MapleMBSE 2025.0 Virtual Features Guide

Copyright © Maplesoft, a division of Waterloo Maple Inc.
2025

MapleMBSE 2025.0 Virtual Features Guide

Contents

Preface xi
1 Introduction 1
1.1 Scope and Purpose of this Document 1
1.2 Prerequisite Knowledge 1
1.3 Motivation for Using MapleMBSE Virtual Features 1
1.4 Importing the MapleMBSE Ecore 3
1.5 General Syntax for the MapleMBSE Virtual Features 3
2 Stereotypes 5
2.1 metaclassName 5
Description 5
Syntax 5
Using the metaclassName Virtual Feature 5
Example 6
2.2 featureName 6
Description 6
Syntax 7
Using the featureName Virtual Feature 7
Example 8
2.3 stereotypeNames 9
Description 9
Syntax 9
Using the stereotypeNames Virtual Feature 9
Example 10
3 Associations 11
3.1 associatedProperty 11
Description 11
Syntax 11
Using the associatedProperty Virtual Feature 12
Example 13
3.2 directedAssociatedProperty 14
Description 14
Syntax 14
Using the directAssociatedProperty Virtual Feature 14
Example 15
3.3 otherAssociatedEnd 16
Description 16
Syntax 16
Using the otherAssociatedEnd Virtual Feature 16
Example 17
3.4 nestedDirectedComposition 19
Description 19

Syntax ... 19
Using the nestedDirectedComposition virtual feature ... 19
Example ... 20
4 Blocks ... 21
4.1 recursivePartProperties ... 21
Description ... 21
Syntax ... 21
Using the recursivePartProperties Virtual Feature ... 21
Example ... 22
4.2 propertyDefaultValue ... 22
Description ... 22
Syntax ... 22
Using the propertyDefaultValue Virtual Feature ... 22
Example ... 22
4.3 getAllProperties ... 23
Description ... 23
Syntax ... 25
Using the getAllProperties virtual feature ... 25
Example ... 26
5 Connectors ... 27
5.1 connectedPropertyOrPort ... 27
Description ... 27
Syntax ... 27
Using the connectedPropertyOrPort virtual feature ... 28
Example ... 28
5.2 otherConnectorEnd ... 29
Description ... 29
Syntax ... 29
Using the otherConnectorEnd Virtual Feature ... 29
Example ... 29
6 Dependencies ... 31
6.1 clientDependencies ... 31
Description ... 31
Syntax ... 31
Using the clientDependencies Virtual Feature ... 31
Example ... 31
6.2 supplierDependencies ... 32
Description ... 32
Syntax ... 33
Using the supplierDependencies Virtual Feature ... 33
Example ... 34
6.3 featureImpact ... 34
Description ... 34

Syntax 35
Using the featureImpact Virtual Feature 35
Example 35
6.4 Multiple Dependencies Class 36
Introduction 36
Creating a Multiple Dependencies Class in an MSE file 36
7 Enumeration 37
7.1 EnumerationName 37
Description 37
Syntax 37
Using the enumerationName Virtual Feature 37
Example 38
7.2 EnumerationLabel 38
Description 38
Syntax 38
8 Util 41
8.1 multiplicityProperty 41
Description 41
Syntax 41
Using the multiplicityProperty Virtual Feature 41
Example 42
9 Activity Diagrams 43
9.1 ActivityControlFlow 43
Description 43
Syntax 43
Using the ActivityControlFlow Virtual Feature 43
Example 44
9.2 ActivityObjectFlow 45
Description 45
Syntax 45
Using the ActivityObjectFlow Virtual Feature 45
Example 46
10 StateMachines 47
10.1 VertexTransition 47
Description 47
Syntax 47
Using the VertexTransition Virtual Feature 47
Example 48
10.2 VerticalTransition 48
Description 48
Syntax 49
Using the VertexTransition Virtual Feature 49
11 Comments 51

11.1 ownedComments 51
Description 51
Syntax 51
Using the ownedComments Virtual Feature 51
Example 52
12 Instance Matrices 55
12.1 SlotValue 55
Description 55
Syntax 55
Using the SlotValue Virtual Feature 56
Example 57
12.2 InstanceTree 57
Description 57
Syntax 58
Using the InstanceTree Virtual Feature 58
Example 59
12.3 InstanceWithSlots 60
Description 60
Syntax 60
Using the InstanceWithSlots Virtual Feature 60
Example 61
12.4 RecursiveInstanceWithSlots 61
Description 61
Syntax 62
Using the RecursiveInstanceWithSlots Virtual Feature 62
Example 63
12.5 AttachedFile 63
Description 63
Syntax 63
Using the attachedFile Virtual Feature 64
Example 64
12.6 Slots 64
Description 64
Syntax 64
Using the slots Virtual Feature 64
Example 65
12.7 ArrayName 65
Description 65
Syntax 65
Using the arrayName Virtual Feature 65
Example 66
12.8 MultiplicityOfInstance 66
Description 66

Syntax 67
Using the multiplicityOfInstance Virtual Feature 67
Example 67
13 Recursivity 69
13.1 getRecursively 69
Description 69
Syntax 69
Using the getRecursively Virtual Feature 70
Example 70
14 Constraints 73
14.1 durationConstraint 73
Description 73
Syntax 73
Using the durationConstraint Virtual Feature 74
Example 75
15 Generalization 77
15.1 specificClass 77
Description 77
Syntax 77
Using the specificClass Virtual Feature 77
Example 77
16 Working with sysML Diagrams 79
16.1 downloadDiagram 79
Description 79
Syntax 79
Using the clientDependencies Virtual Feature 79
Example 79
16.2 diagramType 80
Description 80
Syntax 80
Using the supplierDependencies Virtual Feature 80
Example 80
17 File Attachments 81
17.1 AttachedFile 81
Description 81
Syntax 81
Using the attachedFile Virtual Feature 81
Example 81
18 Element Type 83
18.1 elementType 83
Description 83
Syntax 83
Using the elementType Virtual Feature 83

Example 83
Index 85

List of Figures

Figure 2.1: metaclassName Example 6
Figure 2.2: The appliedStereotypeInstance Structure 7
Figure 2.3: featureName Example 8
Figure 2.4: stereotypeNames Example 10
Figure 3.1: associatedProperty Example 13
Figure 3.2: directAssociatedProperty Example 15
Figure 3.3: otherAssociatedEnd Example 18
Figure 5.1: connectedPropertyOrPort Example 28
Figure 5.2: otherConnectorEnd Example 30
Figure 6.1: clientDependencies Example 32
Figure 6.2: supplierDependencies Example 34
Figure 8.1: multiplicityProperty Example 42
Figure 9.1: ActivityControlFlow Example 44
Figure 9.2: ActivityObjectFlow Example 46
Figure 10.1: VertexTransition Example 48

Preface

MapleMBSE Overview

MapleMBSE™ gives an intuitive, spreadsheet based user interface for entering detailed system design definitions, which include structures, behaviors, requirements, and parametric constraints.

Related Products

MapleMBSE 2025 requires the following products.

- Microsoft® Excel® 2016, Excel 2019 or Excel Office 365 desktop.
- Oracle® Java® SE Runtime Environment 8.

Note: MapleMBSE looks for a Java Runtime Environment in the following order:

1) If you use the -vm option specified in **OSGiBridge.init** (not specified by default), MapleMBSE will use it.

2) If your environment has a system JRE (meaning either: JREs specified by the environment variables JRE_HOME and JAVA_HOME in this order, or a JRE specified by the Windows Registry (created by JRE installer)), MapleMBSE will use it.

3) The JRE installed in the MapleMBSE installation directory.

- Teamwork Cloud™ server 2021.x, 2022.x and 2024.x
- Magic Collaboration Studio 2021.x, 2022x and 2024.x

If you are using Eclipse Capella™ with MapleMBSE, the following version is supported:

- 6.x

If you are using Eclipse™, the following version is supported:

- 2024-3

Related Resources

Resource	Description
MapleMBSE Installation Guide	System requirements and installation instructions for MapleMBSE. The **MapleMBSE Installation Guide** is available in the **Install.html** file located in the folder where you installed MapleMBSE, or on the website. **https://www.maplesoft.com/documentation_center/**
MapleMBSE Applications	Applications in this directory provide a hands on demonstration of how to edit and construct models using MapleMBSE. They, along with an accompanying guide, are located in the Application subdirectory of your MapleMBSE installation.
MapleMBSE Configuration Guide	This guide provides detailed instructions on working with configuration files and the configuration file language.
MapleMBSE User Guide	Instructions for using MapleMBSE software. The **MapleMBSE User Guide** is available in the folder where you installed MapleMBSE.
Frequently Asked Questions	You can find MapleMBSE FAQs here: https://faq.maplesoft.com
Release Notes	The release notes contain information about new features, known issues and release history from previous versions. You can find the release notes in your MapleMBSE installation directory.

For additional resources, visit **http://www.maplesoft.com/site_resources**.

Getting Help

To request customer support or technical support, visit **http://www.maplesoft.com/support**.

Customer Feedback

Maplesoft welcomes your feedback. For comments related to the MapleMBSE product documentation, contact doc@maplesoft.com.

1 Introduction

1.1 Scope and Purpose of this Document

The purpose of the MapleMBSE Virtual Features Guide is to describe MapleMBSE virtual features and explain how to use them.

The intended audience for this document are users who are familiar with UML, SysML and Model-based Systems Engineering concepts and who intend to create their own MapleMBSE configuration files.

1.2 Prerequisite Knowledge

To fully understand the information presented in this document the reader should be familiar with the following concepts:

- The Eclipse Modeling Framework `ecore` serialization. In particular, knowing how to use any tool of your choice to track all the *eReferences* independently of the *eSuperTypes*.
- Thus, some basic concepts of Meta Object Facility like *eClassifiers* and *eStructuralFeatures*. A correct mse configuration file has within each qualifier a concrete UML *eClassifiers* and each dimension should be accessed using a non-derived *StructuralFeature* defined in the `UML.ecore` or a virtual one inside this guide.
- MapleMBSE Configuration Language elements (especially dimension and qualifiers, and the syntax for importing the MapleMBSE ecore). For more information on the MapleMBSE Configuration language, see the **MapleMBSE Configuration Guide**.

1.3 Motivation for Using MapleMBSE Virtual Features

SysML provides a high level of abstraction to cover as many modeling scenarios as possible with the diagrams offered. It is a powerful and complex language that is extremely difficult to master because of its complexity (there are hundreds of pages of technical specifications for SysML).

Many different concrete and abstract *Classifiers*, with very specific semantics, are part of the SysML technical specifications. These *Classifiers* should not be used interchangeably. Even "linking" elements changes depending on the "linked" elements. For example, SysML *Associations* are to *Classes* as *Connectors* are to Ports, or, what *ControlFlows* can be for *ActivityNodes*. However, these elements are not interchangeable.

An end user, defined as a user who will be updating model information using the MapleMBSE spreadsheet interface but likely will not be involved in creating or editing configuration files, who interested in taking advantage of the modeling capabilities of SysML, should not need to know its complexities. MapleMBSE helps to hide this complexity

from the end user, through virtual features. They are called virtual features because, although they extend the capabilities of native SysML, they themselves are not part of SysML.

With the right choice of labels within an Excel template and a well designed configuration (.mse) file that implements MapleMBSE virtual features, an end user can enter a couple of inputs in a spreadsheet and create *Blocks* and the Associations linking them, or Ports and Connectors, or other combinations of elements.

For example, consider the following code snippet from a MapleMBSE configuration file in the figure below. This figure illustrates the scenario where a configuration file is designed without the use of virtual features to represent SysML *Associations* between *Blocks*.

Notice in the generated Excel worksheet, the number of inputs required of the end user to represent the *Association* between **Customer** and **Product**. This requires knowledge of SysML on the part of the end user.

```
1  synctable-schema Schema2(bsc: BlockSchema, asc: AssociationSchema) {
2      record dim [Model] {
3          key column /name as mName
4      }
5
6      alternative {
7          group {
8              record dim /packagedElement[Class | mse::metaclassName="SysML::Blocks::Block"] {
9                  key column /name as cName
10             }
11
12             record dim /ownedAttribute[Property] {
13                 key column /name as pName
14                 reference-query .type @ typeRef
15                 reference-decomposition typeRef = bsc {
16                     foreign-key column blockName as blockRef
17                 }
18                 reference-query .association @ associationRef
19                 reference-decomposition associationRef = asc {
20                     foreign-key column associationName as ascRef
21                 }
22             }
23         }
24         record dim /packagedElement[Association] {
25             key column /name as aName
26         }
27     }
28 }
29
30 worksheet-template Template2(sch: Schema2) {
31     vertical table tab1 at (2, 1) = sch {
32         key field mName : String
33         key field cName : String
34         key field aName : String
35         key field pName : String
36         key field blockRef : String
37         key field ascRef : String
38     }
39 }
```

Creating a configuration file without MapleMBSE virtual features results in an excel file that requires more input from the end user and requires the end user to know SysML to add elements to the spreadsheet.

