeCabinet.shelf as 'Coffee Nook'::coffeeCabinet::Shelf
PartUsage topShelf (bc648717-942a-4d3c-b53a-56a4d2d407f8)
PartUsage bottomShelf (9b79442a-5422-4c78-8ab8-a65c529e0f9d)

>>
```

'Coffee Nook'::coffeeCabinet.shelf as 'Coffee Nook'::coffeeCabinet::Shelf

Performs an istype Filtering to Elements of Direct or Indirect Type Shelf; Subsets not Accepted

META CAST OPERATOR: "META" – META LEVEL

Coffee Nook

«part»

coffeeCabinet

Also to be Used to Access Properties
in More Complex Queries

```
>> SysML  
>> 'Coffee Nook':coffeeCabinet meta SysML::PartUsage  
MetadataFeature PartUsage coffeeCabinet (1e657547-c04f-48ce-b51b-f26631dea429)  
>>
```

meta is shorthand for
“.metadata as”

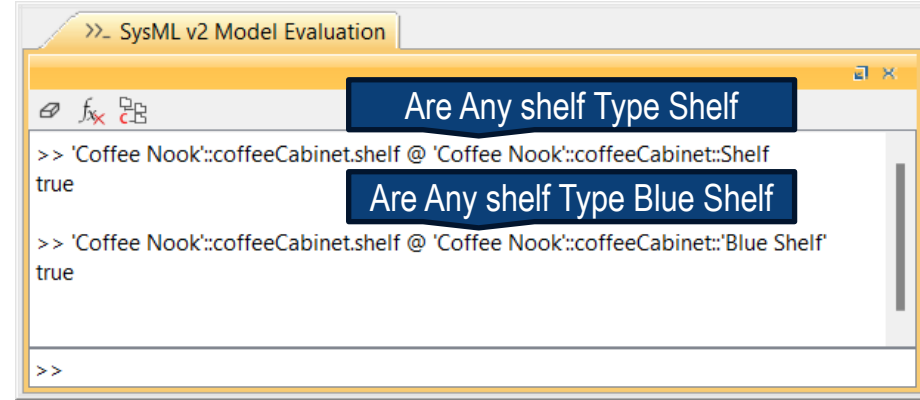
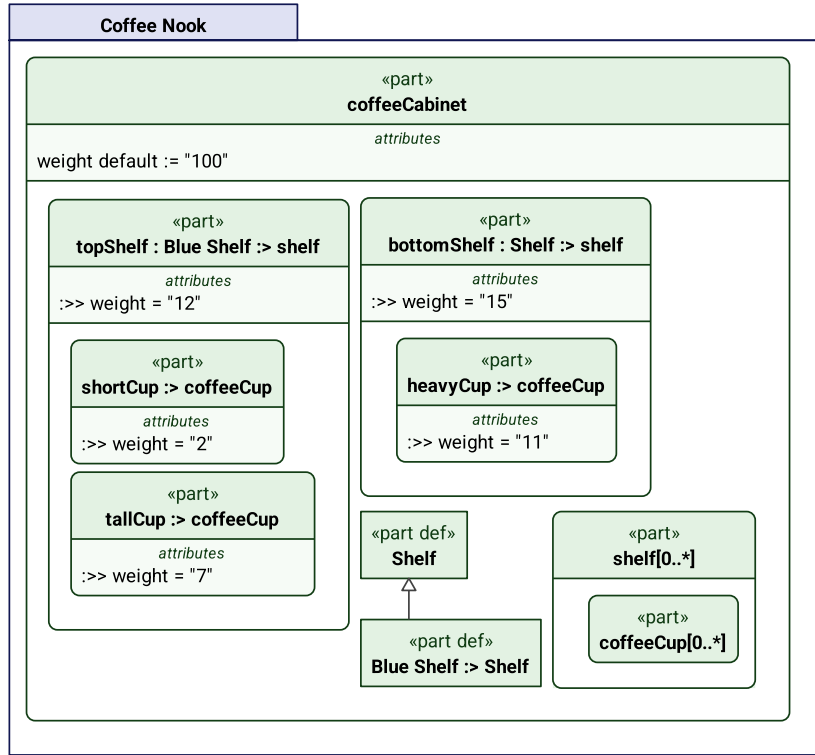
'Coffee Nook':coffeeCabinet meta
SysML::PartUsage

Is the same as

'Coffee Nook':coffeeCabinet.metadata as
SysML::PartUsage

Performs an istype Filtering to Elements of Direct or Indirect Type PartUsage; Subsets not Accepted

CLASSIFICATION TEST OPERATOR: @ - ANY – OBJECT LEVEL

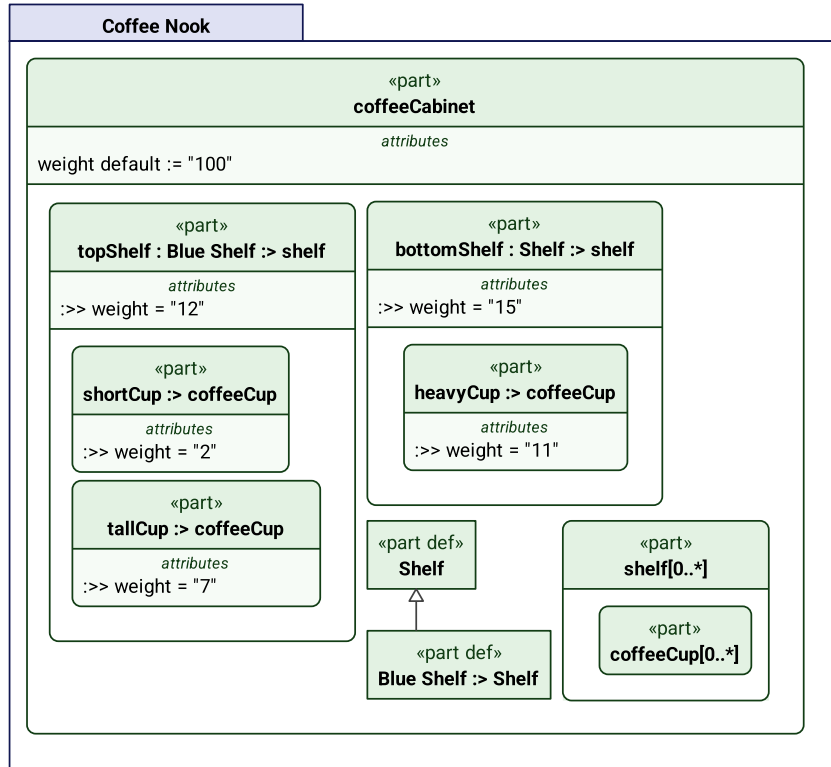


'Coffee Nook'::coffeeCabinet.shelf @ 'Coffee Nook'::coffeeCabinet::Shelf

'Coffee Nook'::coffeeCabinet.shelf @ 'Coffee Nook'::coffeeCabinet::'Blue Shelf'

@ Checks if Any are Direct or Indirect Instances; Subsets not Accepted

METAClassification Test Operator: @@ - ANY – META LEVEL



```
>> SysML v2 Model Evaluation

List All Elements Type shelf

>> 'Coffee Nook'::coffeeCabinet.shelf
PartUsage topShelf (bc648717-942a-4d3c-b53a-56a4d2d407f8)
PartUsage bottomShelf (9b...)

Is bottomShelf a PartUsage?

>> 'Coffee Nook'::coffeeCabinet::bottomShelf @@ SysML::PartUsage
true

Is bottomShelf an ActionUsage?

>> 'Coffee Nook'::coffeeCabinet::bottomShelf @@ SysML::ActionUsage
false

Is bottomShelf an OccurrenceUsage?

>> 'Coffee Nook'::coffeeCabinet::bottomShelf @@ SysML::OccurrenceUsage
true

Are there Any shelf that are PartUsage?

>> 'Coffee Nook'::coffeeCabinet::shelf.(in i; i @@ SysML::PartUsage)
true

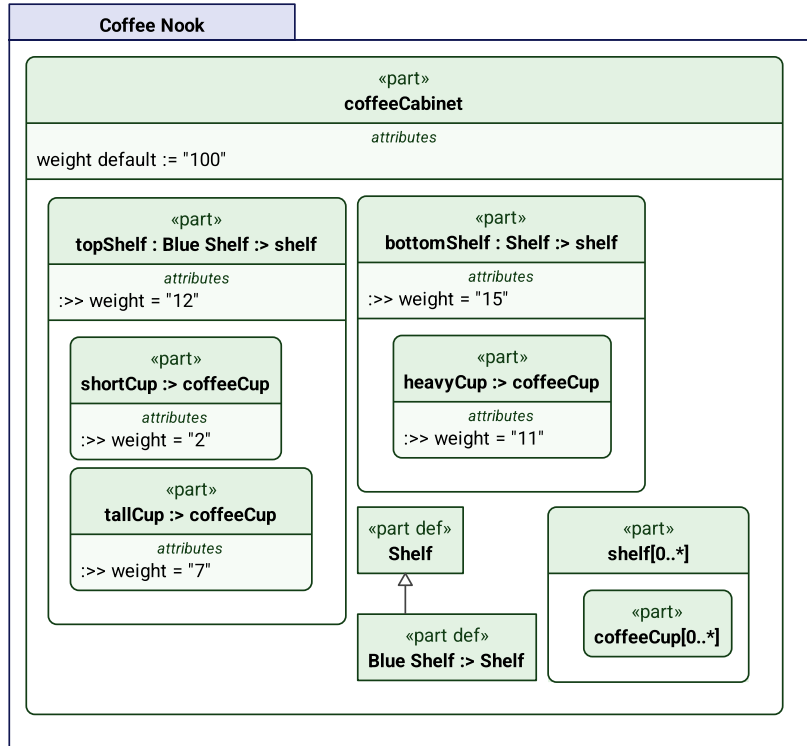
>>
```

'Coffee Nook'::coffeeCabinet::bottomShelf @@ SysML::PartUsage

@@ Checks if Any are Direct or Indirect Instances; Subsets not Accepted

ALL – TYPE EXTENT (INCLUDES SUBSETS)

Works for Meta and Object Levels



```
>> SysML v2 Model Evaluation

>> all 'Coffee Nook'::coffeeCabinet::'Blue Shelf'
PartUsage topShelf (b

>> all 'Coffee Nook'::coffeeCabinet::Shelf
PartDefinition Blue Shelf (8b7db73a-ca60-4cf2-a765-086f95c772c4)
PartUsage bottomShelf (9b79442a-5422-4c78-8ab8-a65c529e0f9d)
PartUsage topShelf (b

>> all 'Coffee Nook'::coffeeCabinet::shelf
PartUsage bottomShelf (9b79442a-5422-4c78-8ab8-a65c529e0f9d)
PartUsage topShelf (b

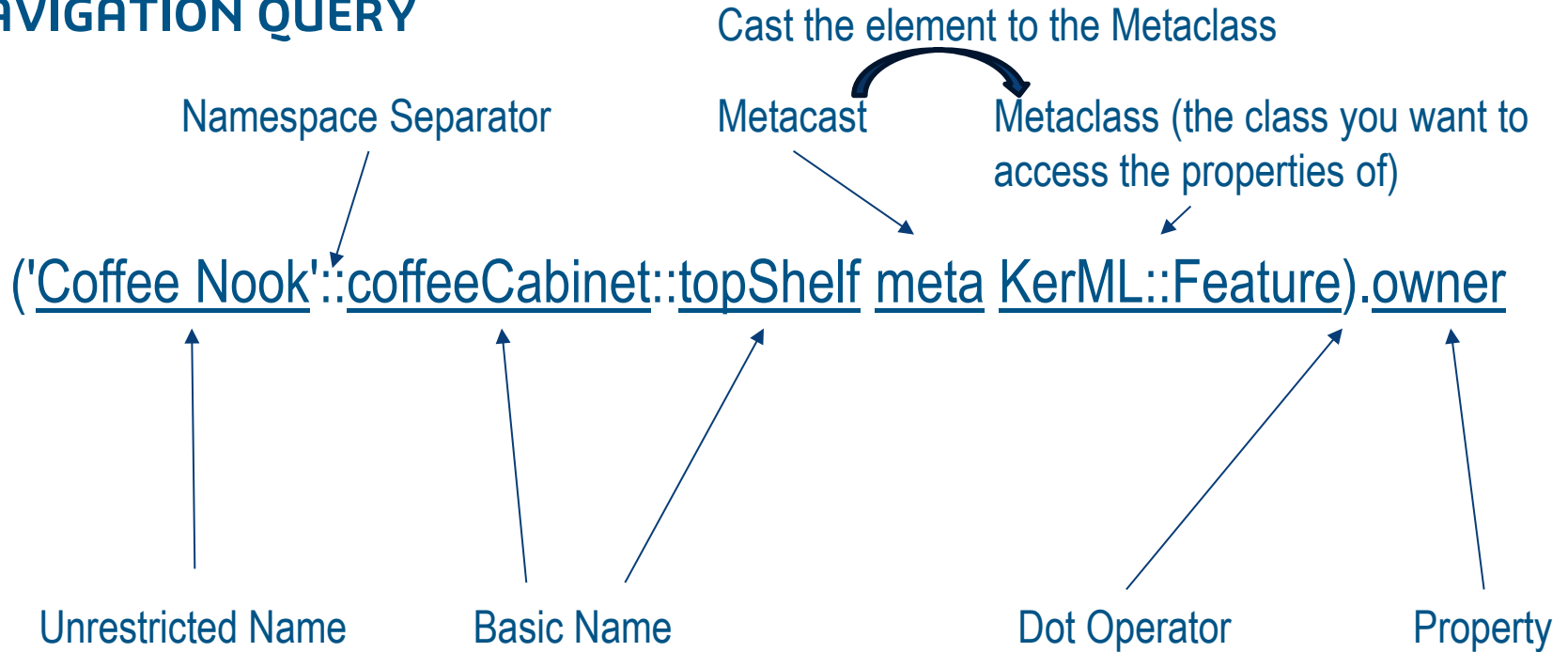
>> all 'Coffee Nook'::coffeeCabinet::shelf::coffeeCup
PartUsage heavyCup (941c9535-ea86-419f-9740-25dd6d810f49)
PartUsage shortCup (fdc41698-8823-493b-a1b6-bb6f118b50ca)
PartUsage tallCup (310a64b5-d6e1-454d-b4a1-4c467d3731c8)

>>
```

all 'Coffee
Nook'::coffeeCabinet::shelf::coffeeCup

All Returns All Direct and Indirect Element Types or Subsets

NAVIGATION QUERY



Goal: Get the Owner of topShelf.

NAVIGATION QUERY EVALUATED

Evaluation Window

Coffee Nook

Returned Value

«part»

coffeeCabinet

attributes

weight default := "100"

«part»

topShelf : Blue Shelf := shelf

attributes

:>> weight = "12"

2

>> SysML v2 Model Evaluation



```
>> ('Coffee Nook::coffeeCabinet::topShelf meta KerML::Feature).owner  
MetadataFeature PartUsage coffeeCabinet (1e657547-c04f-48ce-b51b-f26631dea429)
```

```
>> |
```

1

Namespace Separator

Metacast

Metaclass (the class you want to access the properties of)

('Coffee Nook'::coffeeCabinet::topShelf meta KerML::Feature).owner

Unrestricted Name

Basic Name

Dot Operator

Property

COLLECT FUNCTION – “.{}”

Collect Expression consists of a **primary expression** and **expression body**

primaryExpression.{**expressionBody**} or **primaryExpression**->ControlFunctions::collect{**expressionBody**}

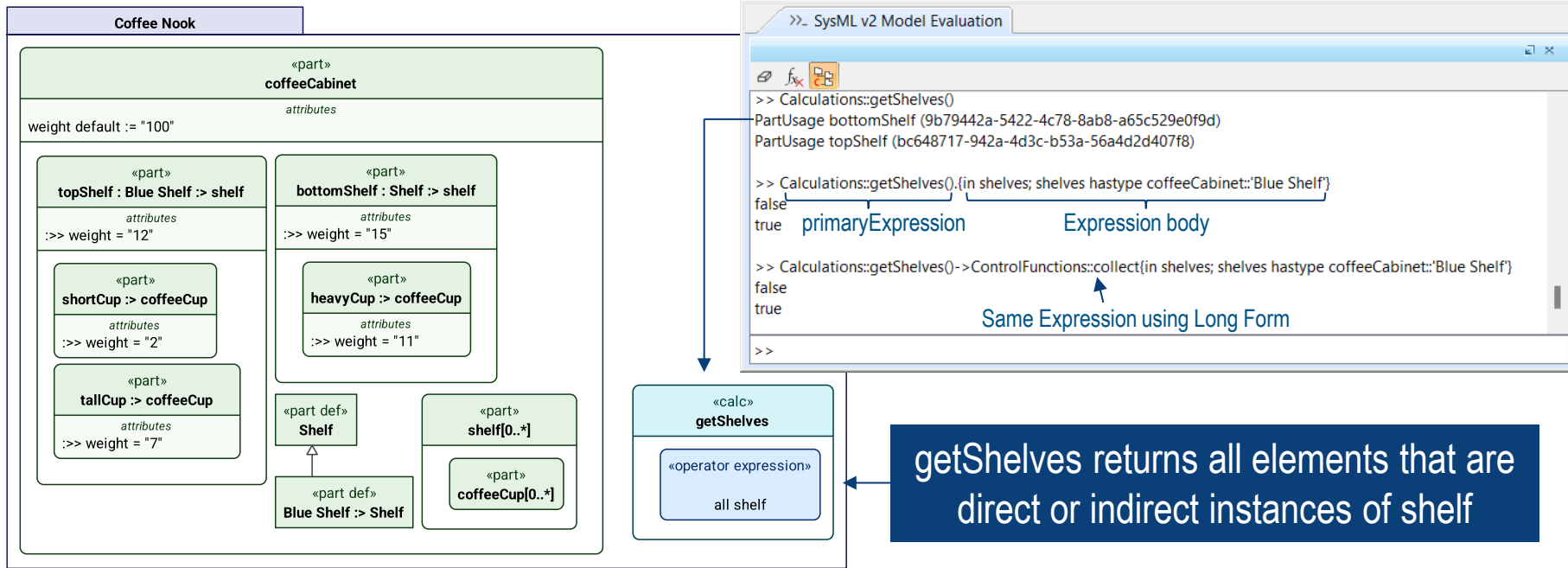
Expression Body requires a single input parameter

Each result of the primary expression is evaluated based on the expression body and collected into a sequence in order of evaluation

.{} is shorthand for ControlFunctions::collect

A Collect Expression Returns Results Of The Primary Expression As An Ordered Sequence.

COLLECT FUNCTION – “.{}” (CONT.)



Calculations::getShelves().{in shelves; shelves hastype coffeeCabinet::'Blue Shelf'}

Collect Returns Evaluated Results in a Sequence (Elements/Booleans/Strings/etc)

SELECT FUNCTION – “.?{”

Select Expression consists of a **primary expression** and **expression body**

primaryExpression.?**{expressionBody}** or **primaryExpression**->ControlFunctions::select**{expressionBody}**

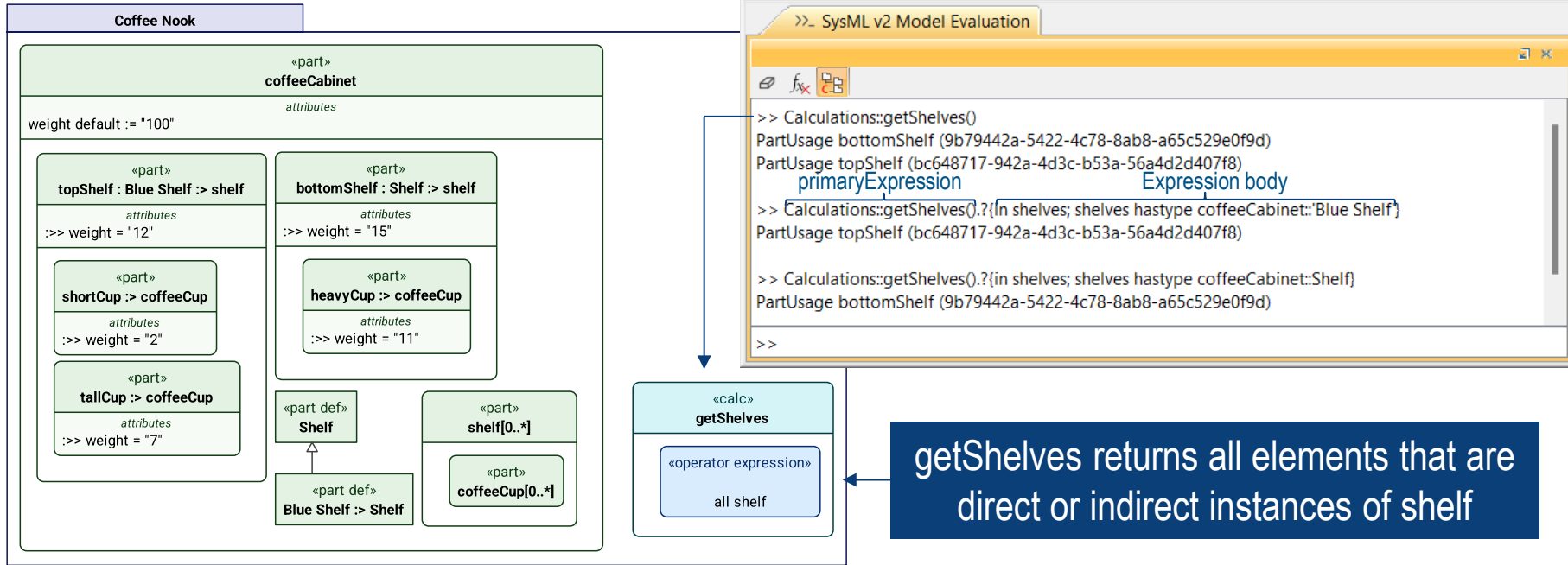
Expression Body requires a single input parameter and a Boolean result

Each result of the primary expression is evaluated based on the expression body and true evaluations are selected into a sequence in order of evaluation

.?**{}** is shorthand for ControlFunctions::select

A Select Expression Filters Results To Keep Only Those Matching A Boolean Condition.

SELECT FUNCTION – “.{ }” (CONT.)



Calculations::getShelves().{in shelves; shelves hastype coffeeCabinet::'Blue Shelf'}

Select Filters Elements into a Sequence

SUMMARY OF TYPE-CHECKING OPERATORS (RETURN BOOLEAN)

Operator	Scope	Strictness	Behavior
istype	Object/ Meta	Direct + Indirect	True if ALL elements are <i>directly or indirectly</i> this type. No subsets.
hastype	Object/ Meta	Direct Only	True if ALL elements are <i>directly</i> this type. No subsets.
@	Object	Direct + Indirect	True if Any elements are <i>directly or indirectly</i> this type at the <i>object-level</i> . No subsets.
@@	Meta	Direct + Indirect	True if Any elements are <i>directly or indirectly this</i> type at the <i>meta-level</i> . No subsets.

CASTING OPERATORS (RETURNS ELEMENTS)

Operator	Logic	Return Type / Behavior
all	ALL instances	Returns all elements that are <i>directly or indirectly</i> this type <i>including subsets</i> .
as	ANY (Object level)	Casts/returns any elements that are <i>directly or indirectly</i> this type at the <i>object-level</i> . No Subsets.
meta	ANY (Meta level)	Casts/returns any elements that are <i>directly or indirectly</i> this type at the <i>meta-level</i> . No Subsets.

PROCESSING OPERATORS

COLLECTION PROCESSING OPERATORS

Operator	Syntax	Purpose
collect	<code>collect()</code> or <code>.{} </code>	Checks the sequence, returning expression results for each.