Package	Block name	Association name	Property name	Name of property's type	name of property's association
Model					
Model	Customer				
Model	Product				
Model		purchases			
Model	Customer		boughtItem	Product	purchases
Model	Product		buyer	Customer	purchases

Now consider an example that represents the same Association between Customer and Product, as shown in the figure below. This time, the configuration file is designed using the MapleMBSE virtual features, specifically, the associatedProperty virtual feature. Notice, the only inputs required of the end user are the two SysML *Blocks*, **Customer** and **Product**. The cross-references need for the Association are completed automatically.

```
1  synctable-schema Schema(bsc: BlockSchema) {
2      record dim [Class | mse::metaclassName="SysML::Blocks::Block"] {
3          key column /name as cName
4      }
5
6      dim /ownedAttribute[Property].mse::associatedProperty[Class] @ classRef {
7      reference-decomposition classRef = bsc {
8              foreign-key column blockName as refCName
9          }
10     }
11 }
12
13 worksheet-template Template(sch: Schema) {
14     vertical table tab1 at (2, 1) = sch {
15         key field cName : String
16         key field refCName : String
17     }
18 }
```

Creating a configuration file that uses MapleMBSE virtual features results in an excel file that requires much less input from the end user and the end user does not need to know uml to create Association.

Block	Target block
Product	
Product	Customer
Customer	
Customer	Product

1.4 Importing the MapleMBSE Ecore

Loading MapleMBSE virtual features is analogous to the way you would load UML Structural Features using UML Ecore. The corresponding MapleMBSE Configuration language uses `import-ecore`.

The general syntax is

import-ecore "URI"

For example, to specify the NoMagic ecore:

"http://www.nomagic.com/magicdraw/UML/2.5"

To specify the MapleMBSE ecore:

`"http://maplembse.maplesoft.com/common/1.0"`

You must create an alias for the `ecore` using the syntax:

import-ecore "URI" as Alias

For example, to specify an alias for the MapleMBSE `ecore`:

```
import-ecore "http://maplembse.maplesoft.com/common/1.0" as
mse
```

This allows you to use the short form, `mse`, instead of the whole syntax.

1.5 General Syntax for the MapleMBSE Virtual Features

The general syntax for the virtual features is

`[./]?alias::virtualfeature`

The first character can be a dot, a forward slash, or a blank. There is no strict rule of thumb for this. For specific syntax, see the Syntax subsection for each virtual feature.

`alias` – This is the alias for the ecore import

`virtualfeature` - This is the virtual feature name you want to use, for example, `associatedProperty`.

2 Stereotypes

SysML can be explained as a subset of elements defined in the UML specifications plus some additional features not included in UML. One of these features is a *Stereotype*. *Stereotypes* are applied to those elements adding extra meaning or modeling semantics. MapleMBSE offers several virtual features to apply *Stereotypes* and navigate their extended modeling capacities.

2.1 metaclassName

Description

Use the **metaclassName** virtual feature to apply *Stereotypes* while creating elements using MapleMBSE. To use this virtual feature you need to identify the qualified name of the *Stereotype* that you want to apply and whether the element is compatible with that stereotype.

Syntax

Any *Element* of the *Model* can have a list of *appliedStereotype* but only certain *Stereotypes* should be applied to certain *Element*. This is one of the few virtual features that is used as a filter inside the qualifier and it does not require a dot or slash notation prior to the alias. The **metaclassName** virtual feature must be followed by an equals symbol and the qualified name of the *Stereotype* between quotation marks.

alias::metaclassName="qualified::name"

It is important to note that this qualified name is basically a path and the name that identifies uniquely each *Stereotype*, and each substring is concatenated with a double colon notation.

Using the metaclassName Virtual Feature

The following steps illustrate what you need to do to use the metaclassName virtual feature:

1. The MapleMBSE ecore is imported and its alias is mse.
2. Two data-sources are used for this example with `metaclassName` to filter *Blocks* and *Requirements*. **Note**: both of those SysML concept are UML Classes but with different Stereotypes.
3. Defining `synctable-schemas`, one for *Blocks* and another for *Requirements*. **Note**: To avoid problems with MapleMBSE it is a good practice to use the same qualifier and *Stereotype* filter in the `data-source` and the first `dimension` of the schema.
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`.

Example

The following example showcases how to use `metaclassName` to create *Classes* applying 2 different *Stereotypes*.

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source blocks = Root/packagedElement[Class | mse::metaclassName="SysML::Blocks::Block"]
6  data-source requirements = Root/packagedElement[Class | mse::metaclassName="SysML::Requirements::Requirement"]
7
8  synctable-schema BlockSchema {
9      record dim [Class | mse::metaclassName="SysML::Blocks::Block"] {
10         key column /name as bName
11     }
12 }
13
14 synctable-schema RequirementSchema {
15     record dim [Class | mse::metaclassName="SysML::Requirements::Requirement"] {
16         key column /name as rName
17     }
18 }
19
20 worksheet-template BlockTemplate (bsc: BlockSchema) {
21     vertical table tab1 at (1, 1) = bsc {
22         key field bName: String
23     }
24 }
25
26 worksheet-template RequirementTemplate(rsc: RequirementSchema) {
27     vertical table tab1 at (1, 1) = rsc {
28         key field rName: String
29     }
30 }
31
32 synctable blockTable = BlockSchema<blocks>
33 synctable requirementTable = RequirementSchema<requirements>
34
35 workbook {
36     worksheet BlockTemplate(blockTable)
37     worksheet RequirementTemplate(requirementTable)
38 }
39
```

Figure 2.1: metaclassName Example

2.2 featureName

Description

As mentioned in the introduction of this section, once you applied a *Stereotype* to any *Element*, you are changing its semantics and extending it. Use `featureName` to access those extended properties stored in *Slots* using their qualified names.

The class diagram in *Figure 2.2 (page 7)* shows the different *EClasses* that need to be queried in order to access those *Slots*. Remember that *Element* is an abstract *EClass* and it should not be used as the qualifier. Basically all elements in a *Model* implement *Element*, thus *EClasses* like *Class* have the structural feature *appliedStereotypeInstance* to query *InstanceSpecification*.

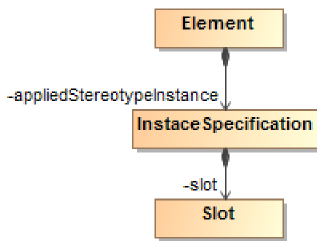


Figure 2.2: The appliedStereotypeInstance Structure

Syntax

Use `featureName` the same way `metaclassName` is used within a qualifier as a filter, meaning that no dot or slash notations are needed before the alias. It expected, following the virtual feature, an equal symbol and a string between quotation marks; this string is the qualified name of the property to access.

```
alias::featureName="qualified::name"
```

This qualified name is similar to the one used to identify the *Stereotype* but it differs slightly at the end with extra information concatenated to identify a single extension. As mentioned before this virtual feature is usable while querying a *Slot* inside a *InstanceSpecification* inside an concrete *Element*, but you must also know that this *Element* must be filtered by `metaclassName` with the qualified name that identifies the *Stereotype*.

Using the featureName Virtual Feature

To access extra Properties added after applying a Stereotype:

1. Import the MapleMBSE ecore.
2. Inside a synctable-schema navigate to a *MultiplicityElement*, in this case, `/ownedAttribute[Property]` within a Class.
3. Within that dimension, define a regular column using `/mse::multiplicityProperty.`
4. Complete the rest of the configuration as usual: worksheet-templates, synctable and workbook.

Example

The following example illustrates how to access extra *Properties* added after applying a *Stereotype*.

1. Import MapleMBSE ecore, for this example use `mse` as the alias.
2. Create a data-source using the `metaclassName` virtual feature mentioned before to filter *Requirements*.
3. Define a synctable-schema for *Requirements*. **Note**: use the same qualifier and *Stereotype* for the first `dimension` as for the `data-source`.
4. To access the `SysML::Requirements::Requirement::Text` *Property* added to a *Class* after applying the Requirement Stereotype you must:
   1. Navigate *appliedStereotypeInstance* to get an *InstanceSpecification*.
   2. Then *slot* to recover all the *Slots* within the *InstanceSpecification*
   3. Use `featureName` with the *Slot* qualifier to filter the *Property* that you want to access

   Note: The qualified name of that *Property* is the name of the qualified *Stereotype* plus 2 colons and the name of the *Property*.
   Stereotype: `SysML::Requirements::Requirement`
   Property: `SysML::Requirements::Requirement::Text`
5. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`.

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source requirements = Root/packagedElement[Class | mse::metaclassName="SysML::Requirements::Requirement"]
6
7
8  synctable-schema RequirementSchema {
9      record dim [Class | mse::metaclassName="SysML::Requirements::Requirement"] {
10         key column /name as rName
11         column /appliedStereotypeInstance[InstanceSpecification]/slot[Slot|mse::featureName=
12            "SysML::Requirements::Requirement::Text"]/value[LiteralString]/value
13               as spec
14     }
15 }
16
17 worksheet-template RequirementTemplate(rsc: RequirementSchema) {
18     vertical table tab1 at (1, 1) = rsc {
19         key field rName: String
20         field spec: String
21     }
22 }
23
24 synctable requirementTable = RequirementSchema<requirements>
25
26 workbook {
27     worksheet RequirementTemplate(requirementTable)
28 }
29
```

Figure 2.3: featureName Example

2.3 stereotypeNames

Description

Use the **stereotypeNames** virtual feature to filter and create *Model Elements* with the specific combination of *Stereotypes*. To use this virtual feature you need a complete and necessary list of *Stereotypes* and their qualified names, and concatenate those qualified names into a single String. Only *Elements* which *Stereotypes* match in number and in qualified name are accepted by this filtering. The order of those *Stereotypes* is not important.

Syntax

This virtual feature is used as a attribute filter inside the qualifier and it does not require a dot or slash. The **stereotypeNames** virtual feature must be followed by an equal symbol and a String with *Stereotypes*. That String must separate the *Stereotypes* qualified names with a comma to work properly.

```
alias::stereotypeNames="one::qualified::name,another::quali-
fied::name"
```

It is important to know that the order of the qualified names are not important. They can be swapped and the same result is to be expected. On the other hand, the String must include the exact number of *Stereotypes* the filter should use. Meaning if you have a model with N *Elements* with *Stereotypes* A and B, filtering using the String "A;B;C" would not show any of those N *Elements* as they do not have the same number of *Stereotypes*.

Using the stereotypeNames Virtual Feature

The following steps illustrate what you need to use the stereotypeNames virtual feature:

1. The MapleMBSE ecore is imported and its alias is mse.
2. A couple data-sources are used for this example with `stereotypeNames` to filter *Packages* and *Classes*.
3. To use this feature to apply *Stereotypes*, you need to define a `synctable-schema`. **Note**: To avoid problems with MapleMBSE it is a good practice to use the same qualifier and *Stereotypes* filter in the `data-source` and the first `dimension` of the schema.
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`.

Example

The following example showcases how to use `stereotypeNames` to create and filter *Elements* with different *Stereotypes*.