select	<code>select()</code> or <code>.?{} </code>	Filters the sequence, keeping only items that evaluate to True.

INTRO TO HANDS-ON ACTIVITIES



Milestone Model Explanation

- ▶ Each Hands-On Activity (HOA) has a model to start with in case you get stuck with a previous HOA
- ▶ For example:
 - ▷ If you couldn't complete the Hands-on Activity 3 for some reason, open the M3 model as a baseline to start the Hands-on Activity 4



MILESTONE MODELS

Starting
Point
Models

Hands-On Activity #	Starting Model	Description
1	NA	Evaluation and Simple Calculations - Create a new model
2	M1	Custom Tables with Calculations
3	M2	Getting Requirements
4	M3	Getting Actions
5	M4	Counting Elements
N/A	M5	Final Result

M2

HOA = Hands-On Activity

Hands-On Activity 1: Evaluation and Simple Calculations

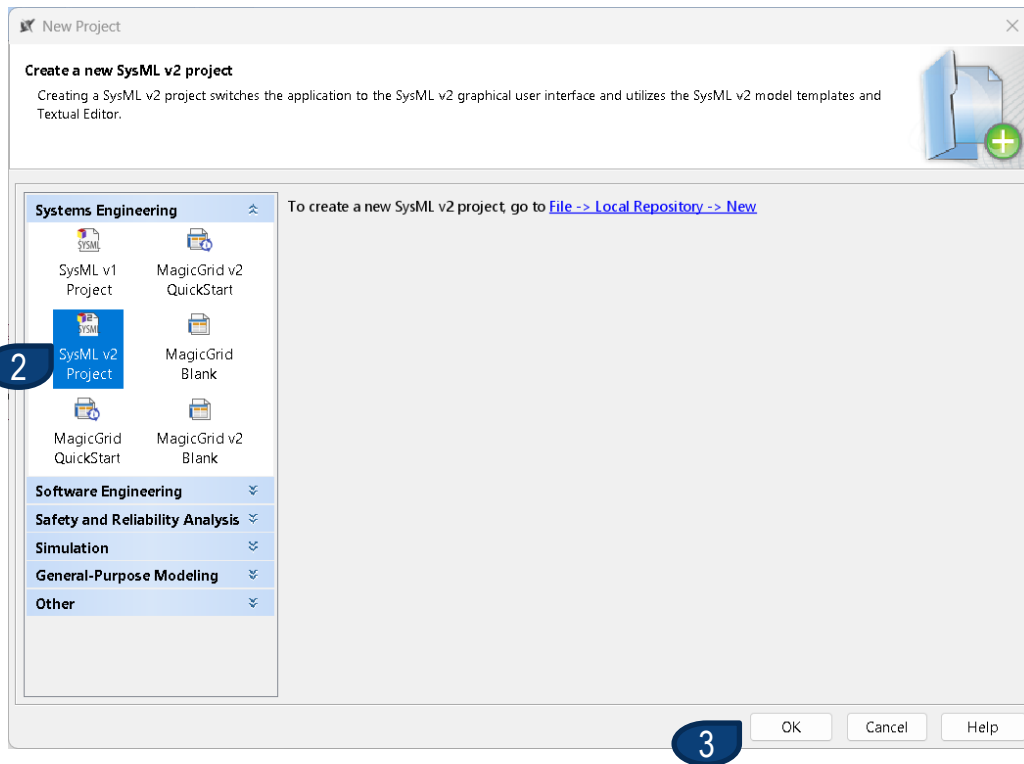
- ▶ Create a new model and patch in a basic structure
- ▶ Learn how to utilize the features of evaluation
- ▶ Create a simple calc def and use it to evaluate



Hands-On Activity 1: Create a New Model

Create a new model

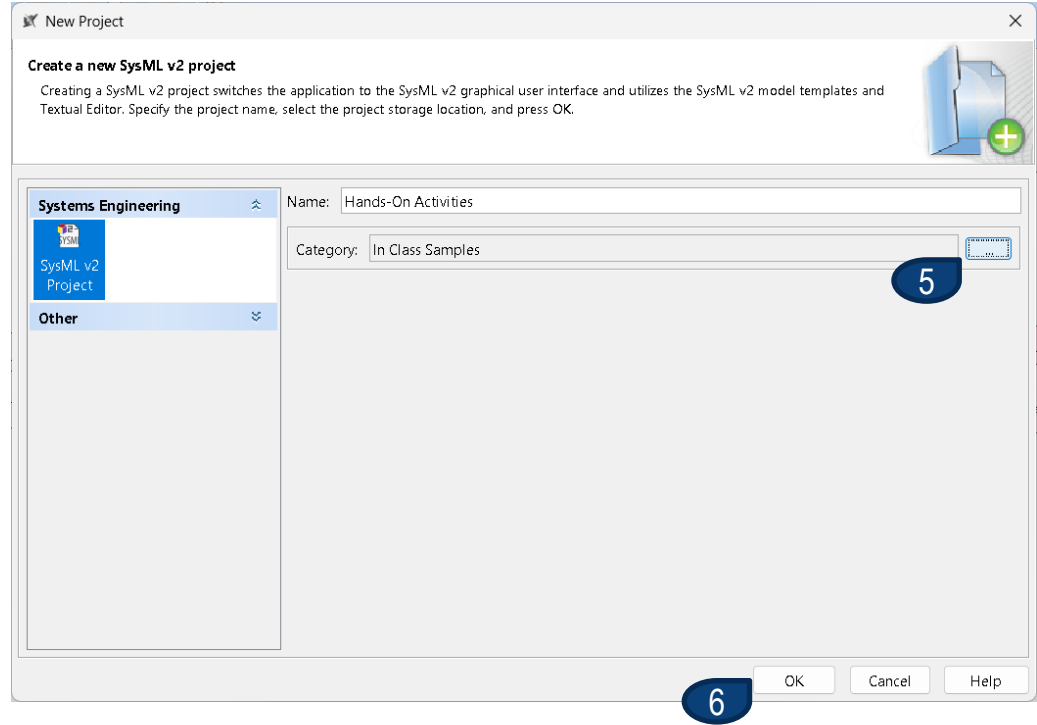
1. Click File>New Project
2. Select “SysML v2 Project” under the Systems Engineering category
3. Select “OK”



Hands-On Activity 1: Create a New Model (Cont.)

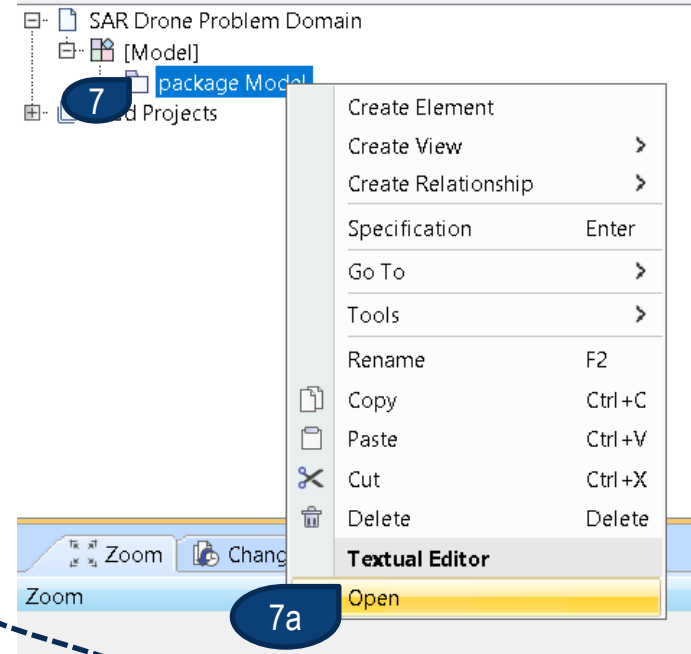
Create a new model

4. Give it a Name
5. Select the Project Category by selecting the "..."
 - a. Create a new category or select Uncategorized 
 - b. Once Selected, Select "OK"
6. Select "OK"

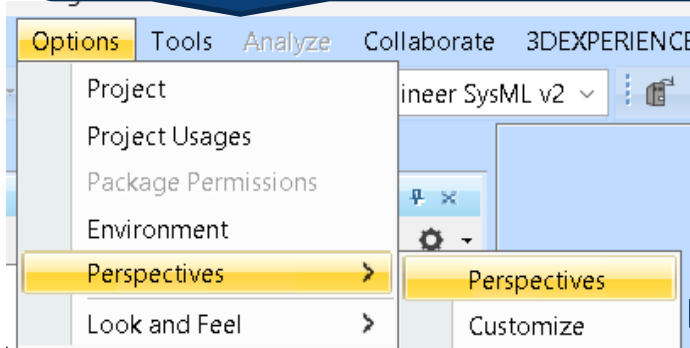


Hands-On Activity 1: Create a New Model (Cont.)

7. Right Click on the package “Model”
 - a. Select Open



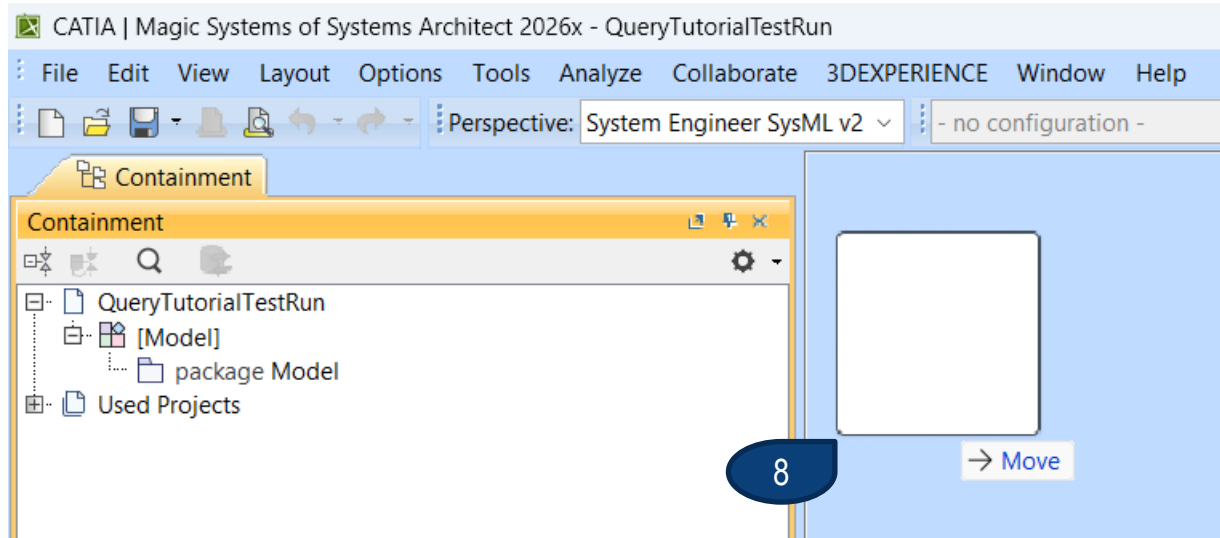
Note: Double check Perspective is set to “System Engineer SysML V2



System Engineer SysML v2 (Current)

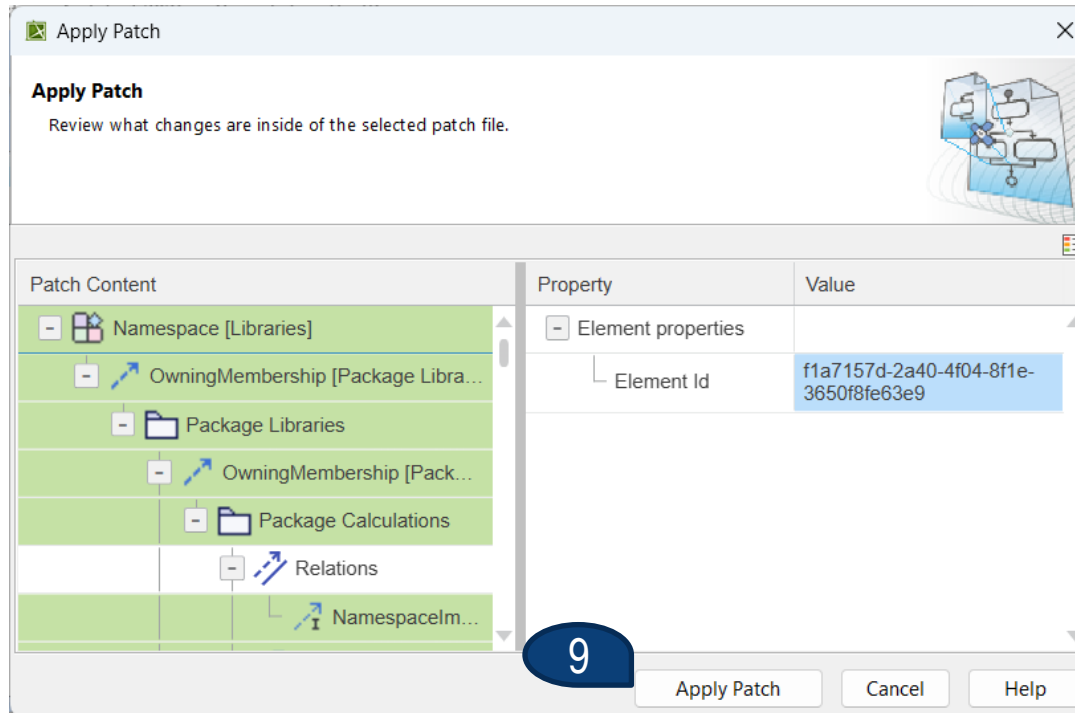
Hands-On Activity 1: Create a New Model (Cont.)

- From your Student Files, drag “Starting Model Patch.pzip” into CATIA Magic



Hands-On Activity 1: Create a New Model (Cont.)

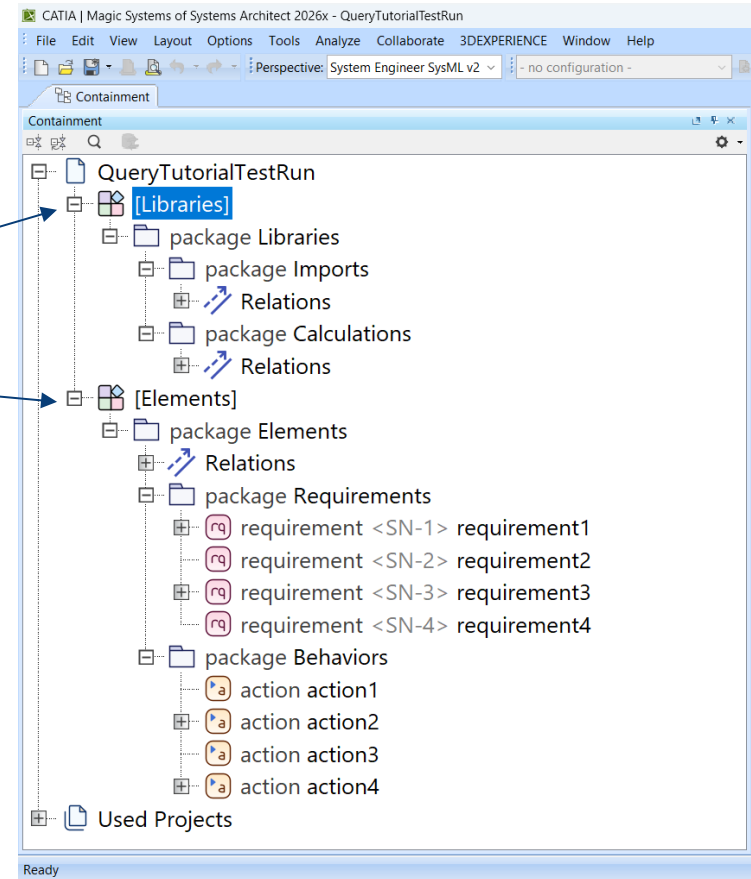
9. From the Apply Patch popup window, select “Apply Patch”



Hands-On Activity 1: Create a New Model (Cont.)

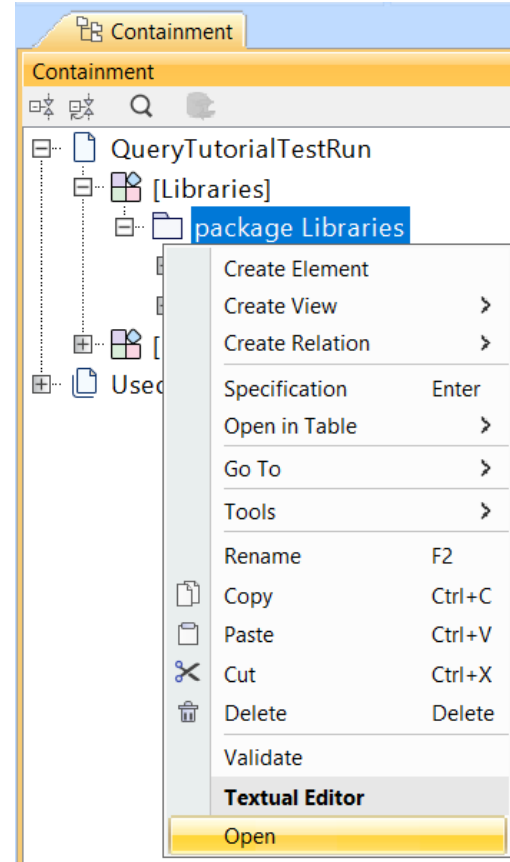
10. Your Containment Window should look similar to this when expanded:

- a. Libraries Root Namespace
- b. Elements Root Namespace



Hands-On Activity 1: Create a New Model (Cont.)

11. Right Click>Open (Textual Editor)



Hands-On Activity 1: Create a New Model (Cont.)

12. Type or copy/paste the calc def “HelloWorld” below after the imports in the Calculations package in the textual notation

Annotations for the HelloWorld definition:

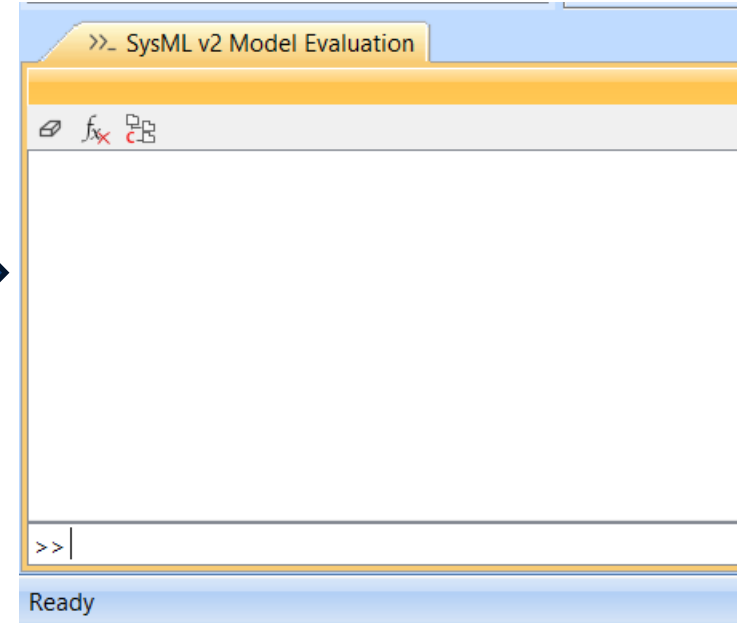
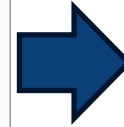
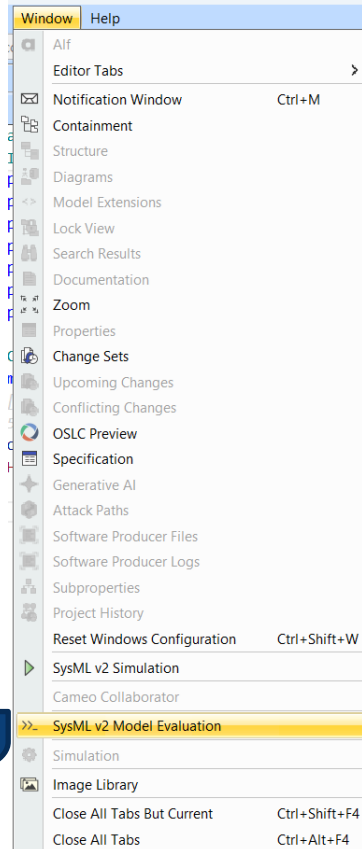
- Element Type: `calc`
- Element Name: `def HelloWorld`
- Element Body: `{`
- Input Definition: `in count : ScalarValues::Integer`
- Multiplicity: `[1];`
- Input Variable: `in count`
- Sequence: `(1..count)`
- KerML Collect Function: `collect`
- String to be repeated each count: `"Hello World"`

```
calc def HelloWorld {  
  in count : ScalarValues::Integer [1];  
  (1..count) -> collect {  
    "Hello World"  
  }  
}
```

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 1: Create a New Model (Cont.)

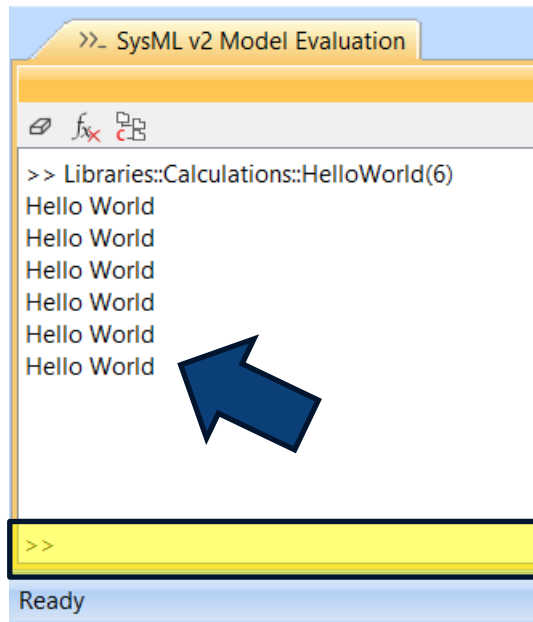
13. Click Window>SysML v2 Model Evaluation



Hands-On Activity 1: Create a New Model (Cont.)

14. Type or copy/paste the function call below into the Eval Window

Libraries::Calculations::HelloWorld(6)



Hands-On Activity 2: Custom Tables

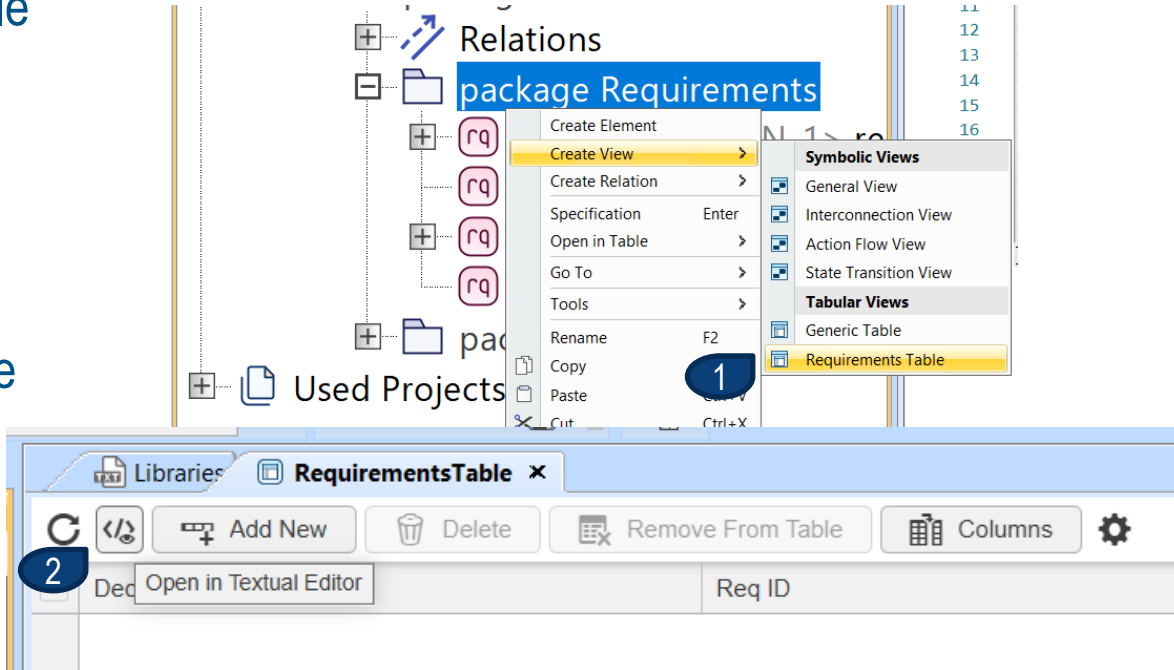
- ▶ Create a new table with custom columns
- ▶ Use new expression on custom column



Hands-On Activity 2: Create a New Table

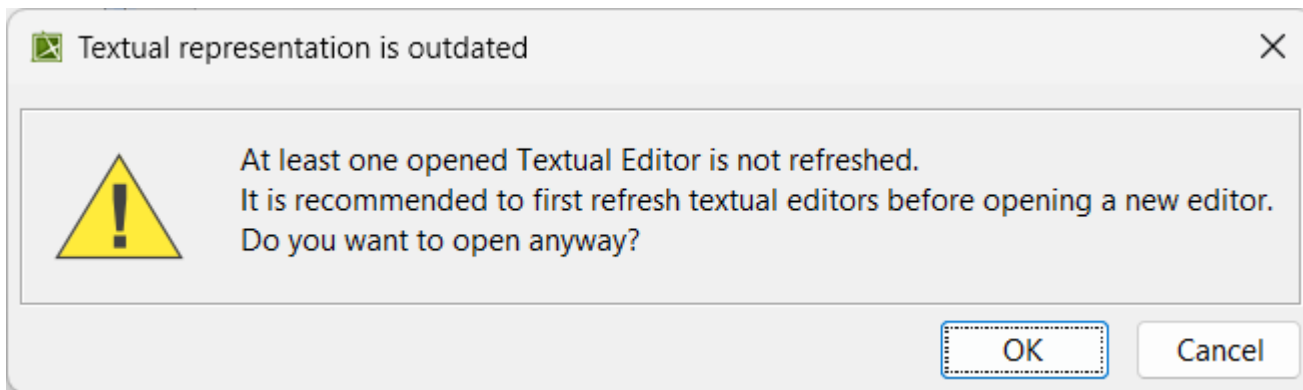
Create a new Requirements Table

1. Right Click on the Requirements package>Create View>Requirements Table
2. Select the  from the Table Toolbar to Open in Textual Editor



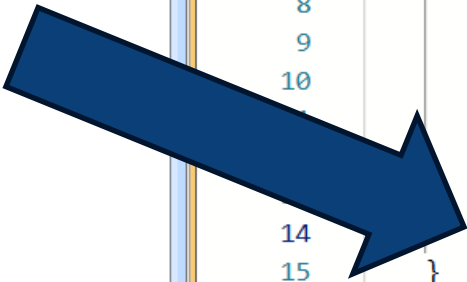
Hands-On Activity 2: Create a New Table

Note: You may receive this popup, it is because you edited the Textual Representation without synchronizing the model. Select OK.



Hands-On Activity 2: Create a New Table

The new table should be highlighted in the textual notation as shown here:



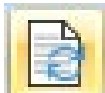
```
1 package [$1] Elements {
3     package [$2] Requirements {
4         requirement [$3] <'SN-1'> requirement1 {
5             requirement [$4] <'SN-1.1'> requirement6;
6         }
7         requirement [$5] <'SN-2'> requirement2;
8         requirement [$6] <'SN-3'> requirement3 {
9             requirement [$7] <'SN-3.1'> requirement7 {
10                requirement [$8] <'SN-3.1.1'> requirement8;
11            }
12        }
13        requirement [$9] <'SN-4'> requirement4;
14        view [$10] RequirementsTable : rt;
15    }
16 package [$11] Behaviors {
```

Hands-On Activity 2: Create a New Table

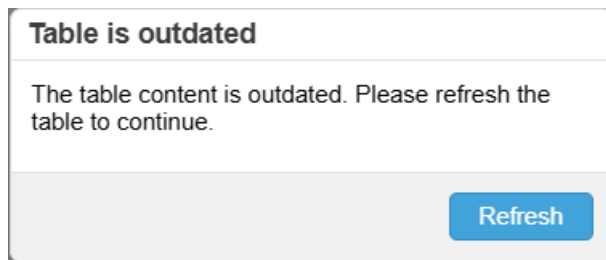
3. Expose the Requirements package in order to set the scope of the table.

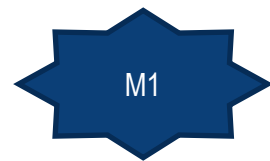
```
view RequirementsTable : rt{  
    expose Requirements::**;  
}
```

4. Click Synchronize Model with Textual Notation



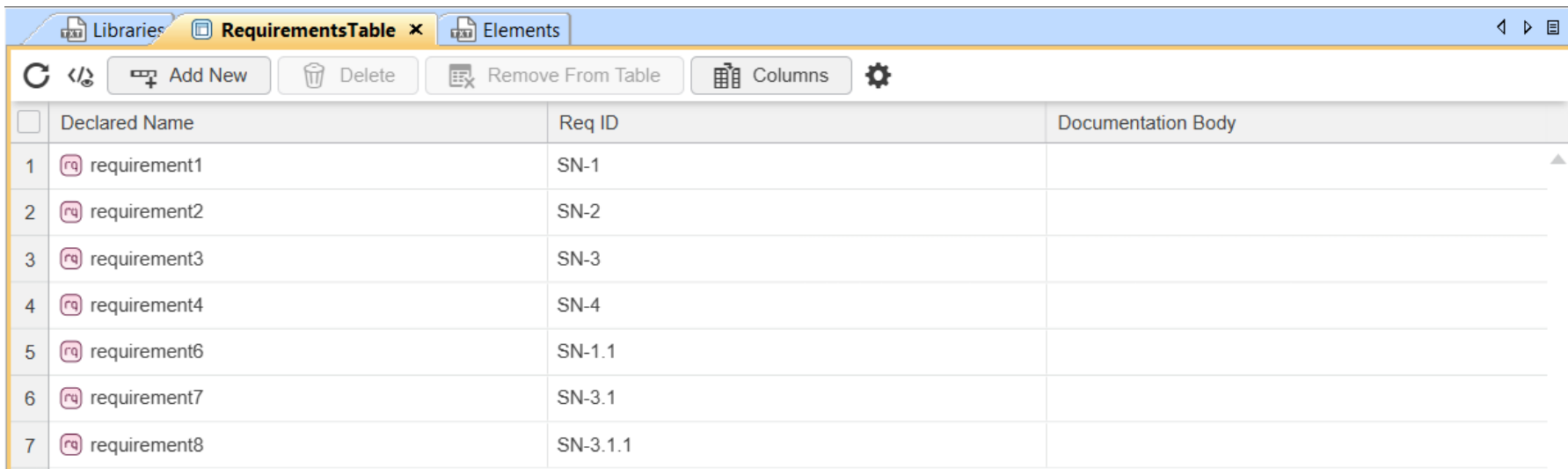
5. Go to the table tab and select Select Refresh





Hands-On Activity 2: Create a New Table

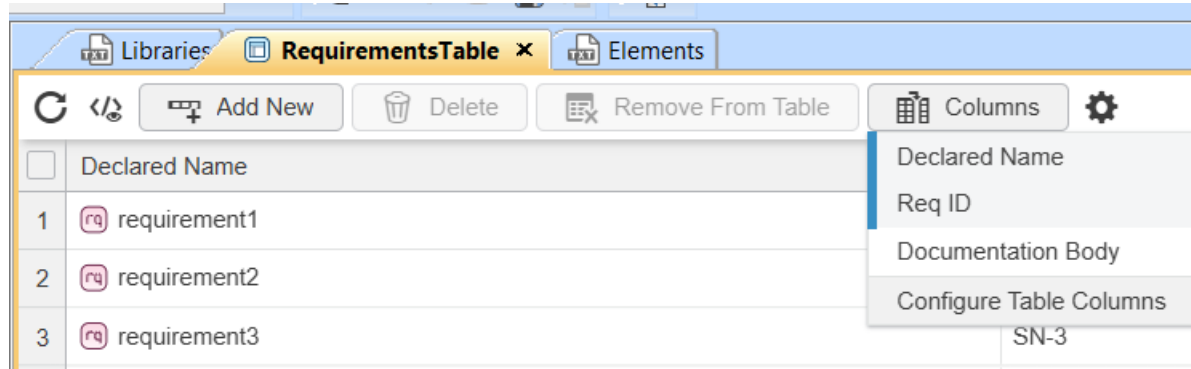
Table should look like this:



	Declared Name	Req ID	Documentation Body
1	 requirement1	SN-1	
2	 requirement2	SN-2	
3	 requirement3	SN-3	
4	 requirement4	SN-4	
5	 requirement6	SN-1.1	
6	 requirement7	SN-3.1	
7	 requirement8	SN-3.1.1	

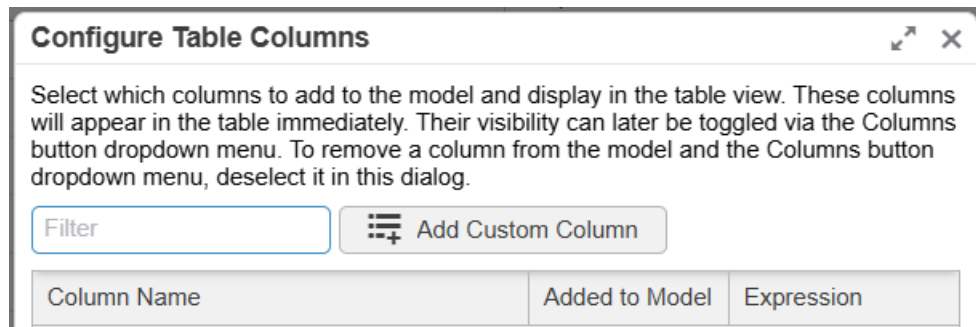
Hands-On Activity 2: Edit Columns

6. Click “Columns”
 - a. Deselect “Documentation Body” by Left Clicking on “Documentation Body”
 - b. Select “Configure Table Columns”



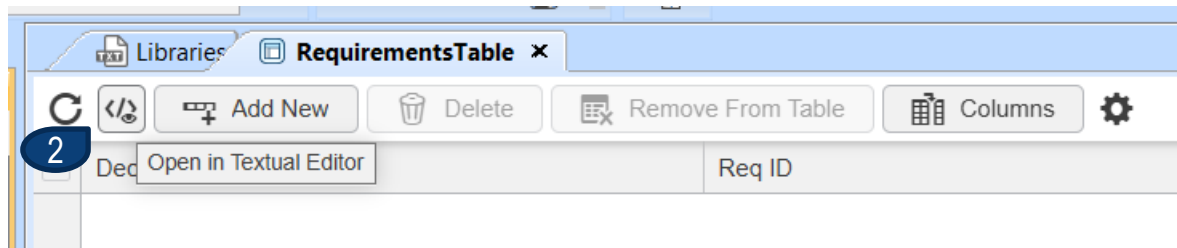
Hands-On Activity 2: Edit Columns

7. From the “Configure Table Columns” window, select “Add Custom Column”



8. Select OK

9. Open the Textual Editor



Hands-On Activity 2: Edit Columns

Note: you may need to select Refresh if the Textual Editor was already open.

Textual representation of model is outdated. Please refresh before continuing. [Refresh?](#)

- Focusing on the text below, notice this is the result of the custom column creation