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

workbook {
    worksheet ActivityTable(functionalTableSchema)
    worksheet FunctionFMEAMatrix(functionalFMEASchema,functionalTableSchema,fMEATableSchema)
    worksheet FunctionsFMEATable(functionalFMEASchema)
    worksheet FMEARequirementTable(fMEARequirementSchema)
    worksheet DerivedFMEARequirementTable(derivedFMEARequirementSchema)
    worksheet RequirementFMEAMatrix(derivedFMEARequirementSchema, fMEARequirementSchema, fMEATableSchema)
}


data-source Root[Model]
data-source pkg = Root/packagedElement[Model|name = "Model-UAVSystem"]/packagedElement[Package|name = "System Behavior"]
data-source act = pkg/packagedElement[Activity]
data-source fmea = pkg/packagedElement[Package|name = "FMEA"]/packagedElement[Class|mse::stereotypeNames="CustomStereotypes::FMEA"]
data-source fmeaR = pkg/packagedElement[Package|name = "FMEA"]/packagedElement[Package|name = "FMEARequirement"]
                    /packagedElement[Class|mse::stereotypeNames="CustomStereotypes::FMEARequirement,SysML::Requirements::AbstractRequirement"]


synctable-schema FunctionalTableSchema{
    record dim[Activity]{
        key column /name as actName
    }
}

synctable-schema FMEARequirementSchema{
    record dim[Class|mse::stereotypeNames="CustomStereotypes::FMEARequirement"]{
        column /name as ReqName
        key column /appliedStereotypeInstance[InstanceSpecification]
            /slot[Slot|mse::featureName="SysML::Requirements::AbstractRequirement::Id"]/value[LiteralString]/value as ReqID
        column /appliedStereotypeInstance[InstanceSpecification]
            /slot[Slot|mse::featureName="SysML::Requirements::AbstractRequirement::Text"]/value[LiteralString]/value as ReqSpecification
    }
}

....

```

Figure 2.4: stereotypeNames Example

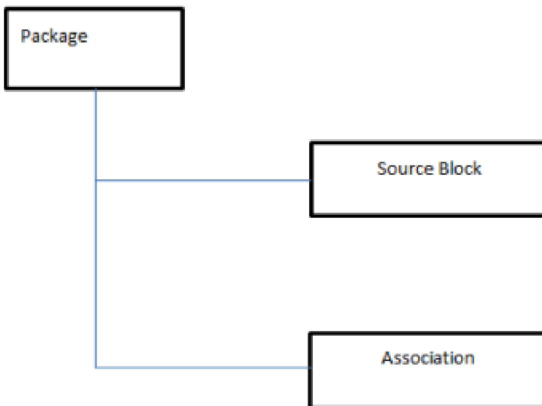
3 Associations

An *Association* between two *Blocks* creates cross references for two UML *Classes* with SysML *Block Stereotypes* (`<<block>>`) to one *Association* using two properties and also makes some cross references, like *Type* and *Association*, within those properties .

3.1 associatedProperty

Description

In MagicDraw, with a couple clicks from one block to another, all of these elements are correctly created. Similarly in MapleMBSE, the `associatedProperty` virtual feature provides the ability to connect two SysML *Blocks*, creating a bidirectional *Association* at the same hierarchicallevel in the diagram as the source *Block*.



When MapleMBSE queries the model, the `associatedProperty` returns the target *Block* (the *Block* that is related to a *Property* through an *Association*).

Syntax

The general syntax for using the `associatedProperty` virtual feature is as follows:

```
.alias::associatedProperty
```

Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.

The `associatedProperty` virtual feature must be used when querying the *Property* of a *Block*.

Using the associatedProperty Virtual Feature

The following example illustrates what you need to do to use AssociatedProperty virtual feature.

1. In line two, the **maplembse ecore** is imported with an alias.
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a class using mse::`associatedProperty`.
4. Complete the `reference-decomposition`.

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source structurePkg = Root/packagedElement[Package]
6  data-source clss = structurePkg/packagedElement[Class]
7
8⊖ synctable-schema ClassTableSchema {
9⊖     dim [Class] {
10         key column /name as ClassName
11     }
12 }
13
14⊖ synctable-schema ClassTreeTableSchema(blocks: ClassTableSchema) {
15⊖     record dim [Class] {
16         key column /name as className1
17     }
18⊖     dim /ownedAttribute[Property].mse::associatedProperty[Class] @ cls {
19⊖         reference-decomposition cls = cts {
20         foreign-key column ClassName as referredClassName
21         }
22     }
23 }
24
25 synctable classTableSchema = ClassTableSchema<clss>
26 synctable classTreeTableSchema = ClassTreeTableSchema<clss>(classTableSchema)
27
28⊖ worksheet-template ClassTable(cts: ClassTableSchema) {
29⊖     vertical table tab1 at (6, 2) = cts {
30         key field ClassName : String
31         key field Name4 : String
32     }
33 }
34
35⊖ worksheet-template ClassTreeTable(ctt: ClassTreeTableSchema) {
36⊖     vertical table tab1 at (6, 2) = ctt {
37         key field ClassName1 : String
38         key field referredClassName : String
39     }
40 }
41
42⊖ workbook{
43     worksheet ClassTable(classTableSchema)
44     worksheet ClassTreeTable(classTreeTableSchema)
45 }
```

Figure 3.1: associatedProperty Example

3.2 directedAssociatedProperty

Description

To create *Associations* with navigability in one direction MapleMBSE uses *directedAssociatedProperty*, using this virtual feature links two *Classes* and adds a *Property* to the source *Block* and other *Property* to an *Association*.
Based on the *aggregation* value we can use this virtual feature to create *Association*, *Aggregation* and *Composition* with direction.

Syntax

The general syntax for using the `directedAssociatedProperty` virtual feature is as follows:

```
.alias::directedAssociatedProperty
```

Where `alias` is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The `directedAssociatedProperty` virtual feature must be used when querying the *Property* of a *Block*.

Using the directAssociatedProperty Virtual Feature

The following example illustrates what you need to do to use `directedAssociatedProperty`.

1. In line two, the **maplembse ecore** is imported with an alias.
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a class using `mse::directedAssociatedProperty`.
4. Complete the `reference-decomposition`.

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source structurePkg = Root/packagedElement[Package]
6  data-source clss = structurePkg/packagedElement[Class]
7
8  synctable-schema ClassTableSchema {
9      dim [Class] {
10         key column /name as ClassName
11     }
12 }
13
14 synctable-schema ClassTreeTableSchema(blocks: ClassTableSchema) {
15     record dim [Class] {
16         key column /name as className1
17     }
18     dim /ownedAttribute[Property].mse::directedAssociatedProperty[Class] @ cls {
19         reference-decomposition cls = cts {
20         foreign-key column ClassName as referredClassName
21         }
22     }
23 }
24
25 synctable classTableSchema = ClassTableSchema<clss>
26 synctable classTreeTableSchema = ClassTreeTableSchema<clss>(classTableSchema)
27
28 worksheet-template ClassTable(cts: ClassTableSchema) {
29     vertical table tab1 at (6, 2) = cts {
30         key field ClassName : String
31         key field Name4 : String
32     }
33 }
34
35 worksheet-template ClassTreeTable(ctt: ClassTreeTableSchema) {
36     vertical table tab1 at (6, 2) = ctt {
37         key field ClassName1 : String
38         key field referredClassName : String
39     }
40 }
41
42 workbook{
43     worksheet ClassTable(classTableSchema)
44     worksheet ClassTreeTable(classTreeTableSchema)
```

Figure 3.2: directAssociatedProperty Example

3.3 otherAssociatedEnd

Description

otherAssociationEnd is used in the case when two classifiers have to be linked and the information about the properties of these classifiers are owned by the association and not the classifiers themselves, such as in the case of UseCase diagram where association exists between an actor and usecase and these two classifiers do not own any property that defines the other classifier.

Syntax

The general syntax for using the `otherAssociationEnd` virtual feature is as follows:

```
.alias::otherAssociationEnd
```

Where `alias` is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The `otherAssociationEnd` virtual feature must always be used when querying a Class
.

Using the otherAssociatedEnd Virtual Feature

The following example illustrates what you need to do to use `otherAssociationEnd`.

1. In line two, the **maplembse ecore** is imported with an alias.
2. Use when a `Class` as the queried dimension.
3. Make a reference-query to a class using `mse::otherAssociationEnd`, unlike other virtual features in this section otherAssociationEnd should not be used when a property is querried.
4. Complete the `reference-decomposition`.

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source useCasePkg = Root/packagedElement[Package]
6  data-source actors = useCasePkg/packagedElement[Actor]
7  data-source useCases = useCasePkg/packagedElement[UseCase]
8
9  synctable-schema ActorsTable {
10     record dim [Actor] {
11         key column /name as Actor
12     }
13 }
14
15 synctable-schema UseCasesTable(ac:ActorsTable) {
16     record dim [UseCase] {
17         key column /name as Name
18         reference-query .mse::otherAssociationEnd[Actor] @ actor
19         reference-decomposition actor = ac {
20         foreign-key column Actor as Actor
21         }
22     }
23 }
24
25
26 synctable actorsTable = ActorsTable<actors>
27 synctable useCasesTable = UseCasesTable<useCases>(actorsTable)
28
29 worksheet-template Actors(ac: ActorsTable) {
30     vertical table tab1 at (5, 3) = ac {
31         key field Actor : String
32     }
33 }
34
35 worksheet-template UseCases(auct: AssociatedUseCasesTable) {
36     vertical table tab1 at (5, 3) = auct {
37         key field Name : String
38         key field Actor : String
39     }
40 }
41
42 workbook {
43     worksheet Actors(actorsTable)
44     worksheet UseCases(useCasesTable)
45 }
```

Figure 3.3: otherAssociatedEnd Example

3.4 nestedDirectedComposition

Description

MapleMBSE is powerful enough to change any SysML feature, in particular a *nestedClassifier*. The effort to change the model in a desired way is always related to creating the right schemas and data sources to offer intuitive views. Unfortunately, creating a nested block and a directed composition to it is not an easy task without this virtual feature. The creation of composition association to non-existing nested block should be possible just by mentioning the name of the target block in the right dimension.

Syntax

The general syntax for using the *nestedDirectedComposition* virtual feature is as follows:

dim */alias::nestedDirectedComposition*[Association]

Where `alias` is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The *nestedDirectedComposition* virtual feature must be used when querying the *Block* and *Association* in a dimension. It always needs to be used in conjunction with another virtual feature to set up the target *Block*: *targetBlockName.*

The general syntax for using the *targetBlockName* virtual feature is as follows:

key column /alias::targetBlockName

This *targetBlockName* virtual feature should be the only key column for a *nestedDirected-Composition* dimension.

Using the nestedDirectedComposition virtual feature

The following example illustrates what you need to do to use *nestedDirectedComposition*

1. In line two, the **maplembse ecore** is imported with an alias.
2. Use a *Class* or an *Association* qualifier in the querried dimension, as shown in line 8, 12 and 16.
3. Create a *targetBlockName* key column for each *nestedDirectedComposition dimension*

Example

```
data-source Root[Model]

data-source blocks = Root
/packagedElement[Package|name="Structure"]
/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]

synctable-schema Schema {
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as bName
    }

    record dim /mse::nestedDirectedComposition[Association] {
        key column /mse::targetBlockName as nbName
    }

    record dim /mse::nestedDirectedComposition[Association] {
        key column /mse::targetBlockName as nbName2
    }

}

worksheet-template Template(sc: Schema){
    vertical table tab1 at (4,2) = sc {
        key field bName
        key field nbName
        key field nbName2
        sort-keys bName, nbName, nbName2
    }
}

synctable blockTable = BlockSchema<rBlocks>
synctable dataTable = Schema<blocks>
synctable associatedDataTable = AssociatedPropertySchema<blocks>(blockTable)

workbook {
    worksheet Template(dataTable) {label="Nested Classifier"}
}
```

4 Blocks

4.1 recursivePartProperties

Description

The recursivePartProperties virtual feature helps finds all the related blocks and sub-blocks and part properties, recursively.

Displaying a block and related sub-blocks in the same syncview is difficult if they are in different packages, and there is a chance that relevant blocks are missing in the syncview. The recursivePartProperties virtual feature helps find all the related blocks, sub-blocks and part properties, recursively. This makes it easier to create a corresponding configuration file.

The recursivePartProperties virtual feature works in a similar fashion to recursiveInstance-WithSlots, and a common use case is to use both of these in conjunction for instance matrices.

Syntax

The configuration file syntax for using recursivePartProperties is illustrated below.

```
datasource blocks = Root/packageElement[Pack-
age|name=”Test”]/packageElement[Class|name=”B1”]/mse::recurs-
ivePartProperties[Class];
synctable-schema TestSchema {
record dim[Class]{
key column /name as mainBlock
}
Dim /mse::recursivePartProperties[Class]{
Key column /name as subBlocks
}
}
```

Using the recursivePartProperties Virtual Feature

The following example illustrates one way to use the recursiveIPartProperties virtual feature:

1. Import the MapleMBSE ecore with an alias.
2. Create a datasource that has the context/main block for which you want to find the properties(for example, `../packageElement[Class|name="B1"]`).
3. Use `recursivePartProperties[Class]` to return all the classes linked to the context/main block and sub-blocks.

4. Create a `sync-schema` and `synctable` and after that use that datasource in the synctable.

Example

```
datasource blocks = Root/packageElement[Pack-
age|name="Test"]/packageElement[Class|name="B1"]/mse::recurs-
ivePartProperties[Class];
```

4.2 propertyDefaultValue

Description

Previously, to view or edit the default value of the value property or property without any Stereotype, the author/editor of the configuration file editor had to write the line column /value[LiteralReal]/value. If the property contains a value other than a real value then MapleMBSE will not display this value in the cell. If the configuration editor were to write the MSE file in such a way as to view the value of every type of value property, It will complicate the MSE file and still, the end user will not able to view the values in one single column. The propertyDefaultValue virtual feature fixes this problem and helps the configuration file editor and the end user to view and edit the value in one column.

Syntax

/mse::propertyDefaultValue

Using the propertyDefaultValue Virtual Feature

1.Import the MapleMBSE ecore with an alias.
2. Create a datasource that has the blocks/classes for which you want to find the properties(for example, `../packageElement[Class]`).
3. Use `ownedAttribute[Property]` to get properties from the block/Class
4. Use the `propertyDefaultValue` to get the value from the property

5. Create a `sync-schema` and `synctable` and then use that datasource in the synctable.

Example

```
synctable-schema Schema{
 record dim [Class|mse::metaclassName="SysML::Blocks::Block] {
 key column /name as BlockName
 }
 record dim /ownedAttribute[Property mse::metaclassName |="MD Customization
 for SysML::additional_stereotypes::ValueProperty"] {
```

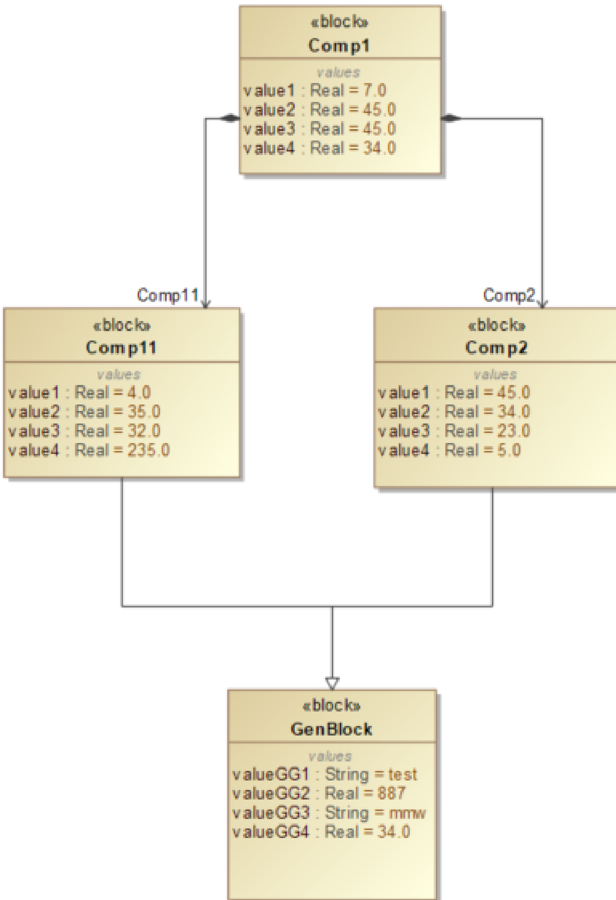
```
 key column /name as pName
 column /mse::propertyDefaultValue as pvalue
 }
}
```

4.3 getAllProperties

Description

The `getAllProperties` virtual feature retrieves properties under a block. These properties can be direct or indirect. If the block has generalization it can go as much as possible in the upper direct to get the properties but for the composition it can only go one step down.

The images below show the model, MSE file, and the view in MapleMBSE. In the model block, Comp11, and Comp2 are generalized to GenBlock using this feature MapleMBSE can query and modify the properties that are also inherited from the GenBlock.



Note: This feature can be used with the `recursivePartProperties` to get the flat view or can be used alone for the hierarchical view.

Blocks	ValueProperties	DefaultValue
Comp1	value1	7
Comp1	value2	45
Comp1	value3	45
Comp1	value4	34
Comp11	value1	4
Comp11	value2	35
Comp11	value3	32
Comp11	value4	235
Comp11	valueGG1	test
Comp11	valueGG2	887
Comp11	valueGG3	mmw
Comp11	valueGG4	34
Comp2	value1	45
Comp2	value2	34
Comp2	value3	23
Comp2	value4	5
Comp2	valueGG1	test
Comp2	valueGG2	887
Comp2	valueGG3	mmw
Comp2	valueGG4	34

Syntax

dim /mse::getAllProperties

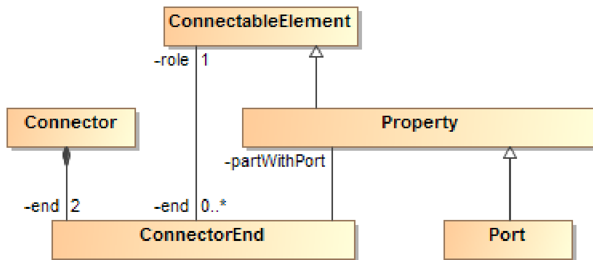
Using the getAllProperties virtual feature

1.Import the MapleMBSE ecore with an alias.
2. Create a datasource that has the blocks/classes for which you want to find the properties(for example, ../packageElement[Class]).
3. Use mse::getAllProperties[Property] to get properties from the block/Class (directly owned properties or inherit properties)
4. Create a sync-schema and synctable and after that use that datasource in the synctable.

Example

```
synctable-schema BlocksValueTable {
    dim[Class|mse::metaclassName="SysML::Blocks::Block"]{
        key column /name as BlockName
    }
    dim /mse::getAllProperties[Property
        |mse::metaclassName="MD Customization for SysML::additional_stereotypes::ValueProperty",
            aggregation="composite"] {
        key column /name as ValueProp
        column /mse::propertyDefaultValue as defaultValue
    }
}
```

5 Connectors



A *Connector* is used to link *ConnectableElements* (for example, *Ports* or *Properties)* of a *Class* through a *ConnectorEnd.* A *Connector* has two *ConnectorEnds*.

Based on the connection between *Properties* of a *Class* the connection can be of two types: *Delegation* (connecting *Ports* or *Properties* from the system to *Ports* or *Properties* inside a *Class*) or *Assembly* (connecting *Ports* or *Properties* within a *Class*).

5.1 connectedPropertyOrPort

Description

To achieve this connection MapleMBSE uses `connectedPropertyOrPort` virtual feature.
The `connectorPropertyOrPort` virtual feature connects *Ports* or *Properties* of a *Class.* It automatically detects the kind of relation required between the *Properties* being connected and creates the appropriate connection.

When MapleMBSE queries the model, the `connectedPropertyOrProt` return the list of target properties.

Syntax

The general syntax for using the `connectedPropertyOrPort` virtual feature is as follows:

```
.alias::connectedPropertyOrPort
```

Where the `alias` is alias you assigned to MapleMBSE ecore.

When the connection is created through `connectedPropertyOrPort,` the owner of the connected *Property* is determined automatically by MapleMBSE, regardless of whether this is a *Delegation* or *Assembly* type connection.

Using the connectedPropertyOrPort virtual feature

In general, to use the `connectedPropertyOrPort` virtual feature:

1. First, import the MapleMBSE ecore with `alias`
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a property using `mse::connectedPropertyOrPort`.
4. Complete the `reference-decomposition`.

Example

A specific example of how to use the `ConnectedPropertyOrPort` virtual feature is shown below.

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

synctable-schema BlocksTable {
        record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
                key column /name as BlockName
        }
        dim /ownedAttribute[Property] {
                key column /name as PropertyName
        }
}
synctable-schema ConnectedPropertyOrPortTable(bT: BlocksTable) {
        record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
                key column /name as className
        }
        record dim /ownedAttribute[Property] {
                key column /name as ParentPort
        }
        record dim .mse::connectedPropertyOrPort @ cls {
                reference-decomposition cls = bT {
                foreign-key column BlockName as referredClassName
                foreign-key column PortName as referredPortName
                }
        }
}
```

Figure 5.1: connectedPropertyOrPort Example

5.2 otherConnectorEnd

Description

To achieve this connection MapleMBSE also use `otherConnectorEnd` virtual feature. This virtual feature can connect between ports or properties of a class, `otherConnectorEnd` automatically create the relation required between the properties being connected and creates appropriate connection.

When MapleMBSE queries the model, the `otherConnectorEnd` return the list of connectorEnds which is associated with the property.

Syntax

The general syntax for using the `otherConnectorEnd` virtual feature is as follows:

```
.alias::otherConnectorEnd
```

Where the `alias` is the alias you assigned to the MapleMBSE ecore.

When the connection is created using `otherConnectorEnd`, the owner of the connected *Property* is determined automatically by MapleMBSE, regardless of whether this is a *Delegation* or *Assembly* type connection.

Using the otherConnectorEnd Virtual Feature

How to use the `otherConnectorEnd` virtual feature is shown in the example below:

1. First, import the MapleMBSE ecore with an appropriate alias
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a property using `mse::`otherConnectorEnd.
4. Complete the `reference-decomposition`.

Example

A specific example of how to use the `otherConnectorEnd` virtual feature is shown below.

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3⊖ synctable-schema BlocksTable {
4⊖         record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
5                  key column /name as blockName
6          }
7⊖         dim /ownedAttribute[Property] {
8                  key column /name as propertyName
9          }
10 }
11⊖ synctable-schema OtherConenctorEndTable(bt:BlocksTable){
12⊖         record dim[Class|mse::metaclassName="SysML::Blocks::Block"]{
13                 key column /name as ownerBlockName
14         }
15
16⊖         record dim /ownedAttribute[Property]{
17                 key column /name as pname
18         }
19⊖         record dim .mse::otherConnectorEnd[ConnectorEnd] {
20                 key reference-query .role @  cls
21⊖                 reference-decomposition cls = bt {
22                         foreign-key column BlockName as refRoleBlock
23                         foreign-key column PropertyName as refportName
24                 }
25                 reference-query .partWithPort @ pwp
26⊖                 reference-decomposition pwp = bt {
27                         foreign-key column BlockName as refPropertyBlock
28                         foreign-key column PropertyName as refPropertName
29                 }
30         }
31 }
```

Figure 5.2: otherConnectorEnd Example

6 Dependencies

A *Dependency* is used between two model elements to represent a relationship where a change in one element (the supplier element) results in a change to the other element (client element).

A *Dependency* relation can be created between any *namedElement*. Different kinds of *Dependencies* can be created between the model elements such as *Refine*, *Realization*, *Trace*,*Abstraction* etc.

6.1 clientDependencies

Description

The `clientDependencies` virtual feature creates a relation between the client being the dependent and supplier who provides further definition for the dependent.

Syntax

The general syntax for using the `clientDependencies` virtual feature is as follows:

```
/mse::clientDependencies
```

This virtual feature is used while querying a Class that has to be assigned as client to the dependency that is being created and is used in a following dimension the class that is being queried.

Where `alias` is the alias you assigned to the MapleMBSE ecore.

Using the clientDependencies Virtual Feature

In general, the following steps outline how to use `clientDependencies`:

1. It should be used when a named element is queried.
2. Information about the type of relationship is specified as `[Dependency]`, `[Abstraction]` etc.
3. When querying the model element with `mse::clientDependencies`, the reference decomposition should be to a supplier element.

Example

The example below is an illustration of how to use the `clientDependencies` virtual feature.