```

13     requirement [$9] <'SN-4'> requirement4;
14     view [$10] RequirementsTable : rt {
15         expose Requirements::**;
16         render rendering [$11] asTable :>> asTable {
17             view [$12] :>> 'Declared Name';
18             view [$13] :>> 'Req ID';
19             view [$14] :>> 'Documentation Body' [0];
20             view [unnamed1 : ColumnByExpressionView :> column {
21                 render rendering [DS_Views::CoreViews::asTableCell {
22                     calc [$20] :>> getValue {
23                         in [$26] : rowElement;
24                         return [$27] : result = "";
25                     }
26                 }
27             }
28         }
29     }

```

Column Name

Column Type

Rendering Type (Return Type)

Calculated Expression

OVERVIEW ADDING TABLE COLUMNS BY FEATURE

► 2 Ways

▷ By Feature or Expression

Column	Description	Cells	Column attributes
<i>ColumnByFeatureView</i>	Dedicated to simple metadata feature value retrieval. It requires the expression to return only 1 metadata feature, like <i>Element::declaredName</i> or <i>FeatureValue::value</i> . The expression is specified for the feature <i>columnFeature</i> .	•For <i>ColumnByFeatureView</i>, the cell type is calculated out of the metadata feature and does not need to be specified explicitly. •The String, Element, Enum, Integer, Boolean, and Real cell values can be edited in the table view's graphical representation for <i>ColumnByFeatureView</i> column cells.	•<i>isMultiline</i>. It allows you to specify whether single-valued string cells should be displayed in a multiline format. The default value is <i>false</i>.

OVERVIEW ADDING TABLE COLUMNS BY EXPRESSION

Column	Description	Cells	Column attributes
<i>ColumnByExpressionView</i>	Dedicated to complex metadata feature value retrieval. For example, when the value of the owned attribute needs to be retrieved, the feature chain expression is needed to access this information. See procedure >>	•For <i>ColumnByExpressionView</i> , the cell type is specified by defining the cell rendering with the appropriate rendering definition. It works as a kind of filter for the type of wanted expression results. See procedure >>	• <i>isMultiline</i> . It allows you to specify whether single-valued string cells should be displayed in a multiline format. The default value is <i>false</i> . See procedure >>

Rendering Definitions that can be used for each cell:

- Boolean
- Element
- Enum
- Integer
- Metaobject
- RealCellRendering
- String
- Table

The editing of *ColumnByExpressionView* cells is not yet supported in the table view's graphical representation.

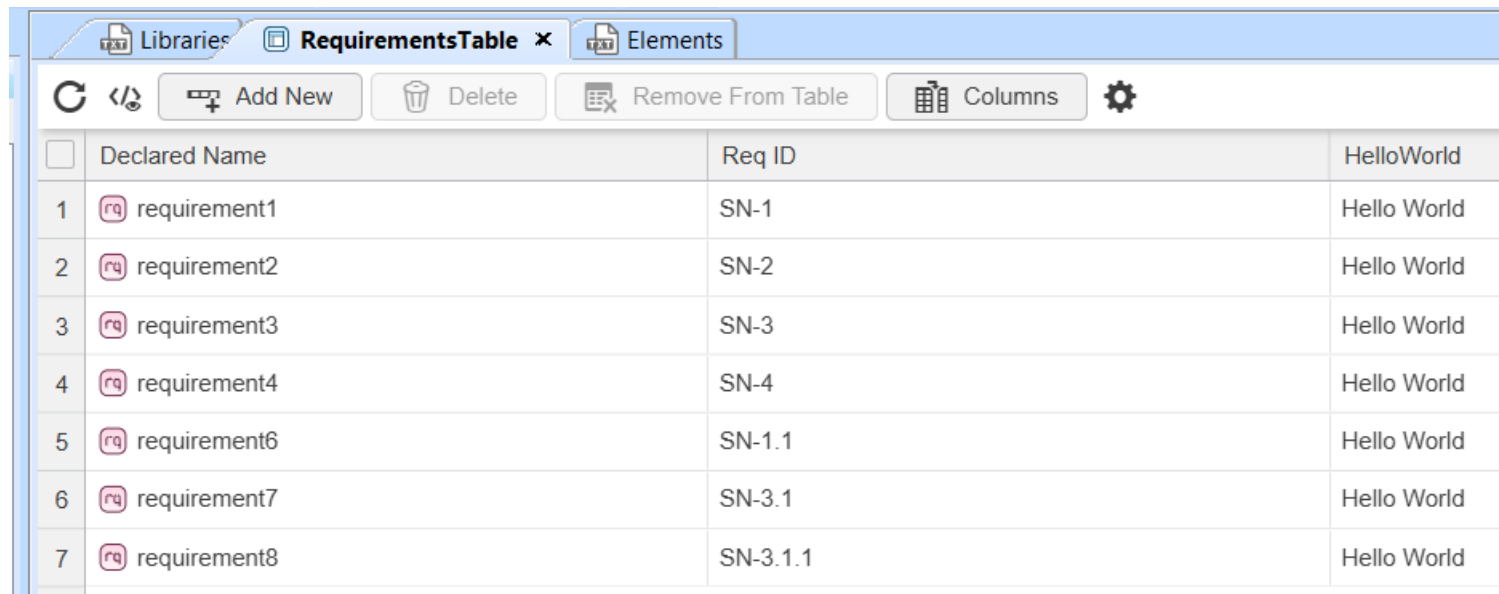
Hands-On Activity 2: Edit Columns

11. Update result string to be “Hello World” and the Column Name to “HelloWorld”

```
view [$10] RequirementsTable : rt {  
  expose Requirements::**;  
  render rendering [$11] asTable :>> asTable {  
    view [$12] :>> 'Declared Name';  
    view [$13] :>> 'Req ID';  
    view [$14] :>> 'Documentation Body' [0];  
    view [$23] HelloWorld : ColumnByExpressionView :> column {  
      render rendering [$24] : StringCellRendering :>> DS_VIEWS::CoreViews::asTableCell  
      calc [$25] :>> getValue {  
        in [$26] rowElement;  
        return [$27] result = "Hello World";  
      }  
    }  
  }  
}
```

Hands-On Activity 2: Review Table

Result:



The screenshot shows a software interface with a tabbed header containing 'Libraries', 'RequirementsTable' (selected), and 'Elements'. Below the header is a toolbar with icons for 'Add New', 'Delete', 'Remove From Table', 'Columns', and a settings gear. The main area displays a table with 8 rows of requirement data.

	Declared Name	Req ID	HelloWorld
1	 requirement1	SN-1	Hello World
2	 requirement2	SN-2	Hello World
3	 requirement3	SN-3	Hello World
4	 requirement4	SN-4	Hello World
5	 requirement6	SN-1.1	Hello World
6	 requirement7	SN-3.1	Hello World
7	 requirement8	SN-3.1.1	Hello World

Hands-On Activity 3: Getting Requirements

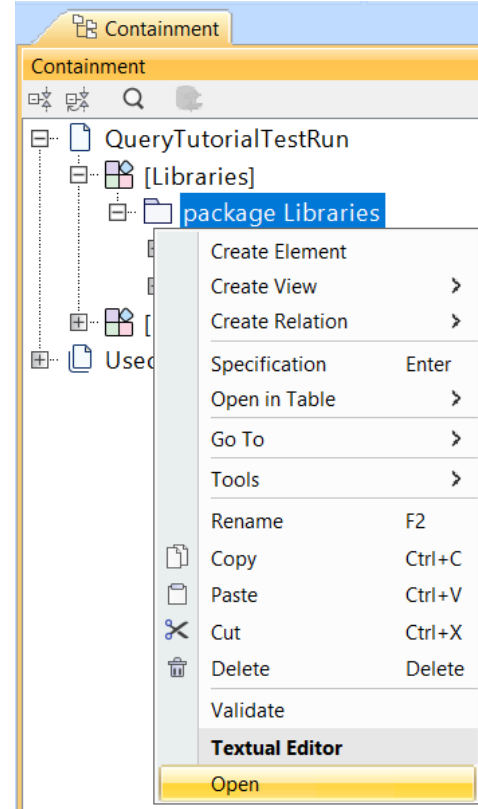
- ▶ Create a new calc def for getting all requirements in a package
- ▶ Apply it in the table



Hands-On Activity 3: Getting Requirements

1. Right Click>Open (Textual Editor)

Note: For the following steps we will be adding textual notation to the model. You will need to remember to Synchronize the textual notation each time in order for it to apply.



Hands-On Activity 3: Getting Requirements (Cont.)

2. Add the following text under the package “Calculations” after the HelloWorld function:

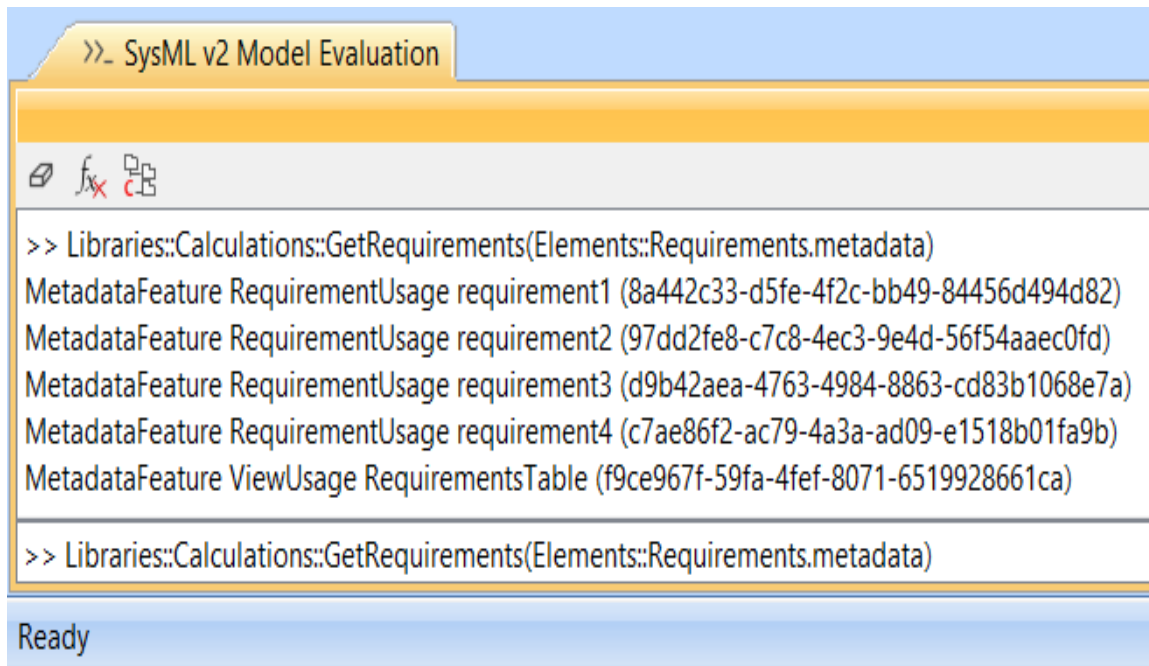
```
calc def GetRequirements {  
    in scope : Package [1];  
    ownedMembers = scope.ownedMember.? {  
        in x;  
        (x as Element).owningRelationship hastype  
            OwingMembership |  
            (x as Element).owningRelationship hastype  
                FeatureMembership  
    };  
    ownedMembers  
}
```

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 3: Getting Requirements (Cont.)

3. In the Model Evaluation Window, type:

```
Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```



The screenshot shows a software interface titled ">> SysML v2 Model Evaluation". Below the title bar is a toolbar with icons for a file, a function (fx), a red 'x', and a block definition. The main area contains a command prompt where the command `>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)` has been entered. The output of the command is displayed as a list of five items, each consisting of a metadata feature name, a requirement usage name, and a GUID. The status bar at the bottom of the window displays the word "Ready".

```
>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
MetadataFeature RequirementUsage requirement1 (8a442c33-d5fe-4f2c-bb49-84456d494d82)
MetadataFeature RequirementUsage requirement2 (97dd2fe8-c7c8-4ec3-9e4d-56f54aaec0fd)
MetadataFeature RequirementUsage requirement3 (d9b42aea-4763-4984-8863-cd83b1068e7a)
MetadataFeature RequirementUsage requirement4 (c7ae86f2-ac79-4a3a-ad09-e1518b01fa9b)
MetadataFeature ViewUsage RequirementsTable (f9ce967f-59fa-4fef-8071-6519928661ca)

>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```

Ready

Hands-On Activity 3: Getting Requirements (Cont.)

4. Add the following highlighted text:

```
calc def GetRequirements {  
    in scope : Package [1];  
    ownedMembers = scope.ownedMember.? {...};  
    ...  
    direct default ownedMembers.? {  
        in x;  
        x hastype RequirementUsage  
    };  
    direct  
}
```

Replace “ownedMembers”
from last function

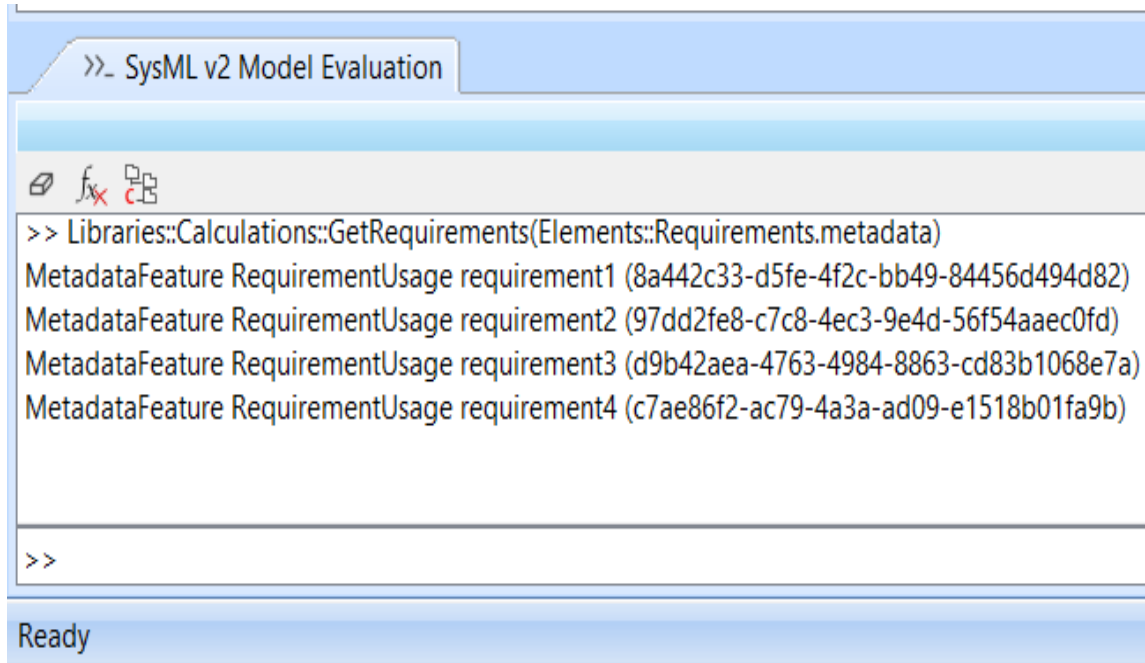
Note: Copy with formatting
for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 3: Getting Requirements (Cont.)

5. In the Model Evaluation Window, type:

```
Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```



Hands-On Activity 3: Getting Requirements (Cont.)

6. Add the following highlighted text:

```
calc def GetRequirements {  
    in scope : Package [1];  
    ownedMembers = scope.ownedMember.? {...};  
    direct default ownedMembers.? {...};  
    nested = ownedMembers.? {  
        in x;  
        x @Namespace & not (@VerificationCaseUsage)  
    };  
    nested  
}
```

Replace “direct”
with nested

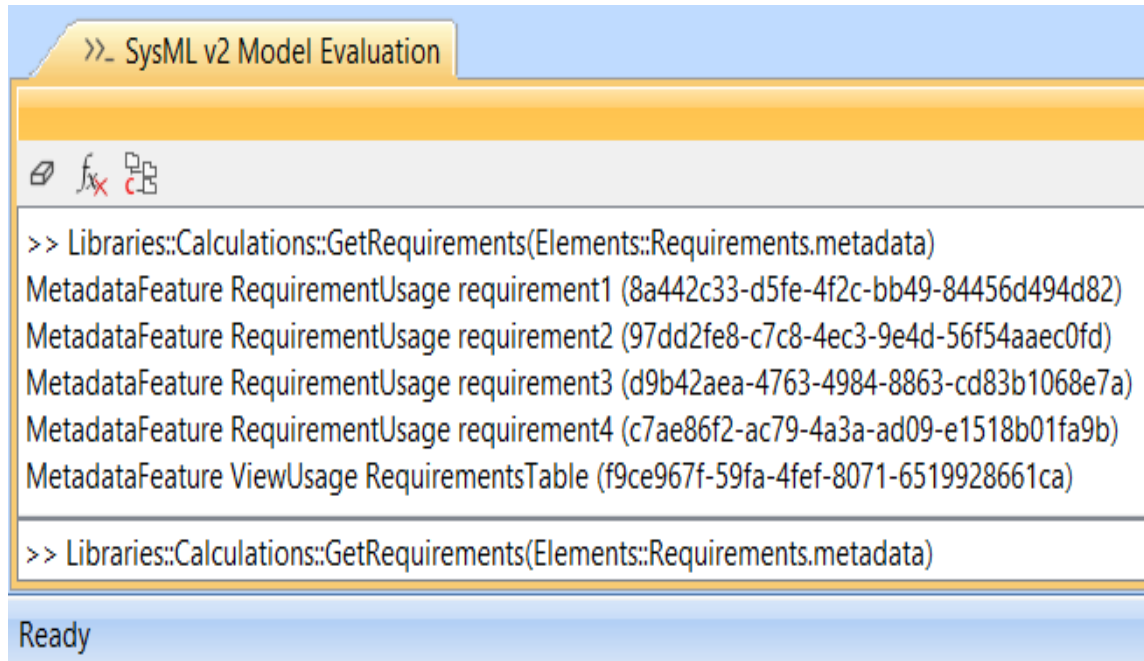
Note: Copy with formatting for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 3: Getting Requirements (Cont.)