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source package = Root/packagedElement[Package|name="Package"]
data-source act = package/packagedElement[Activity]
data-source cls = package/packagedElement[Class]

synctable-schema ActivityTableSchema {
    record dim [Activity] {
        key column /name as ActName
    }
}

synctable-schema ClassAbstractionTableSchema(acts:ActivityTable) {
    record dim [Class] {
        key column /name as ActName1
    }
    record dim /mse::clientDependencies[Dependency] {
        key reference-query .supplier @ refDecomp
            reference-decomposition refDecomp = reqs {
            foreign-key column ActName as AbsName
            }
    }
}

synctable activityTableSchema = ActivityTableSchema<act>
synctable classAbstractionTableSchema = ClassAbstractionTableSchema<cls>(ActivityTable)

worksheet-template ActivityTable(cts:ActivityTableSchema){
    vertical table tab1 at (4,5) = cts{
        key field ActName : String
    }
}

worksheet-template ClassAbstractionTable(cds:ClassAbstractionTableSchema){
    vertical table tab1 at (4,5) = cds{
        key field ActName1 : String
        key field AbsName : String
    }
}
workbook{
    worksheet ActivitiesTable(ActivityTable)
    worksheet ClassAbstractionTable(classAbstractionTableSchema)
}

```

Figure 6.1: clientDependencies Example

6.2 supplierDependencies

Description

Similar to `clientDependencies`, `supplierDependencies` is used to create a relation between two named elements. The only difference between the two virtual features is `supplierDependencies` is used when the relationship has to be made from supplier to client instead of client to supplier, as in the case of `clientDependencies`.

Syntax

The general syntax for using the `supplierDependencies` virtual feature is as follows:

```
/mse::supplierDependencies
```

This virtual feature is used while querying a Class that has to be assigned as supplier to the dependency that is being created and is used in a dimension following the class that is being queried.

Where `alias` is the alias you assigned to the MapleMBSE ecore.

Using the supplierDependencies Virtual Feature

The following example illustrates what you need to do to use supplierDependencies

1. It should be used when a named element is being queried.
2. Information about the type of relationship is specified as [Dependency], [Abstraction] etc.
3. When querying the model element with mse::supplierDependencies the reference decomposition should be to a client element.

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source package = Root/packagedElement[Package|name="Package"]
6  data-source cls = package/packagedElement[Class|mse::metaclassName="SysML::Requirements::Requirement"]
7
8  synctable-schema RequirementsTableSchema {
9      record dim [Class|mse::metaclassName="SysML::Requirements::Requirement"] {
10         key column /name as ReqName
11     }
12 }
13
14 synctable-schema RequirementsDerivesTableSchema(reqs:RequirementsTable) {
15     record dim [Class|mse::metaclassName="SysML::Requirements::Requirement"]    {
16         key column /name as ReqName1
17     }
18     record dim /mse::supplierDependencies[Abstraction|mse::metaclassName="SysML::Requirements::DeriveReqt"] {
19         key reference-query .client @ reqDecomp
20         reference-decomposition reqDecomp = reqs {
21             foreign-key column ReqName as DeriveName
22         }
23     }
24 }
25
26 synctable requirementsTableSchema = RequirementsTableSchema<cls>
27 synctable requirementsDerivesTableSchema = RequirementsDerivesTableSchema<cls>(requirementsTable)
28
29 worksheet-template ReqClassTable(cts:RequirementsTableSchema){
30     vertical table tab1 at (4,5) = cts{
31         key field Name : String
32     }
33 }
34
35 worksheet-template ReqClassDependency(cds:RequirementsDerivesTableSchema){
36     vertical table tab1 at (4,5) = cds{
37         key field Name1 : String
38         key field DeriveName : String
39     }
40 }
41
42 workbook{
43     worksheet ReqClassTable(requirementsTableSchema)
44     worksheet ReqClassDependency(requirementsDerivesTableSchema)
45 }
```

Figure 6.2: supplierDependencies Example

6.3 featureImpact

Description

The featureImpact virtual feature is used in context with the MBPLE profile. Using this profile a user can model a 150% model and use the feature impact (Dependency) relation to link the Existence with the Feature Model. When a feature impact relationship is created, the tag values should also be set based on the client and supplier. Adding the relation with real features requires the user to add additional details that complicates the user actions.

Using featureImpact, just by specifying the source and target, the tag values are updated automatically. The featureImpact virtual feature works similar to the clientDependencies virtual feature. The constraint is always set as client and the supplier is the feature.

Syntax

/mse::featureImpact

Using the featureImpact Virtual Feature

1. Import the MapleMBSE ecore with an alias.
2. Create a datasource for the existence (constraints with stereotypes) to which the features has to be linked.
3. Use /mse::featureImpact and this will set the constraint as client, and to set the supplier use the supplier feature.

Example

```
1  synctable-schema featureImpactVFSchema (es : featureSchema, sps : valpropertySchema) {
2       record dim [Class|mse::stereotypeNames="SysML::Blocks::Block"]{
3            key column /name as blockName
4       }
5
6       record dim /ownedRule[Constraint|mse::stereotypeNames=
   "MBPLE Profile::ExistenceVariationPoint,UML Standard Profile::MagicDraw Profile::InvisibleStereotype"]{
7            key column /name as existenceName
8            reference-query .constrainedElement[Property|mse::stereotypeNames=
   "MD Customization for SysML::additional_stereotypes::PartProperty"] @ feaRef
9            reference-decomposition feaRef = sps {
10                foreign-key column propertyName as propertyName
11           }
12      }
13      /**
14      * create dependency, constraint  and tag values
15      */
16      record dim /mse::featureImpact[Dependency|mse::stereotypeNames="MBPLE Profile::FeatureImpact"] {
17           key reference-query .supplier[Property|mse::stereotypeNames="MBPLE Profile::Feature"] @ feaRef
18           reference-decomposition feaRef = es {
19                foreign-key column featureName as featureName
20           }
21           column /appliedStereotypeInstance[InstanceSpecification]/slot[Slot|mse::featureName=
   "MBPLE Profile::FeatureImpact::testFor"]/value[InstanceValue].instance[EnumerationLiteral]/name as fName
22      }
23 }
```

6.4 Multiple Dependencies Class

Introduction

Earlier on in this guide, when the Matrix concept was introduced, the focus was on a Matrix where you can view only one type of relation in a Matrix.

However, through the use of a virtual mse::MultipleDependencies Eclass and the multipleDependencies virtual feature, multiple types of relations can be displayed at once in a Matrix, where the user can pass a parameter to the virtual feature to control the type of relations displayed in the matrix.

Creating a Multiple Dependencies Class in an MSE file

The syntax for creating the multiple dependencies is:

data-source deps = <previous-ds>/mse::multipleDependencies(StringFilters)[mse::MultipleDependencies]

where where <previous-ds> represents the needed navigation to all the dependencies expected to be displayed, e.g. Root/packagedElement[Package|name="Dependencies"].

Next, mse::multipleDependencies(StringFilters) represents the virtual feature, multipleDependencies. In this case, StringFilters represents the parameters passed to multipleDependencies used to select the type of relations displayed in the matrix.

An example:

This example is taken from the TWCSysML-RelationMatrix.MSE model file from the **Application\TWCSysML\2021x** directory of your MapleMBSE installation.

data-source multiDep = sysStructurePkg/mse::multipleDependencies("SysML::Requirements::Satisfy","SysML::Requirements::Verify")[mse::MultipleDependencies]

The data-source, multiDep is defined as a multi-relational datasource. Here sysStructurePkg represents the Package data source where the virtual feature, multipleDependencies will find the Dependencies. Here, the string filters provided are two dependencies, "Satisfy" and "Verify". The [mse::MultiDependencies] part of this code represents the output from the multiDependences virtual feature, which is a MultipleDependencies object.

Note: If you do not specify a stereotype as a parameter to multiDependencies, then all stereotypes attached to the dependencies will be used and they will appear in the model worksheet as a list of stereotypes.

7 Enumeration

Enumeration is a special *DataType* that can be compared to a list of possible values, the way that "colors" can be an enumeration and possible values can be: red, blue, green, etc. These *Enumerations* are composed of *EnumerationLiterals* which are the different values and the actual *Elements* to be referenced. MapleMBSE supports a couple virtual features that need to be used in conjunction to access and reference any *Enumeration* and its *EnumerationLiterals* independently of where in the TWCloud project those values are stored (for example, under *Model* or customized profile)

7.1 EnumerationName

Description

MapleMBSE, to simplify *Enumeration* identification*,* supports an `enumerationName` virtual feature that allows simpler access to a specific Enumeration while creating an MSE configuration. Note that MapleMBSE, while using this virtual feature, will by default instantiate the accessed *Element* to the first *EnumerationLiteral* of the *Enumeration*. Nonetheless, *enumerationLabel* can be used to change to another *EnumerationLiteral.* See the next section for further details.

Syntax

The general syntax for using the `enumrationName` virtual feature is as follows:
`alias::enumrationName="qualified::name"`
Where `alias` is the alias you assigned to the MapleMBSE ecore and `qualified::name is` the `qualifiedName` of the *Enumeration*. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.
The `enumrationName` virtual feature must be used while querying an *Element* with a *Stereotype* that supports some *Property* with an *Enumeration* `type.` For more information on how to access a Slot, see the sections in the guide on the *metaclassName* and *featureName* virtualFeatures. Once you get the specific *Slot,* retrieve its *value* and within its *Qualifier* filter use `enumrationName`.

Using the enumerationName Virtual Feature

The following example illustrates what you need to do to use the `enumerationName` virtual feature:

1. Import the maplembse ecore with an alias.
2. Create a schema that takes an Element with a Stereotype and navigate down to its InstanceValue for a Property with an Enumeration type. See lines 15 to 18 in the example code in the next section for an illustration.

3. Make sure you are using the right combination of qualified names for Stereotypes, Slot Properties and Enumeration.
4. Complete the /value[InstanceValue] navigation with an enumerationLabel (see next section for further details).

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  workbook {
5      worksheet EnumrationTemplate(enums)
6  }
7
8  data-source Root[Model]
9  data-source reqs = Root/packagedElement[Package|name="Enum"]
10         /packagedElement[Class|mse::metaclassName="SysML::Non-Normative Extensions::Requirement::extendedRequirement"]
11
12 synctable-schema EnumSchema {
13     dim [Class|mse::metaclassName="SysML::Non-Normative Extensions::Requirement::extendedRequirement"] {
14         key column /name as rName
15         column /appliedStereotypeInstance[InstanceSpecification]
16             /slot[Slot|mse::featureName="SysML::Non-Normative Extensions::Requirement::extendedRequirement::verifyMethod"]
17             /value[InstanceValue|mse::enumerationName="SysML::Non-Normative Extensions::Requirement::VerificationMethodKind"]
18             /mse::enumerationLabel as verificationMethod
19     }
20 }
21
22 synctable enums = EnumSchema<reqs>
23
24 worksheet-template EnumrationTemplate (es: EnumSchema) {
25     vertical table tab1 at (1, 1) = es {
26         key field rName
27         field verificationMethod
28     }
29 }
```

7.2 EnumerationLabel

Description

As shown in the previous sections on EnumerationName, MapleMBSE allows you to make a reference to Enumeration using a qualifiedName. However, without the right mechanism to translate from String to EnumerationLiterals and vice versa, the end user will be forced to deal with strange Object references or unusable Excel cells. This is exactly the problem enumerationLabel was designed to solve. Using this virtual feature allows the end user to see the String name of the EnumerationLiteral without forcing any reference-decomposition and it allows also the end user to change the reference from the Slot Property using the String name of the desired EnumerationLiteral

Syntax

The general syntax for using the `enumerationLabel` virtual feature is as follows:
`/alias::enumerationLabel`
Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.
The `enumerationLabel` virtual feature must be used while querying an *InstanceValue* with a *Stereotype* that supports some *Property* with a *Enumeration* `type` and which was

filtered with `enumerationName.` For more information how to access this kind of *InstanceValue*, see the previous section.

8 Util

This section contains all other virtual features that do not create elements but offer a better alternative to access and map model information.

8.1 multiplicityProperty

Description

The UML specification contains several *MultiplicityElements* like *Properties* that have *upper* and *lower* features to describe their multiplicity. Use the **`multiplicityProperty`** virtual feature to make a configuration that translates a string into those *upper* and *lower* values and the other way around.
This virtual feature recognizes the UML commonly used notation for multiplicity (e.g. 0..*). Supporting this notation makes MapleMBSE much easier to use without adding complexity and thus the final user has less to input into Excel.

Syntax

The general syntax for using the `multiplicityProperty` virtual feature is as follows:

```
/alias::multiplicityProperty
```

Where the `alias` is the alias you assigned to the MapleMBSE ecore.

This virtual feature can only be used while querying a concrete *EClass* implementing a *MultiplicityElement* like a *Property* or a *Pin*. A slash notation is needed prior to the alias, the 2 colons, and **`multiplicityProperty`**.

As mention previously `multiplicityProperty` uses a string to represent the multiplicity, meaning that this particular virtual feature cannot being used as a dimension with a qualifier. It is intended to be used only at a column declaration.

Using the multiplicityProperty Virtual Feature

The following example shows you how to map the multiplicity of a concrete *MultiplicityElement* like *Property* and a string.

1. Import the MapleMBSE ecore, as usual the alias used is `mse`
2. Inside a synctable-schema navigate to a *MultiplicityElement*, in this case */ownedAttribute[Property]* within a *Class*
3. Within that dimension, define a regular column using `/mse::multiplicityProperty`

4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source classes = Root/packagedElement[Class]
6
7  synctable-schema Schema {
8      record dim [Class] {
9          key column /name as cName
10     }
11
12     record dim /ownedAttribute[Property] {
13         key column /name as pName
14         column /mse::multiplicityProperty as multiplicity
15     }
16 }
17
18 worksheet-template Template(sch: Schema) {
19     vertical table tab1 at (2, 2) = sch {
20         key field cName : String
21         key field pName : String
22         field multiplicity : String
23         sort-keys cName, pName
24     }
25 }
26
27 synctable tableProperty = Schema<classes>
28
29 workbook {
30     worksheet Template(tableProperty)
31 }
```

Figure 8.1: multiplicityProperty Example

9 Activity Diagrams

An Activity Diagram is a diagram with a direct connection, ActivityEdge, that connects a node, ActivityNode, to another ActivityNode. An Activity Diagram is useful to abstract behavioral information within a system. In order to improve MSE configurations, MapleMBSE supports control and object flow, the 2 kind of ActivityEdges, with two distinct virtual features.

9.1 ActivityControlFlow

Description

A *ControlFlow* is an *ActivityEdge* that is used to control the execution of *ActivityNodes* within an Activity.

In MapleMBSE, the virtual feature ActivityNode is used as a reference to create ControlFlows. Note that in MapleMBSE, abstract classes such as ActivityNode cannot be instantiated. Thus, you must instantiate concrete classes such as CallActionBehavior, ActivityParameterNode, or InitialNode. *S*ee the example section for further details.

Syntax

The general syntax for using the `activityControlFlow` virtual feature is as follows:
`.alias::activityControlFlow`
Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.

The `activityControlFlow` virtual feature must be used when querying the *ActivityNode* of Activity.

Using the ActivityControlFlow Virtual Feature

The following example illustrates what you need to do to use `activityControlFlow` virtual feature:

1. Import the **maplembse ecore** with an alias.
2. Create a schema that navigates until an *ActivityNode* or an element that has an ActivityNode as its first dimension
3. Make a dimension `reference-query` to another *ActivityNode* using `.mse::activityControlFlow.`
4. Complete the `reference-decomposition`.

This example has extra schema, `CallBehaviorActionSchema` used to create concrete *ActivityNodes*. The other schemas in this example will fail to instantiate *Element* because

ActivityNode is an abstract class.

Note: Some data sources specific to a fictional project were created to simplify the `reference-decomposition`. In a real life scenario you might need to identify the *Package*, the *Activity* and the *ActivityNode* that you want to connect to.

Example

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4  data-source Root[Model]
5  data-source pkg = Root/packagedElement[Package|name = "controlflow"]
6  data-source activities = pkg/packagedElement[Activity|name="activity"]
7  data-source nodes = activities/node[ActivityNode]
8  data-source cba = activities/node[CallBehaviorAction]
9
10 synctable-schema NodeSchema {
11     dim [ActivityNode] {
12         key column /name as nName
13     }
14 }
15
16 synctable-schema CallBehaviorActionSchema {
17     dim [CallBehaviorAction] {
18         key column /name as nName
19     }
20 }
21
22 worksheet-template CallBehaviorActionTemplate (cbasc: CallBehaviorActionSchema) {
23     vertical table tab1 at (2, 1) = cbasc {
24         key field nName
25     }
26 }
27
28 synctable-schema Schema(nsc: NodeSchema) {
29     dim [ActivityNode] {
30         key column /name as nName
31     }
32
33     dim .mse::activityControlFlow[ActivityNode] @ tgtNode {
34         reference-decomposition tgtNode = nsc {
35             foreign-key column nName as tgtNode
36         }
37     }
38 }
```

Figure 9.1: ActivityControlFlow Example

9.2 ActivityObjectFlow

Description

An *ObjectFlow* is an *ActivityEdge* that represents the flow of object data between *ActivityNodes* within an *Activity*. Sometimes, the *ObjectFlow* directly connects two *ActivityNodes*. However, due to UML specifications, some *ActivityNodes* cannot be connected directly using an *ObjectFlow*. In these cases *Pins* are required. *Pins* are objects that accept and provide values to actions. These values represent an input to an action or output from an action.

If an *ActivityNode* that requires *Pins*, such as *CallBehaviorAction*, also has a *Behavior* that further describes it's functionality, then both the *ActivityNode* and *Behavior* need to have their *Pins* (specifically *ActivityParameterNode* and *Parameters*) synchronized, both in quantity and direction.

Syntax

The general syntax for using the `activityObjectFlow` virtual feature is as follows:
`.alias:: activityObjectFlow`
Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Introduction (page 1)*.
The `activityObjectFlow` virtual feature must be used when querying the *ActivityNode* of *Activity*.

Using the ActivityObjectFlow Virtual Feature

The following example illustrates what you need to do to use `activityObjectFlow` virtual feature:

1. Import the **maplembse ecore** with an alias.
2. Create a schema that navigates until an *ActivityNode* or an element which has an ActivityNode as its dimension.
3. Make a dimension `reference-query` to another *ActivityNode* using `.mse:: activityObjectFlow`.
4. Complete the reference-decomposition.

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name = "objectflow"]
data-source activities = pkg/packagedElement[Activity|name="activity"]
data-source nodes = activities/node[ActivityNode]

synctable-schema NodeSchema {
    dim [ActivityNode] {
        key column /name as nName
    }
}

synctable-schema Schema(nsc: NodeSchema) {
    dim [ActivityNode] {
        key column /name as nName
    }

    dim .mse::activityObjectFlow[ActivityNode] @ tgtNode {
        reference-decomposition tgtNode = nsc {
            foreign-key column nName as tgtNode
        }
    }
}

worksheet-template Template (sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
        key field nName
        key field tgtNode
    }
}

synctable nodeTable = NodeSchema<nodes>
synctable controlFlowTable = Schema<nodes>(nodeTable)

workbook {
    worksheet Template(controlFlowTable)
}
```

Figure 9.2: ActivityObjectFlow Example

10 StateMachines

StateMachine diagrams are used to define the different states that a system will exist in. This kind of diagram helps modelers to describe discrete, event-driven behaviors of the whole system or its parts.

10.1 VertexTransition

Description

MapleMBSE, in order to simplify *Transition* between *Vertexes,* supports a `vertexTransition` virtual feature that allows a better end user experience while inputting data.Note that MapleMBSE will fail to instantiate abstract classes like *Vertex* and it will be required to instantiate instead concrete classes like *Pseudostate*, *State* or *FinalState*. Nonetheless, *Vertex* can be used as reference to create *Transitions.* See the example section for further details.

Syntax

The general syntax for using the `vertexTransition` virtual feature is as follows:
`.alias:: vertexTransition`
Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.
The `vertexTransition` virtual feature must be used when querying the any kind of *Vertex* within a given *Region* of a *StateMachine*.

Using the VertexTransition Virtual Feature

The following example illustrates what you need to do to use the `vertexTransition` virtual feature:

1. Import the maplembse ecore with an alias.
2. Create an schema that navigates till an Vertex or which first dimension is an Vertex.
3. Make a dimension reference-query to another Vertex using .mse:: vertexTransition.
4. Complete the reference-decomposition.

This example has some extra schema, called `StateSchema`, used to create concrete *States*. The other schemas in this example will fail to instantiate *Element* because *Vertex* is an abstract class.
Note: some data sources specific to a fictional project were create in order to simplify the `reference-decomposition`, in a real life scenario you might need to identify the *Package*, the *StateMachine*, the *Region* and the *Vertex* that you want to connect to.

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="statemachine"]
data-source stm = pkg/packagedElement[StateMachine|name="stm"]
data-source rg = stm/region[Region|name="rg"]
data-source vertexes = rg/subvertex[Vertex]
data-source states = rg/subvertex[State]

synctable-schema VertexSchema {
    dim [Vertex] {
        key column /name as vName
    }
}

synctable-schema StateSchema {
    dim [State] {
        key column /name as sName
    }
}

worksheet-template StateTemplate(ssc: StateSchema) {
    vertical table tab1 at (2, 1) = ssc {
            key field sName
    }
}

synctable-schema Schema(vsc: VertexSchema) {
    dim [Vertex] {
        key column /name as vName
    }

    dim .mse::vertexTransition[Vertex] @ tgtRef {
        reference-decomposition tgtRef = vsc {
            foreign-key column vName as tgtVertex
        }
    }
}

worksheet-template Template(sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
            key field vName
            key field tgtVertex
    }
}

synctable vertexTable = VertexSchema<vertexes>
synctable stateTable = StateSchema<states>
synctable transitionTable = Schema<vertexes>(vertexTable)

workbook {
    worksheet StateTemplate(stateTable)
    worksheet Template(transitionTable)
}
```

Figure 10.1: VertexTransition Example

10.2 VerticalTransition

Description

MapleMBSE, in order to simplify *Transition* between *Vertexes,* supports a `vertical-Transition` virtual feature that allows a better end user experience while inputting data.Note that MapleMBSE will fail to instantiate abstract classes like *Vertex* and it will be required to instantiate instead concrete classes like *Pseudostate*, *State* or *FinalState*. Non-

etheless, *Vertex* can be used as reference to create *Transitions.* See the example section for further details.

Syntax

The general syntax for using the `verticalTransition` virtual feature is as follows:
`.alias:: verticalTransition`
Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.
The `verticalTransition` virtual feature must be used when querying the any kind of *Vertex* within a given *Region* of a *StateMachine*.

Using the VertexTransition Virtual Feature

The following example illustrates what you need to do to use the `vertexTransition` virtual feature:

1. Import the maplembse ecore with an alias.
2. Create an schema that navigates till an Vertex or which first dimension is an Vertex.
3. Make a dimension reference-query to another Vertex using .mse:: vertexTransition.
4. Complete the reference-decomposition.

11 Comments

11.1 ownedComments

Description

A comment is an element that represents a textual annotation that can be attached to other elements or a set of elements.
A comment can be owned by any element.
This virtual feature creates a comment and annotated it with the owner element.

Syntax

The general syntax for using the ownedComments virtual feature is as follows:
`/alias:: ownedComments`
Where the alias is the alias you assigned to the MapleMBSE ecore.

Using the ownedComments Virtual Feature

The following example shows you how to use the ownedComments feature.

1. Import the MapleMBSE ecore, as usual the alias used is `mse`
2. Inside a synctable-schema navigate to a *Class*
3. Within that dimension, define a regular column using `/mse::ownedComments[Comment]`
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`

Example

```
1   import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2   import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4   data-source Root[Model]
5   data-source blocks = Root/packagedElement[Package]
6
7   synctable-schema Schema {
8       record  dim [Package] {
9           key column /name as packageName
10      }
11
12      record dim /packagedElement[Class] {
13          key column /name as className
14          column /mse::ownedComments[Comment]/body as body
15      }
16  }
17
18  worksheet-template Template(ts: Schema) {
19      vertical table t at (1,1) = ts {
20          key field packageName
21          field className
22          field body :String
23          sort-keys packageName
24      }
25  }
26
27  synctable-schema Schema2 {
28      record  dim [Package] {
29          key column /name as packageName
30      }
31
32      record dim /packagedElement[Class] {
33          key column /name as className
34      }
35
36      record dim /mse::ownedComments[Comment] {
37          key column /body as body
38      }
39  }
40
41  worksheet-template Template2(ts: Schema2) {
42      vertical table t at (1,1) = ts {
43          key field packageName
44          field className
45          field body :String
46          sort-keys packageName
47      }
48  }
49
50  synctable Table1 = Schema<blocks>
51  synctable Table2 = Schema2<blocks>
52
53  workbook {
54      worksheet Template(Table1)
55      worksheet Template2(Table2)
56  }
```

12 Instance Matrices

SysML permits users to create an instance of the classifiers with their properties. If the classifier is defined with some properties, the instances will own slots that contain the properties defined. This instance allows users to create concrete elements from the more general model. In order to simplify the task related to InstanceSpecification, MapleMBSE proposes the following virtual features to support the creation, edition, and removal of instances and their Slots.

12.1 SlotValue

Description

SysML has a complex structure to access the values within Slots. Those values change widely depending on the type of the property defining the owing Slot. Imagine a real property defining Slot, which, in order to contain that value, requires a LiteralReal, and then the real value will be stored within that literal. Each type has its own literal class, and for reference to other instances the mechanisms are another matter altogether. This is just a reminder of how much complexity this InstanceSpecification modeling has, but thanks to this virtual feature, MapleMBSE simplifies and hides that complexity. Using a single access point and without caring about the concrete type of the property, SlotValue will return a string representing the value given Slot. MapleMBSE proposes an easy mechanism to display, create, and edit that first value associated to any Slot. Emphasis in first, SysML metamodel allows to associate several values to a single Slot, it is by design that MapleMBSE does not use this virtual feature for a different multiplicity.

Syntax

The general syntax for using the slotValue virtual feature is as follows:

```
column [Slot]/alias::slotValue as column_name
```

Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.
The slotValue virtual feature must be used while querying a Slot, and the return string can only be used within a column.

Using the SlotValue Virtual Feature

The following example illustrates what you need to do to use the slotValue virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing a Slot from an InstanceSpecification, see line 24.
3. Make sure that you are using the right combination of applied Classifier to the InstanceSpecification and the Slot's definingFeature.
4. Access that Slot's value using the slotValue virtual feature, see line 30

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="Slots"]
data-source blocks =  pkg/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]
data-source instances = pkg/packagedElement[InstanceSpecification]

synctable-schema PropertySchema {
          record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
                            key column /name as bName
                                      }
                            dim /ownedAttribute[Property] {
                                key column /name as pName
                                          }
}

synctable-schema Schema*(psc: PropertySchema){
          record dim [InstanceSpecification] {
                  key column /name as iName
                        }
                        dim /slot[Slot] {
                         key reference-query .definingFeature @pRef
                         reference-decomposition pRef = psc {
                             foreign-key column bName as bName
                             foreign-key column pName as pName
                               }
                               column /mse::slotValue as value
                            }
}

worksheet-template Template(sc: Schema) {
          vertical table tab1 at*(2, 1) = sc {
                 key field iName
                 key field bName
                 key field pName
                 field value
                 sort-keys iName, bName, pName
                 }
}

synctable propertyTable = PropertySchema<blocks>
synctable syncTable = Schema<instances>(propertyTable)

workbook {
worksheet Template(syncTable)
  }
```

12.2 InstanceTree

Description

SysML forces each Slot to be owned by an InstanceSpecification. The regular way to navigate would be from InstanceSpecification to Slot, and without any other mechanisms it

would be hard get a list of the InstanceSpecification tree for a given Slot. Remember that a Slot can have, as values, references to other InstanceSpecifications, and those would be part of tree for that given Slot. Returning this special tree list of InstanceSpecifications is the goal of instanceTree virtual feature.