7. In the Model Evaluation Window, type:

```
Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```



The screenshot shows a software interface titled ">> SysML v2 Model Evaluation". Below the title bar is a toolbar with icons for a file, a function (fx), a red 'x', and a block definition. The main area contains a command prompt with the following text:

```
>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)  
MetadataFeature RequirementUsage requirement1 (8a442c33-d5fe-4f2c-bb49-84456d494d82)  
MetadataFeature RequirementUsage requirement2 (97dd2fe8-c7c8-4ec3-9e4d-56f54aaec0fd)  
MetadataFeature RequirementUsage requirement3 (d9b42aea-4763-4984-8863-cd83b1068e7a)  
MetadataFeature RequirementUsage requirement4 (c7ae86f2-ac79-4a3a-ad09-e1518b01fa9b)  
MetadataFeature ViewUsage RequirementsTable (f9ce967f-59fa-4fef-8071-6519928661ca)  
  
>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```

At the bottom of the window, the status bar displays the word "Ready".

Hands-On Activity 3: Getting Requirements (Cont.)

8. Add the following highlighted text:

```
calc def GetRequirements {  
    in scope : Package [1];  
    ownedMembers = scope.ownedMember.? {...};  
    direct default ownedMembers.? {...};  
    nested = ownedMembers.? {...};  
    return result default union(direct,  
                                nested->collect GetRequirements);  
}
```

Replace “nested”
from last function

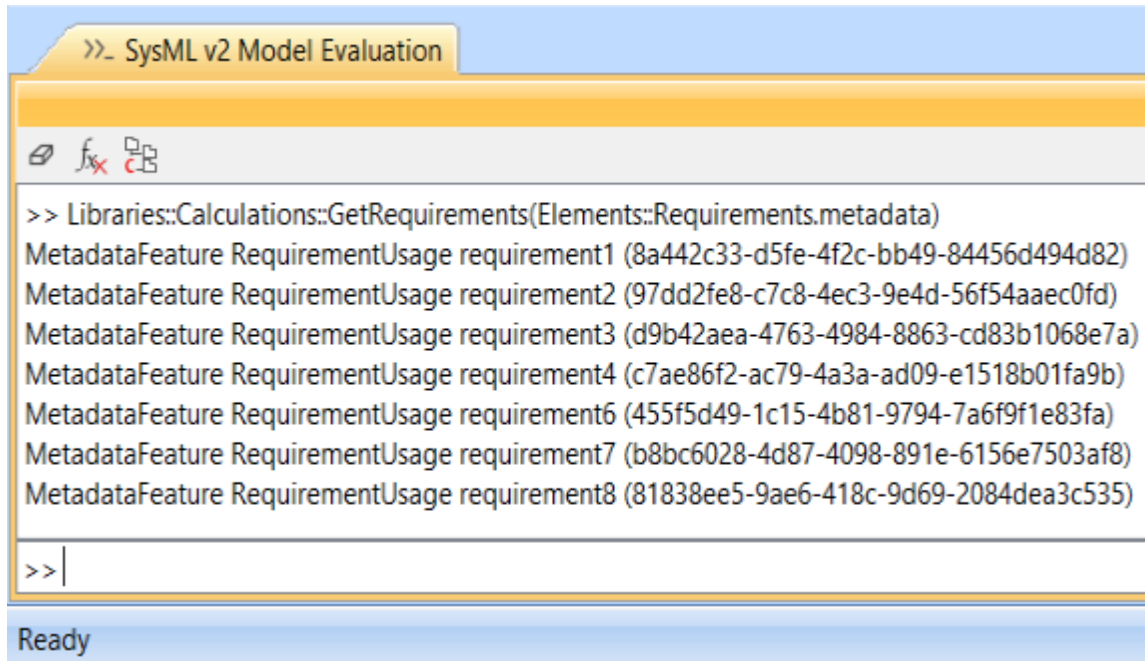
Note: Copy with formatting for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 3: Getting Requirements (Cont.)

9. In the Model Evaluation Window, type:

```
Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
```



The screenshot shows a software window titled ">> SysML v2 Model Evaluation". Inside the window, there is a command prompt area with a cursor. Below the prompt, the command `>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)` has been entered and executed. The output consists of eight lines, each starting with `MetadataFeature RequirementUsage` followed by a requirement ID in parentheses. The IDs are: `requirement1 (8a442c33-d5fe-4f2c-bb49-84456d494d82)`, `requirement2 (97dd2fe8-c7c8-4ec3-9e4d-56f54aaec0fd)`, `requirement3 (d9b42aea-4763-4984-8863-cd83b1068e7a)`, `requirement4 (c7ae86f2-ac79-4a3a-ad09-e1518b01fa9b)`, `requirement6 (455f5d49-1c15-4b81-9794-7a6f9f1e83fa)`, `requirement7 (b8bc6028-4d87-4098-891e-6156e7503af8)`, and `requirement8 (81838ee5-9ae6-418c-9d69-2084dea3c535)`. The window has a status bar at the bottom that says "Ready".

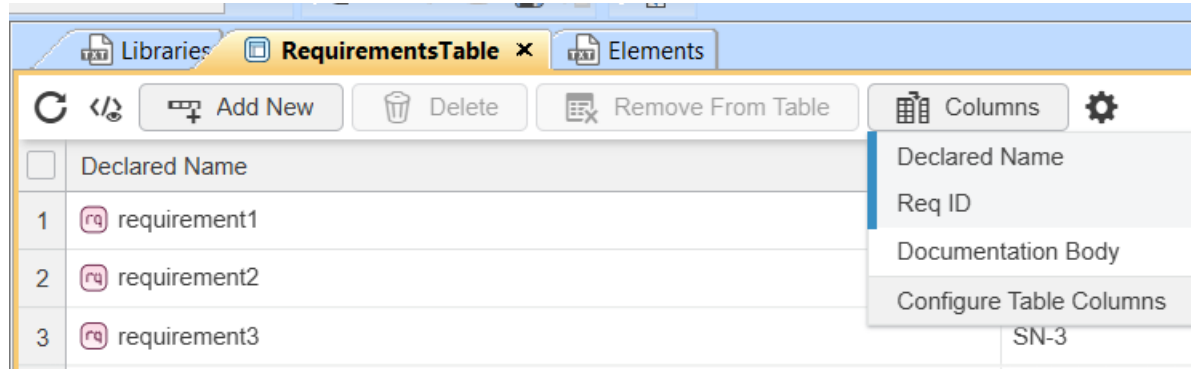
```
>> Libraries::Calculations::GetRequirements(Elements::Requirements.metadata)
MetadataFeature RequirementUsage requirement1 (8a442c33-d5fe-4f2c-bb49-84456d494d82)
MetadataFeature RequirementUsage requirement2 (97dd2fe8-c7c8-4ec3-9e4d-56f54aaec0fd)
MetadataFeature RequirementUsage requirement3 (d9b42aea-4763-4984-8863-cd83b1068e7a)
MetadataFeature RequirementUsage requirement4 (c7ae86f2-ac79-4a3a-ad09-e1518b01fa9b)
MetadataFeature RequirementUsage requirement6 (455f5d49-1c15-4b81-9794-7a6f9f1e83fa)
MetadataFeature RequirementUsage requirement7 (b8bc6028-4d87-4098-891e-6156e7503af8)
MetadataFeature RequirementUsage requirement8 (81838ee5-9ae6-418c-9d69-2084dea3c535)

>> |
```

Ready

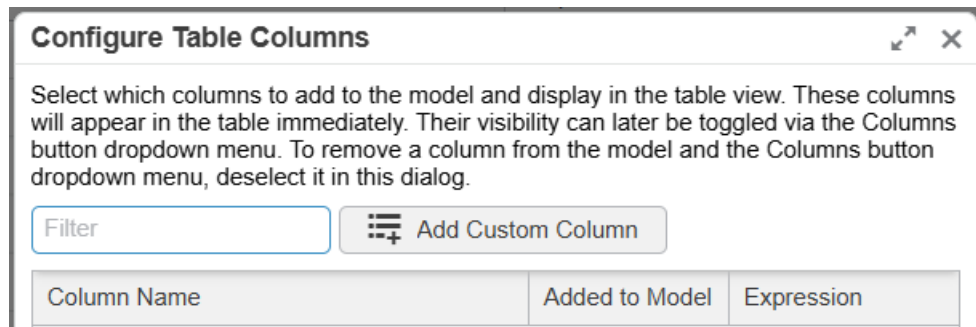
Hands-On Activity 3: Add Column

10. Click “Columns”
 - a. Select “Configure Table Columns”



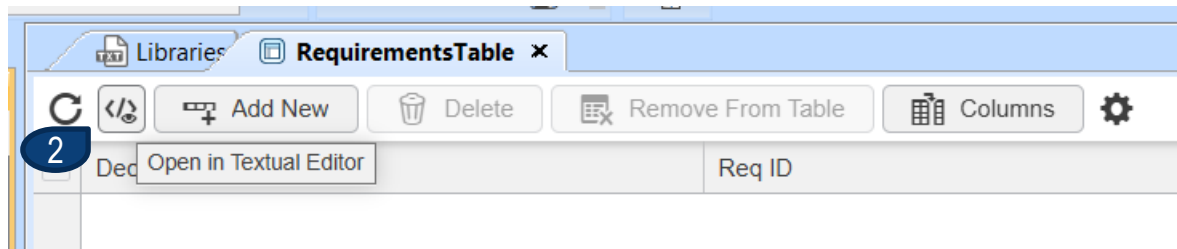
Hands-On Activity 3: Add Column

11. From the “Configure Table Columns” window, select “Add Custom Column”



12. Select OK

13. Open the Textual Editor



Hands-On Activity 3: Edit Columns

Note: you may need to select Refresh if the Textual Editor was already open.

Textual representation of model is outdated. Please refresh before continuing. [Refresh?](#)

- Focusing on the text below, notice this is the result of the custom column creation

```

13     requirement [$9] <'SN-4'> requirement4;
14     view [$10] RequirementsTable : rt {
15         expose Requirements::**;
16         render rendering [$11] asTable :>> asTable {
17             view [$12] :>> 'Declared Name';
18             view [$13] :>> 'Req ID';
19             view [$14] :>> 'Documentation Body' [0];
20             view [unnamed1 : ColumnByExpressionView :> column {
21                 render rendering [DS::StringCellRendering :>> DS_VIEWS::CoreViews::asTableCell {
22                     calc [$20] :>> getValue {
23                         in [$26] :>> rowElement;
24                         return [$27] :>> result = "";
25                     }
26                 }
27             }
28         }
29     }

```

Column Name

Column Type

Rendering Type (Return Type)

Calculated Expression

Hands-On Activity 3: Edit Column

15. Update the new calc by updating the highlighted sections below:

```
view Requirements : ColumnByExpressionView :> column {  
  render rendering : ElementCellRendering :>>  
    DS_Views::CoreViews::asTableCell {  
      calc :>> getValue {  
        in rowElement : SysML::RequirementUsage;  
        Libraries::Calculations  
          ::GetRequirements(rowElement.owner)  
      }  
    }  
}
```

Note: Copy with formatting for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 3: Review Table

16. Result:

Libraries RequirementsTable Elements				
Add New Delete Remove From Table Columns Settings				
☐	Declared Name	Req ID	HelloWorld	Requirements
2	requirement2	SN-2	Hello World	requirement <SN-2> requirement2 requirement <SN-3> requirement3 requirement <SN-4> requirement4 requirement <SN-1.1> requirement6 requirement <SN-3.1> requirement7 requirement <SN-3.1.1> requirement8
3	requirement3	SN-3	Hello World	requirement <SN-1> requirement1 requirement <SN-2> requirement2 requirement <SN-3> requirement3 requirement <SN-4> requirement4 requirement <SN-1.1> requirement6 requirement <SN-3.1> requirement7 requirement <SN-3.1.1> requirement8
4	requirement4	SN-4	Hello World	requirement <SN-1> requirement1 requirement <SN-2> requirement2 requirement <SN-3> requirement3 requirement <SN-4> requirement4 requirement <SN-1.1> requirement6 requirement <SN-3.1> requirement7 requirement <SN-3.1.1> requirement8
5	requirement6	SN-1.1	Hello World	requirement <SN-1.1> requirement6
6	requirement7	SN-3.1	Hello World	requirement <SN-3.1> requirement7 requirement <SN-3.1.1> requirement8
7	requirement8	SN-3.1.1	Hello World	requirement <SN-3.1.1> requirement8
Total Rows: 7 Total Visible Rows: 7 Selected Rows: 0				

Hands-On Activity 4: Getting Actions

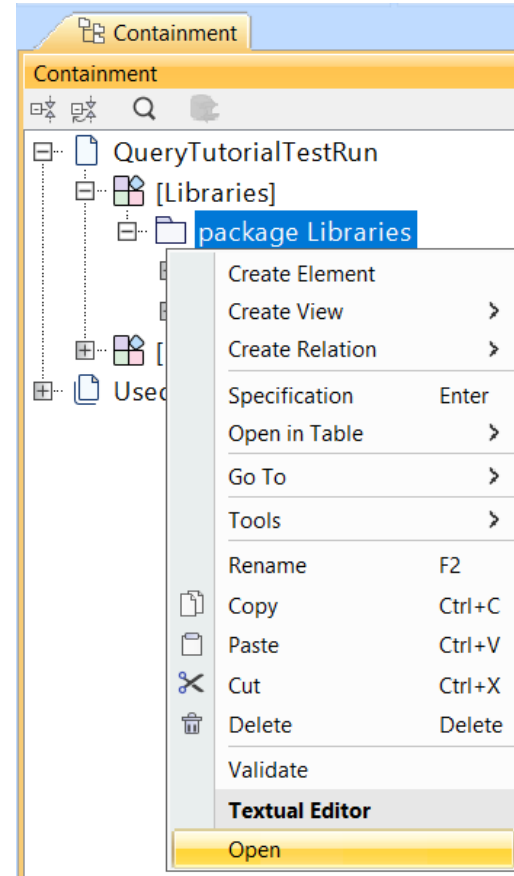
- ▶ Create a new calc def for counting actions by subclassification
- ▶ Create a new Behaviors Table
- ▶ Apply the new calc def to the table



Hands-On Activity 4: Getting Actions

1. Right Click>Open (Textual Editor)

Note: For the following steps we will be adding textual notation to the model. You will need to remember to Synchronize the textual notation each time in order for it to apply.