Syntax

The general syntax for using the instanceTree virtual feature is as follows:

```
dim .alias::instanceTree[InstanceSpecification]
```

Where alias is the alias you assigned to the MapleMBSE ecore.

For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*. The instanceTree virtual feature must be used in a dimension level after querying a Slot, the return type is a list of reference to the InstanceSpecifications which belong to the tree of the queried Slot.

Using the InstanceTree Virtual Feature

The following example illustrates what you need to do to use the instanceTree virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing a Slot from an InstanceSpecification, see line 24.
3. The dimension after the Slot one should use the instanceTree, see line 32

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="Slots"]
data-source blocks =  pkg/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]
data-source instances = pkg/packagedElement[InstanceSpecification]

synctable-schema PropertySchema {
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as bName
    }

    dim /ownedAttribute[Property] {
        key column /name as pName
    }
}

synctable-schema Schema (psc: PropertySchema){
    record dim [InstanceSpecification] {
        key column /name as iName
    }

    record dim /slot[Slot] {
        key reference-query .definingFeature @pRef
        reference-decomposition pRef = psc {
            foreign-key column bName as bName
            foreign-key column pName as pName
        }
    }

    dim .mse::instanceTree[InstanceSpecification] {
        key column /name as iNameTree
    }
}

worksheet-template Template(sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
        key field iName
        key field bName
        key field pName
        key field iNameTree
        sort-keys iName, bName, pName
    }
}

synctable propertyTable = PropertySchema<blocks>
synctable syncTable = Schema<instances>(propertyTable)

workbook {
    worksheet Template(syncTable)
}
```

12.3 InstanceWithSlots

Description

It is well known that InstanceSpecifications and their Slots are an essential part of a useful and meaningful model. They are necessary to achieve results, but the task of instantiating, editing, and removing those elements is slow and error prone. MapleMBSE helps to create very complex structures using InstanceWithSlots, when you pass Class as parameter to an InstanceSpecification using this virtual feature, you will see how:

- MapleMBSE updates the list of classifiers that are applied to a given InstanceSpecification
- For each defining property related to that applied class, MapleMBSE will create a Slot defined by a property with its default value.

Syntax

To use instanceWithSlots virtual feature as a column within an InstanceSpecification dimension, the syntax is as follows:

```
reference-query .alias::instanceWithSlots @reference_name
```

This configuration line needs to be completed with a reference-decomposition that uses a Class schema, see the example for further information. Also remember that alias is the alias you assigned to the MapleMBSE ecore.

For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.

Using the InstanceWithSlots Virtual Feature

The following example illustrates one way to use the instanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an InstanceSpecification, see line 16.
3. Reference-query instanceWithSlots, see lines 18/19

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="Slots"]
data-source blocks =  pkg/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]
data-source instances = pkg/packagedElement[InstanceSpecification]

synctable-schema BlockSchema {
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as bName
    }
}

synctable-schema Schema (bsc: BlockSchema){
    record dim [InstanceSpecification] {
        key column /name as iName
        reference-query .mse::instanceWithSlots @ bRef
        reference-decomposition bRef = bsc {
            foreign-key column bName as cName
        }
    }
}

worksheet-template Template(sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
        key field iName
        field cName
    }
}

synctable blockTable = BlockSchema<blocks>
synctable syncTable = Schema<instances>(blockTable)

workbook {
    worksheet Template(syncTable)
}
```

12.4 RecursiveInstanceWithSlots

Description

The RecursiveInstanceWithSlots virtual feature does the same thing that InstanceWithSlots does but for all possible InstanceSpecifications in the tree. If a Class A is composed by other Class B and you use recursiveInstanceWithSlots to create an InstanceSpecification of Class A, MapleMBSE will also create an InstanceSpecification for Class B with Slots.

Syntax

To use recursiveInstanceWithSlots virtual feature as a column within an InstanceSpecification dimension, the syntax is as follows:

```
reference-query .alias::recursiveInstanceWithSlots @refer-
ence_name
```

This configuration line needs to be completed with a reference-decomposition that uses a Class schema, see the example for further information. Also remember that alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*.

Using the RecursiveInstanceWithSlots Virtual Feature

The following example illustrates one way to use the recursiveInstanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an InstanceSpecification, see line 16.
3. Reference-query recursiveInstanceWithSlots, see lines 18/19

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="Slots"]
data-source blocks =  pkg/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]
data-source instances = pkg/packagedElement[InstanceSpecification]

synctable-schema BlockSchema {
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as bName
    }
}

synctable-schema Schema (bsc: BlockSchema){
    record dim [InstanceSpecification] {
        key column /name as iName
        reference-query .mse::recursiveInstanceWithSlots @ bRef
        reference-decomposition bRef = bsc {
            foreign-key column bName as cName
        }
    }
}

worksheet-template Template(sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
        key field iName
        field cName
    }
}

synctable blockTable = BlockSchema<blocks>
synctable syncTable = Schema<instances>(blockTable)

workbook {
    worksheet Template(syncTable)
}
```

12.5 AttachedFile

Description

The attachedFile virtual feature supports MagicDraw file attachments, which are accessible through comments. It downloads all the relevant file attachments, and displays hyperlinks to temporary locations in the user interface. When the user clicks on the link, the user interface will open the file.

Syntax

The syntax to use attachedFile is as follows:

```
column /mse::attachedFile as fileName
```

Using the attachedFile Virtual Feature

The following example illustrates one way to use the recursiveInstanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an ownedComment line 5, in the column level call the attached VF see line 6 in the example code

Example

```
synctable-schema AttachedFileSchema {
record dim[Package]{
key column /name as PkgName
}
dim /ownedComment[Comment|mse::stereotypeNames="UML Standard
Profile::MagicDraw Profile::AttachedFile"]{
column /mse::attachedFile as fileName
}
}
```

12.6 Slots

Description

Use the slots virtual feature to add a new slot and display all the slots recursively under a top-level instance. The slots virtual feature can also be used to delete a child slot under a top level instance and then recreate the child slot.

Syntax

The syntax to use slots is as follows:

```
dim /mse::slots[Slot]
```

Using the slots Virtual Feature

The following example illustrates one way to use the Slots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing slots using the slots virtual feature (shown in line 6 in the slots VF get all the slots Recursively under a instance specification

Example

```
synctable-schema InstanceTable(blk : BlocksTable){
record dim [InstanceSpecification] {
key column /name as instanceName
}

dim /mse::slots[Slot] {
key reference-query .definingFeature[Property] @ dfRef
reference-decomposition dfRef = blk {
foreign-key column bName as bName
foreign-key column valName as valName
}
key column /mse::slotValue as cValue
}
}
```

12.7 ArrayName

Description

This feature is used to display the classifier of an instance along with its multiplicity. This feature gets a list of the sub-instance values for a given instance and returns the classifier, along with numbering, based on the list length. In the case of only one instance, numberings are not displayed. For example, if an Instance has two sub-instances due to the properties multiplicity, MapleMBSE will display the classifier name with array numbering i.e., Classifier[1], Classifier[2]

Syntax

/ alias::arrayName

Using the arrayName Virtual Feature

This feature works only in the context of instance specification.

The following example illustrates the use of the arrayName virtual feature.

1. Navigate to the sub-instance from a top-level instance as shown below.
2. Use the arrayName virtual feature to get the classifier name along with the sub-instance multiplicity.

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

workbook {
    worksheet InstanceMultiTable(instanceMultiplicitySchema)
}

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="InstanceView"]
data-source instan = pkg/packagedElement[Package|name="Instances"]
data-source instance = instan/packagedElement[InstanceSpecification
            |classifier=[Class|qualifiedName="SysML-Sample::InstanceView::Vehicle"]]

synctable-schema InstanceMultiplicitySchema {
    dim [InstanceSpecification|name = "vehicle_instance1"] {
        key column .classifier[Class]/name as blockName
    }
    optional {
        allow-empty dim /slot[Slot|definingFeature=[Property|
            mse::stereotypeNames="MD Customization for SysML::additional_stereotypes::
            PartProperty"]]/value[InstanceValue].instance[InstanceSpecification] {
            key column /mse::arrayName as propName
        }
    }
    dim ._instanceValueOfInstance[InstanceValue] {
        key column /mse::multiplicityOfInstance as multiVal
    }
}

synctable instanceMultiplicitySchema = InstanceMultiplicitySchema<instance>

worksheet-template InstanceMultiTable(ay : InstanceMultiplicitySchema){
    vertical table tab1 at (4,5) = ay{
        key field blockName
        key field propName
        key field multiVal
        unmapped-field
        sort-keys blockName,propName,multiVal
    }
}
```

12.8 MultiplicityOfInstance

Description

This feature is used to increase or decrease the number of sub-instances of the template instance (Top level instance based on which the rows of the instance matrix are displayed) that is defined for the rows of an instance matrix. It navigates using the instance classifier to get details of the property multiplicity, based on this value defined for the property in the model user can update the value in the table for the template instance.

Syntax

The general syntax for using the multiplicityOfInstance virtual feature is as follows: /alias::multiplicityOfInstance

Using the multiplicityOfInstance Virtual Feature

1. Multiplicity of Instance should be used in a table view that queries the InstanceSpecification and its sub-instances as shown in the example.
2. It is recommended that multiplicityOfInstance be used with the arrayName feature so that the tables can display all the available Instances and its sub-instances.

Example

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse

workbook {
    worksheet InstanceMultiTable(instanceMultiplicitySchema)
}

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name="InstanceView"]
data-source instan = pkg/packagedElement[Package|name="Instances"]
data-source instance = instan/packagedElement[InstanceSpecification
            |classifier=[Class|qualifiedName="SysML-Sample::InstanceView::Vehicle"]]

synctable-schema InstanceMultiplicitySchema {
    dim [InstanceSpecification|name = "vehicle_instance1"] {
        key column .classifier[Class]/name as blockName
    }
    optional {
        allow-empty dim /slot[Slot|definingFeature=[Property|
            mse::stereotypeNames="MD Customization for SysML::additional_stereotypes::
            PartProperty"]]/value[InstanceValue].instance[InstanceSpecification] {
            key column /mse::arrayName as propName
        }
    }
    dim ._instanceValueOfInstance[InstanceValue] {
        key column /mse::multiplicityOfInstance as multiVal
    }
}

synctable instanceMultiplicitySchema = InstanceMultiplicitySchema<instance>

worksheet-template InstanceMultiTable(ay : InstanceMultiplicitySchema){
    vertical table tab1 at (4,5) = ay{
        key field blockName
        key field propName
        key field multiVal
        unmapped-field
        sort-keys blockName,propName,multiVal
    }
}
```

13 Recursivity

13.1 getRecursively

Description

The getRecursively virtual feature works as a chained data source, traversing all subelements recursively under the owner data source or QPE and then filters out elements matching the qualifier and filter.

Syntax

The general syntax for using the getRecursively virtual feature is as follows:

```
data-source packages = Root/packagedElement[Pack-
age|name="C3"]/getRecursively[Package]
```

Where C3 is the name of the package, class, port, etc. For this example, all packages under C3 will be retrieved.

```
data-source packages = Root/packagedElement[Package]/getRe-
cursively[Package]
```

In the above data source syntax example, all the packages under root are retrieved. After that, all the elements (packages) under those packages are retrieved, recursively. When you are adding a new element, in this case, it will go under one of the packages which was retrieved from the root.