Hands-On Activity 4: Getting Actions

2. Add the following text under the package “Calculations”:

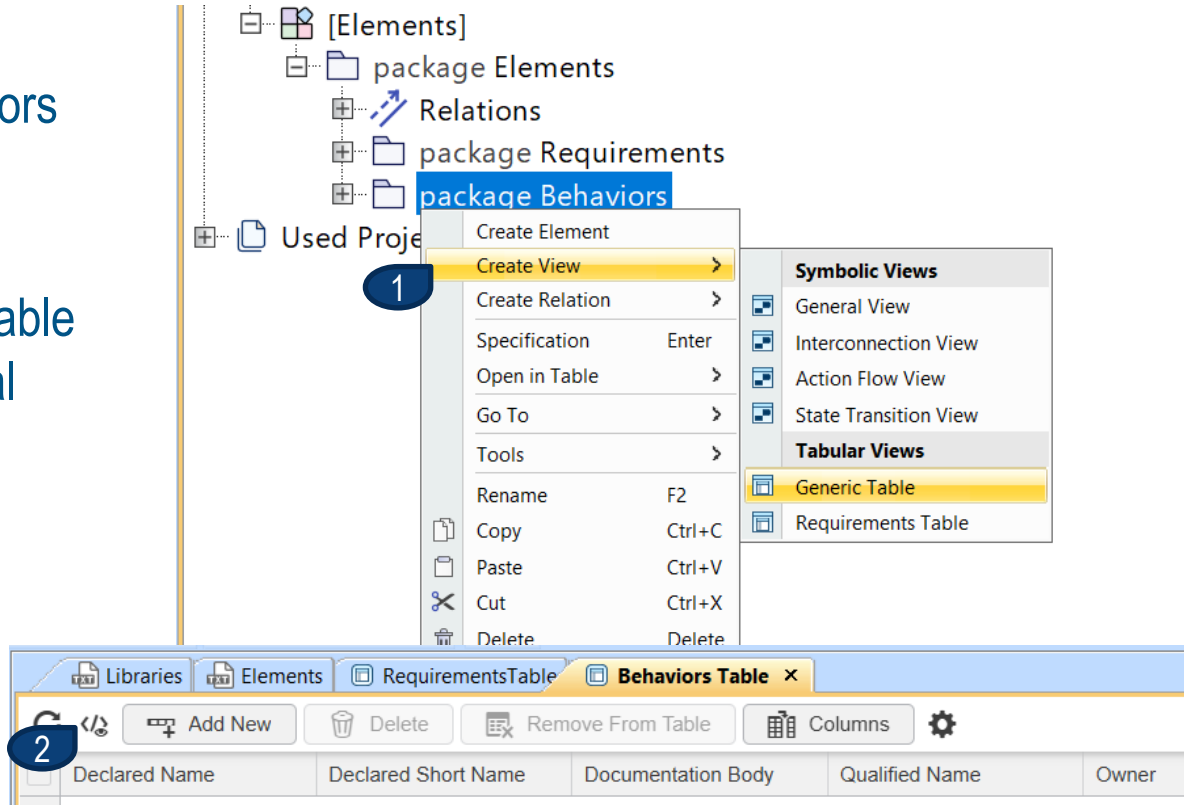
```
calc def GetActions :> GetRequirements {  
  :>> direct = ownedMembers.? {  
    in x;  
    x hastype ActionUsage  
  };  
  return result = union(direct,  
    nested->collect GetActions);  
}
```

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 4: Create a New Table

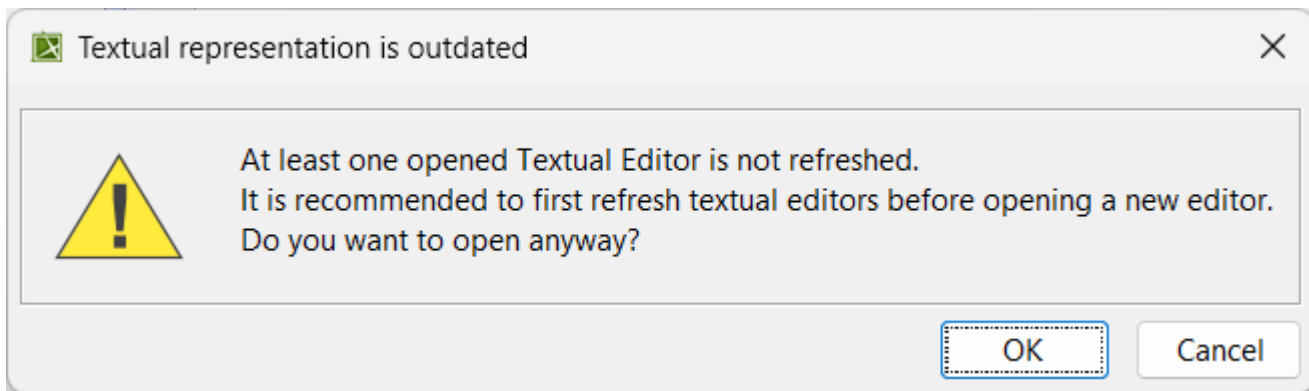
Create a new Behaviors Table

3. Right Click on the Behaviors package>Create View>Generic Table
4. Select the  from the Table Toolbar to Open in Textual Editor



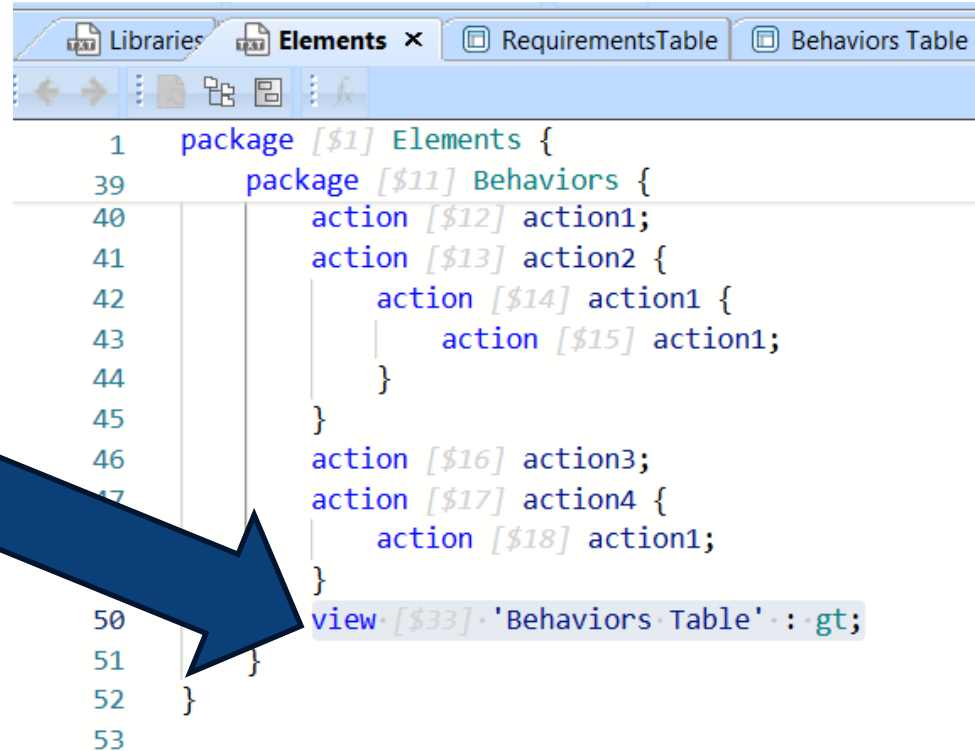
Hands-On Activity 4: Create a New Table

Note: You may receive this popup, it is because you edited the Textual Representation without synchronizing the model. Select OK.



Hands-On Activity 4: Create a New Table

The new table should be highlighted in the textual notation as shown here:



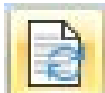
```
1 package [$1] Elements {
39 package [$11] Behaviors {
40     action [$12] action1;
41     action [$13] action2 {
42         action [$14] action1 {
43             action [$15] action1;
44         }
45     }
46     action [$16] action3;
47     action [$17] action4 {
48         action [$18] action1;
49     }
50 view [$33] 'Behaviors Table'::gt;
51 }
52 }
53 }
```

Hands-On Activity 4: Edit Table

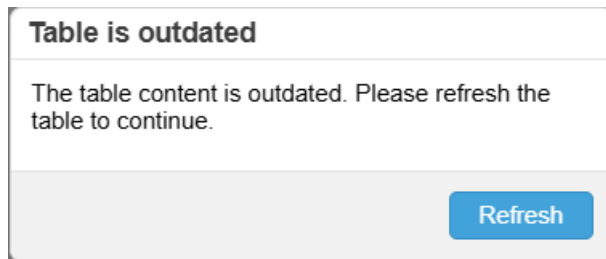
5. Expose the Behaviors package in order to set the scope of the table.

```
view BehaviorsTable : gt{  
    expose Behaviors::**;  
}
```

6. Click Synchronize Model with Textual Notation

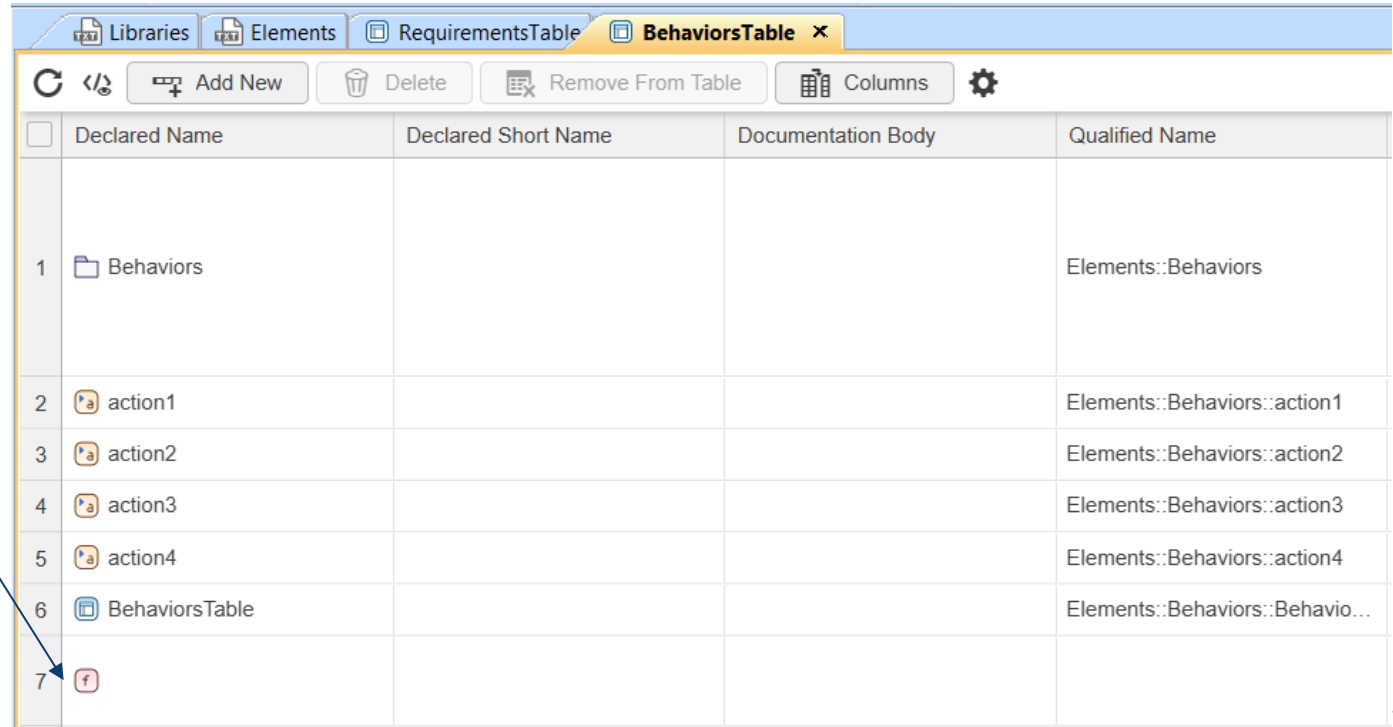


7. Go to the table tab and select Select Refresh



Hands-On Activity 4: Review Table

Table should look like this:



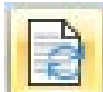
	Declared Name	Declared Short Name	Documentation Body	Qualified Name
1	 Behaviors			Elements::Behaviors
2	 action1			Elements::Behaviors::action1
3	 action2			Elements::Behaviors::action2
4	 action3			Elements::Behaviors::action3
5	 action4			Elements::Behaviors::action4
6	 BehaviorsTable			Elements::Behaviors::Behavio...
7				

Library Elements
To be filtered

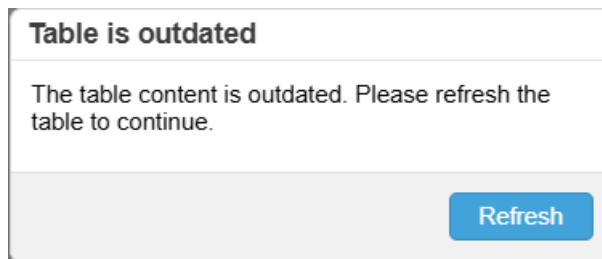
Hands-On Activity 4: Edit Table

8. Add the highlighted text to ignore library elements and anything that isn't an ActionUsage

```
view 'Behaviors Table' : gt,  
    NonStandardLibraryElementFilter {  
        expose Behaviors::**;  
        filter @ActionUsage & not (@CalculationUsage);  
    }
```

9.  Click Synchronize Model with Textual Notation

10. Go to the table tab and select Select Refresh



Tip: Press shift+alt+f to Format the Textual Notation after Pasting

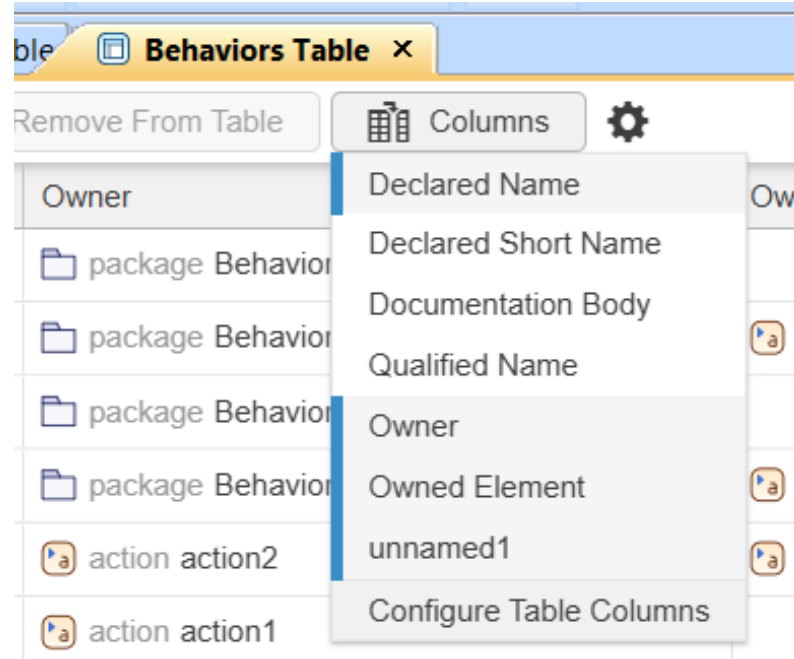
Hands-On Activity 4: Review Table

Table should look like this:

Libraries Elements RequirementsTable Behaviors Table x Refresh Code Add New Delete Remove From Table Columns Settings						
	Declared Name	Declared Short Name	Documentation Body	Qualified Name	Owner	Owned Element
1	 action1			Elements::Behaviors::action1	 package Behaviors	
2	 action2			Elements::Behaviors::action2	 package Behaviors	 action action1
3	 action3			Elements::Behaviors::action3	 package Behaviors	
4	 action4			Elements::Behaviors::action4	 package Behaviors	 action action1
5	 action1			Elements::Behaviors::action2::...	 action action2	 action action1
6	 action1			Elements::Behaviors::action2::...	 action action1	
7	 action1			Elements::Behaviors::action4::...	 action action4	

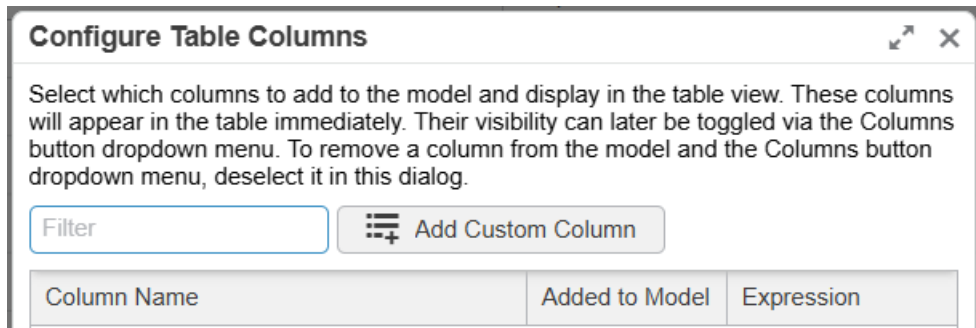
Hands-On Activity 4: Edit Columns

11. Click “Columns”
 - a. Select “Configure Table Columns”
 - b. Deselect “Declared Short Name”, “Documentation Body”, and “Qualified Name”



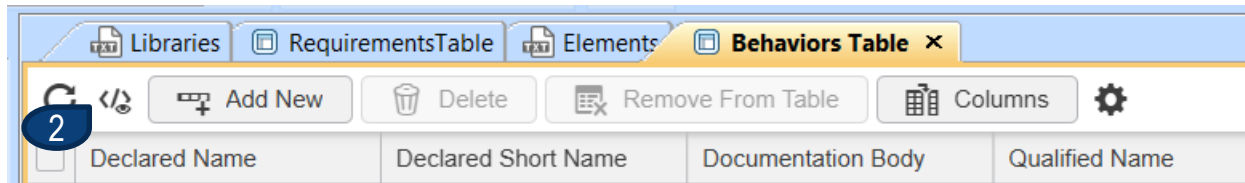
Hands-On Activity 4: Add Column

12. From the “Configure Table Columns” window, select “Add Custom Column”



13. Select OK

14. Open the Textual Editor



Hands-On Activity 4: Edit Column

15. Update the new calc by updating the highlighted sections below:

```
view Actions : ColumnByExpressionView :> column {  
  render rendering : ElementCellRendering :>>  
    DS_Views::CoreViews::asTableCell {  
      calc :>> getValue {  
        in rowElement : SysML::ActionUsage;  
        Libraries::Calculations  
          ::GetActions(rowElement.owner)  
      }  
    }  
}
```

Note: Copy with formatting for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 4: Review Table

16. Remove Unwanted Columns using the Column dialog

Libraries RequirementsTable Elements Behaviors Table x				
↺ ↻ Add New Delete Remove From Table Columns ⚙				
☐	Declared Name	Owner	Owned Element	Actions
2	 action2	 package Behaviors	 action action1	 action action1  action action2  action action3  action action4  action action1  action action1  action action1

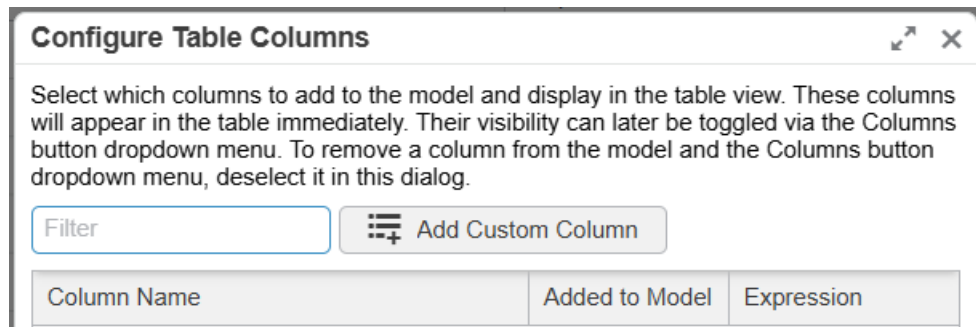
Hands-On Activity 5: Counting Elements

- ▶ Create a new calc def for counting actions by subclassification
- ▶ Apply the new calc def to the table

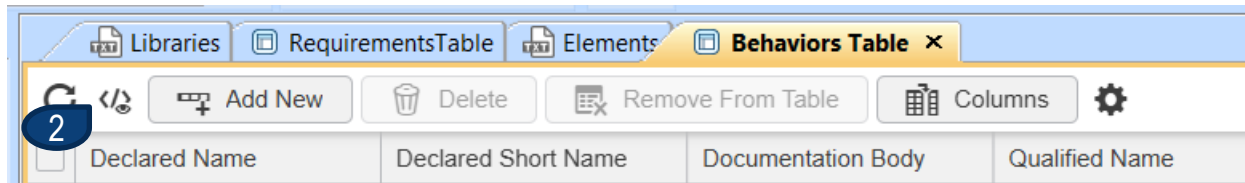


Hands-On Activity 5: Add Column

1. From the “Configure Table Columns” window, select “Add Custom Column”



2. Select OK
3. Open the Textual Editor



Hands-On Activity 5: Edit Count Column

4. Update the new calc by updating the highlighted sections below:

```
view NumOfActions : ColumnByExpressionView :> column {  
  render rendering : IntegerCellRendering :>>  
  DS_Views::CoreViews::asTableCell {  
    calc :>> getValue {  
      in rowElement : SysML::ActionUsage;  
      size(Libraries::Calculations  
        ::GetActions(rowElement.owner))  
    }  
  }  
}
```

Note: Copy with formatting for best results

Tip: Press shift+alt+f to Format the Textual Notation after Pasting

Hands-On Activity 5: Review Column

5. Remove Unwanted Columns using the Column dialog

Libraries Elements RequirementsTable Behaviors Table x					
Add New Delete Remove From Table Columns					
	Declared Name	Owner	Owned Element	Actions	NumOfActions
1	action1	package Behaviors		action action1 action action2 action action3 action action4 action action1 action action1 action action1	7
2	action2	package Behaviors	action action1	action action1 action action2 action action3 action action4 action action1 action action1 action action1	7

Future Work

- ▶ REST API
- ▶ Open API
- ▶ Recreating Other Queries Based on Customer Feedback
- ▶ Extending to Other Metrics
- ▶ Building Validation Suites

Sources

- ▶ Object Management Group. (2026, March). *System Modeling Language (OMG SysML®) specification* (Version 2.0, Formal/26-03-02). <https://www.omg.org/spec/SysML/2.0>
- ▶ Object Management Group. (2026, March). *Kernel Modeling Language (KerML) specification* (Version 1.0, Formal/26-03-01). <https://www.omg.org/spec/KerML/1.0>
- ▶ Strategic Technology Consulting. (n.d.). *From structured expressions to KerML notation* [Technical training documentation / Presentation]. Strategic Technology Consulting, An Arcfield Company. CUSA 2026, Reno Nevada
- ▶ Densford, S., & Jankauskas, O. (2025). *Querying the Model in SysML V2* [Presentation / Tutorial]. INCOSE International Symposium 2025, Ottawa, Canada.

BACKUP

OPERATOR MAPPING

Table 5. Operator Mapping

Operator	Library Function	Description	Model-Level Evaluable?
all	BaseFunctions::'all'	Type extent	No
istype	BaseFunctions::'istype'	All argument values are directly or indirectly instances of a type	Yes
hastype	BaseFunctions::'hastype'	All argument values are directly instances of a type	Yes
@	BaseFunctions::'@'	Any argument value is directly or indirectly an instance of a type	Yes
@@	BaseFunctions::'@@'	Any argument value is directly or indirectly an instance of a metaclass	Yes
as	BaseFunctions::as	Select instances of type (cast)	Yes
meta	BaseFunctions::meta	Select instances of a metaclass (metacast)	Yes
==	BaseFunctions::'=='	Equality	Yes
!=	BaseFunctions::'!='	Inequality	Yes
===	BaseFunctions::'==='	Same (equality for data values, same lives for occurrences)	Yes
!==	BaseFunctions::'!=='	Not same	Yes
xor	DataFunctions::'xor'	Logical "exclusive or"	Yes
not	DataFunctions::'not'	Logical "not"	Yes
~	DataFunctions::'~'	Undefined	No
 	DataFunctions::' '	Logical "inclusive or"	Yes
&	DataFunctions::'&'	Logical "and"	Yes
<	DataFunctions::'<'	Less than	Yes

OPERATOR MAPPING (CONT.)

Operator	Library Function	Description	Model-Level Evaluable?
>	DataFunctions::'>'	Greater than	Yes
<=	DataFunctions::'<='	Less than or equal to	Yes
>=	DataFunctions::'>='	Greater than or equal to	Yes
+	DataFunctions::'+'	Addition	Yes
-	DataFunctions::'-'	Subtraction	Yes
*	DataFunctions::'*'	Multiplication	Yes
/	DataFunctions::'/'	Division	Yes
%	DataFunctions::'%'	Remainder	Yes
^ **	DataFunctions::'^'	Exponentiation	Yes
..	DataFunctions::'..'	Range construction	Yes
??	ControlFunctions::'??'	Null coalescing	Yes
if	ControlFunctions::'if'	Conditional test (ternary)	Yes
or	ControlFunctions::'or'	Conditional "or"	Yes
and	ControlFunctions::'and'	Conditional "and"	Yes
implies	ControlFunctions::'implies'	Conditional "implication"	Yes

- *Classification expressions.* The *classification operators* are syntactically similar to binary operators, but, instead of an expression as their second operand, they take a type name. The classification operators **istype** and **hastype** test whether all of the values of their first operand is classified by the named type (either including or not including subtypes, respectively). The **@** operator is similar to **istype**, but tests whether at least one of the values of its first operand is classified by the named type. Note that this means that **istype** and **hastype** evaluate to true on a **null** (empty list) value, while **@** evaluates to false.

```
sensors istype ThermalSensor // Are all sensors ThermalSensors?
sensors @ ThermalSensor // Is any sensor a ThermalSensor?
person hastype Administrator
```

The classification operator **as**, known as the *cast operator*, performs an **isType** test of whether each of the values of its first operand is classified by the named type, and then it selects only those values that pass the test to include in its result. The result values of such a cast expression (if any) are always guaranteed to be instances of the named type.

```
sensors as ThermalSensor
person as Administrator
```

The classification operators may also be used without a first operand, in which case the first operand is implicitly `Anything::self` (see [9.2.2.2.1](#)). This is useful, in particular, when used as a test within an element filter condition expression (see [7.4.14](#)).

```
istype ThermalSensor
@ThermalSensor
hastype Administrator
as Supervisor
```

► Metaclassification Expressions

- *Metaclassification expressions.* The metaclassification operators `@@` and `meta` take the qualified name of any kind of element as their first operand and a metaclass (see [7.4.13](#)) as their second operand. They are shorthands for classification expressions with the operators `@` and `as`, respectively, and a metadata access expression (see [7.4.9.4](#)) as their first operand. As such, `@@` tests whether any metadata associated with an element are classified by the given metaclass, while `meta` filters the metadata associated with an element and evaluates to those that are classified by the given metaclass.

```
// Shorthand for designModel.metadata @ ApprovalAnnotation  
designModel @@ ApprovalAnnotation
```

```
// Shorthand for sensors.metadata as KerML::Feature  
sensors meta KerML::Feature
```

```
// Evaluates to the string "sensors".  
(sensors meta KerML::Feature).name
```

1. Index Expression (#)

- Purpose:** Accesses specific elements from a sequence or collection.
- Syntax:** source#(index)
- Key Rules:**
 - Default indexing starts at 1.
 - For multi-dimensional Arrays, the index can be a sequence: array#(n, m).

2. Sequence Expression (,)

- Purpose:** Concatenates multiple expressions into a single "flat" sequence.
- Syntax:** (expr1, expr2, ...)
- Key Rules:**
 - Sequences are always **flat**; nesting sequences ((1, 2), 3) results in a single sequence (1, 2, 3).
 - The empty sequence () represents a null expression.

3. Feature Chain Expression (.)

- Purpose:** Evaluates a feature (property or behavior) for every element in a sequence.
- Syntax:** primaryExpression.featureName
- Key Rules:**
 - It maps the feature over the entire sequence and collects the results into a new sequence.
 - Example: platforms.sensors returns all sensors across all platforms.

4. Collect Expression (.)

- Purpose:** Transforms elements of a sequence using an expression body (shorthand for a "map" function).
- Syntax:** `source.{in x; transformation_logic}`
- Key Rules:** Evaluates the body for each element and collects the results in order.

5. Select Expression (.)

- Purpose:** Filters a sequence based on a Boolean condition.
- Syntax:** `source.?[in x; condition_logic]`
- Key Rules:** Only elements where the condition evaluates to true are included in the resulting subsequence.

6. Function Operation Expression (->)

- Purpose:** Supports functional chaining (data flow) by passing the result of one expression as the first argument of the next.
- Syntax:** `source -> functionName(remaining_args)`
- Key Rules:**
 - Useful for readable pipelines: `data -> filter(x) -> sort()`.
 - Supports passing expression bodies directly for control functions like `reduce` or `reject`.
 - Can use the shorthand factors `-> reduce RealFunctions::"*"` to pass a function by name.

- *Function operation expression.* A function operation expression is a special syntax for an invocation expression in which the first argument is given before the arrow (`->`) symbol, which is followed by the name of the function to be invoked and an argument list for any remaining arguments (see [7.4.9.4](#)). This is useful for chaining invocations in an effective data flow.

```
sensors -> selectSensorsOver(limit) -> computeCriticalValue()
```

If the invoked function has exactly two input parameters, and the second input parameter is an expression, then an expression body (see [7.4.9.4](#)) can be used as the argument for the second argument without surrounding parentheses. The argument expression body should declare parameters consistent with those on the parameter expression (if any). This is particularly useful when invoking functions from the `ControlFunctions` library model (see [9.4.17](#)).

```
sensors -> select {in s: Sensor; s::isActive}  
members -> reject {in member: Member; not member->isInGoodStanding()}  
factors -> reduce {in x: Real; in y: Real; x * y}
```

If the argument expression is simply the direct invocation of another function, then the argument expression may be specified using simply the name of the invoked function.

```
factors -> reduce RealFunctions::'*'
```