```
data-source packagesC3Class = Root/mse::getRecursively[Pack-
age|name="C3"]/packagedElement[Class]
```

In the syntax example above, first, getRecursively finds the packages under the model. The model may have more than one C3 package.

Note that this is maybe very inefficient when the model is big, and it would be much faster to explicitly specify the path for each existing C3 package.

```
data-source packagesC3Class = Root/mse::getRecursively[Pack-
age|name="C3"]/ mse::getRecursively[Class]
```

This data source gets all packages recursively and then sorts them and shows the ones named C3. After that, you get all classes under these C3 packages and any of their subpackages.

Using the getRecursively Virtual Feature

The following example shows you how to use the ownedComments feature.

1. Import the MapleMBSE ecore, as usual the alias used is `mse`
2. Inside a synctable-schema navigate to a *Package*
3. In the next dimension, use `/mse::getRecursively[Class]` to get all the class under the top package(Previous dim/root dim) and sub packages
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`

Example

In this example, the code snippet retrieves all the packages and sub packages under the package C3

```
synctable-schema PackageTable {
    record dim [Package] {
        key column /name as pName
    }
    record dim / mse:: getRecursively[Class] {
       key column /name as cName
    }
}
```

This feature can be used anywhere in a QPE or data source, but not at the start of QPE or data source.

```
synctable-schema PackageTable {
    record dim [Package] {
        key column /name as Name
    }
    record dim / mse:: getRecursively[Package] {
       key column /name as Name1
   }
}
```

14 Constraints

SysML makes it hard to accessing the minimum and maximum constraint data in SysML can be difficult because the model is forced to use LiteralString and other elements (e.g. TimeExpression, Constraints). The main purpose of the virtual feature in this chapter is to allow the MSE file to access this data with ease and aggregate it into simple double period notation (..). Another benefit to the use of a virtual feature for working with constraint data is that the end user has fewer inputs to provide, reducing human error.

14.1 durationConstraint

Description

To display and set duration constraints, MapleMBSE provides a virtual feature that allows the simultaneous creation and editing of the min and max limits of the constraint using a simple double dot notation (for example, minConstraint..maxConstraint). In the case of both minConstraint and maxConstraint representing numerical values (in decimal or scientific notation), MapleMBSE performs a check to determine if the minConstraint value is less than or equal to the maxConstraint value.

Syntax

The general syntax for using the `durationConstraint` virtual feature is as follows:

```
column /mse::durationConstraint as dcValue
```

For user input, the durationConstraint virtual feature accepts numerical values as well as arbitrary string values.These values are joined by double periods (..). In addition, you can use an escape character (\) to include periods as part of the minimum and maximum constraint values.

	A	B
1	Constraint Name	Value
2	Minor	0..18
3	Voter	18..99
4	TimeConfTx	10..60
5	Temp	cold..hot

The table below gives specific examples of both valid and invalid syntax for use of double periods.

User Input Example	Minimum Constraint Value	Maxim Constraint Value	Comment
`2..3.5`	`2`	`3.5`	Valid
`-1..3.2`	`-1`	`2.3`	Valid
`5.5..2`	`5.5`	`2`	Invalid. The upper bound is less than lower bound
`2.3 .. 2.3`	`2.3`	`2.3`	Valid. Numerical values are trimmed, and both bounds are allowed to be equal
`1....3`	`1`	`..3`	Valid. First double dot is taken as the separator
`abc..foo bar`	`abc`	`foo bar`	Valid. String values can contain spaces
`1.3`			Invalid. No double period present
`..`			Valid. Empty Min/Max is allowed
`min..`	`min`		Valid. Empty max
`..max`		`max`	Valid. Empty min
`escape\....dots`	`escape.`	`.dots`	Valid. Escaping first period
`invalid\..input`			Invalid. No double period present
`2 . 5 .. 2.4`	`2 .5`	`2.4`	Valid. The min is not a valid number because of the space.

Using the durationConstraint Virtual Feature

The following example shows you how to use the `durationConstraint` feature.

1. Import the MapleMBSE ecore, as usual the alias used is `mse`.
2. Inside a synctable-schema navigate to a `DurationConstraint`, in this case `/ownedRule[DurationConstraint]` within an Activity.
3. Within that dimension, define a regular column using `/mse::durationConstraint`.
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`.

Example

```
synctable-schema Schema {
 record dim [Activity] {
  key column /name as aName
 }
 record dim /ownedRule[DurationConstraint] {
  key column /name as dcName
  column /mse::durationConstraint as dcValue
 }
}
```

15 Generalization

This section contains all other virtual features that do not create elements but offer a better alternative to access and map model information.

15.1 specificClass

Description

The `specificClass` virtual feature provides a simple, more direct way of creating generalizations between a more generalized element and a more specialized element. The `specificClass` virtual feature also sets the values for the specific and general elements and then stores the generalization relationship information in the specific class.

Syntax

The general syntax for using the specificClass virtual feature is as follows:

`.mse::specificClass`

Using the specificClass Virtual Feature

The following example illustrates one way to use the `specificClass` virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a datasource which query the Blocks from the model ”`../PackageElement-Class|mse::metaclassName="SysML::Blocks::Block"]`”
3. In the synctable schema start from the generalized block
4. For the next dimension, use the virtual feature which uses the reference decomposition to create the generalization between the generalized block and the specific block

Example

```
synctable-schema BlockpropertiesTable(blocks: BlocksTable){
 record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
  key column /name as generalClassName
 }
 record dim .mse::specificClass[Class] @genname{
  reference-decomposition genname = blocks{
   foreign-key column BlockName as specificClassName
  }
```

```
  }
}
```

16 Working with sysML Diagrams

This chapter describes how to work with a sysML Diagram.

16.1 downloadDiagram

Description

Use the downloadDiagram feature to download the sysML diagram associated with your model, using the MagicDraw or Cameo.
To use this feature, first install the MapleMBSE plugin (see the MapleMBSE Installation Guide for instructions) into MagicDraw and Cameo. Start MagicDraw or Cameo and open the Project from which you want to download the diagram in MapleMBSE. Start MapleMBSE.

Syntax

mse::downloadDiagram

Using the clientDependencies Virtual Feature

1. Import the ecore statement e.g “import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse”
2. In the Synctable schema start the dimension from the Diagram object.
3. Inside the Diagram dimension use the downloadDiagram feature “mse::downloadDiagram”

Example

```
synctable-schema Schema {
    dim[Package]{
        key column /name as Pkg
    }
    dim /ownedDiagram[Diagram]{
        key column /mse::downloadDiagram as diagraName
        column /mse::diagramType as diagramType
    }
}
```

16.2 diagramType

Description

The diagramType virtual feature shows the type of diagram which was downloaded using the downloadDiagram feature. It cannot be used individually. It can only be used after the downloadDiagram virtual feature.

Syntax

mse::downloadDiagram

Using the supplierDependencies Virtual Feature

1. Import the ecore statement e.g “import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse”
2. In the Synctable schema start the dimension from the Diagram object
3. Inside the Diagram dimension use the downloadDiagram feature “mse::downloadDiagram” after that use the “mse::diagramtype”

Example

```
synctable-schema Schema {
    dim[Package]{
        key column /name as Pkg
    }
    dim /ownedDiagram[Diagram]{
        key column /mse::downloadDiagram as diagraName
        column /mse::diagramType as diagramType
    }
}
```

17 File Attachments

17.1 AttachedFile

Description

The attachedFile virtual feature supports MagicDraw file attachments, which are accessible through comments. It downloads all the relevant file attachments, and displays hyperlinks to temporary locations in the user interface. When the user clicks on the link, the user interface will open the file.

Syntax

The syntax to use attachedFile is as follows:

```
column /mse::attachedFile as fileName
```

Using the attachedFile Virtual Feature

The following example illustrates one way to use the attachedFile virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an ownedComment line 5, in the column level call the attached VF see line 6 in the example code

Example

```
synctable-schema AttachedFileSchema {
record dim[Package]{
key column /name as PkgName
}
dim /ownedComment[Comment|mse::stereotypeNames="UML Standard
Profile::MagicDraw Profile::AttachedFile"]{
column /mse::attachedFile as fileName
}
}
```

18 Element Type

This chapter describes virtual featues that can be used to find the type of model element

18.1 elementType

Description

The elementType virtual feature helps to identify the metaclass type of model element (Class, Activity, etc) that is displayed in MapleMBSE.

Syntax

The general syntax for the elementType virtual feature is as follows:

`/alias::elementType`

Where:

- `alias` is the alias you assigned to the MapleMBSE ecore.
- `elementType` virtual feature is used to display the metaclass Type of an element.

Using the elementType Virtual Feature

1. Import the ecore statement e.g “import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse”
2. In the data-source we add the elementType to the Qualifier sorting feature.

Example

`data-source elementsNew*[NamedElement||elementFilter|mse::elementType]`

Where:

- `NamedElement` can be any of Class, Activity, StateMachine, etc.
- `elementFilter` can be name, visibility, stereotype, etc.
- `mse::elementType` refers to the elementType virtual feature.

```
import-ecore "http://www.nomagic.com/magicdraw/UML/2.5.1.1"
import-ecore "http://maplembse.maplesoft.com/twc/1.0" as mse
data-source Root[Model]
```

```
data-source elementList = Root/packagedElement[Package|name="Structure"]
synctable-schema DifferentElementTypeSchema{
    dim [NamedElement]{
        key column /name as differentElementName
        column /mse::elementType as typeName
    }
}
synctable differentElementTypeSchema =
DifferentElementTypeSchema<elementList>
worksheet-template DifferentElements(fis : DifferentElementTypeSchema){
    vertical table featureTable at (5,4) = fis {
        key field differentElementName
        key field typeName
        sort-keys differentElementName
    }
}
workbook{
    worksheet DifferentElements(differentElementTypeSchema)
}
```

Index

A

activity diagrams, 43
ActivityControlFlow, 43
description, 43
syntax, 43
using, 43
ActivityObjectFlow, 45
description, 45
syntax, 45
using, 45
associations, 11
associatedProperty, 13
description, 11
syntax, 11
using, 12
directedAssociatedProperty, 15
description, 14
syntax, 14
using, 14
nestedDirectedComposition, 19
description, 19
syntax, 19
using, 19
otherAssociatedEnd, 17
description, 16
syntax, 16
using, 16

B

blocks, 21
getAllProperties, 25
description, 23
example, 26
syntax, 25
using, 25
propertyDefaultValue, 22
recursivePartProperties, 21
description, 21
syntax, 21
using, 21

C

comments, 51
ownedComments, 53
description, 51
syntax, 51
using, 51
connectors, 27
connectedPropertyOrPort, 27
description, 27
syntax, 27
using, 28
otherConnectorEnd, 29
description, 29
syntax, 29
using, 29
constraints, 73
durationConstraint, 75
description, 73
syntax, 73
using, 74

D

dependencies, 31
clientDependencies, 31
description, 31
syntax, 31
using, 31
supplierDependencies, 33
description, 32
syntax, 33
using, 33

E

Element Type, 83
elementType, 83
description, 83
example, 83
using, 83
elementType syntax, 83
enumeration, 37

enumerationLabel, 39
description, 38
syntax, 38
enumerationName, 37
description, 37
syntax, 37
using, 37

F

File Attachments, 81
attachedFile, 81
description, 81
syntax, 81
using, 81

G

Generalization, 77
specificClass, 77
description, 77
example, 77
syntax, 77
using, 77

I

Instance Matrices, 55
arrayName, 65
description, 65
syntax, 65
attachedFile, 63
description, 63
syntax, 63
using, 64
instanceTree, 59
description, 57
syntax, 58
using, 58
instanceWithSlots, 60
description, 60
syntax, 60
using, 60
multiplicityOfInstance, 67
description, 66
example, 67
syntax, 67
using, 67
RecursiveInstanceWithSlots, 61
description, 61
syntax, 62
using, 62
slots, 64
description, 64
syntax, 64
using, 64
slotValue, 55
description, 55
syntax, 55
using, 56

M

Multiple Dependencies, 36
Creating, 36
Example, 36
Introduction, 36

R

recursivity, 69
getRecursively, 71
description, 69
syntax, 69
using, 70

S

specificClass, 77
statemachines, 47
vertexTransition, 47
description, 47
syntax, 47
using, 47, 49
verticalTransition, 49
description, 48
syntax, 49
stereotypes, 5
featureName, 7
description, 6, 7

syntax, 7
metaclassName, 5
description, 5
syntax, 5
using, 5
stereotypeNames, 9
description, 9
syntax, 9
using, 9
sysML Diagrams, 79
diagramType, 80
description, 80
example, 80
syntax, 80
using, 80
downloadDiagram, 79
description, 79
example, 79
syntax, 79
using, 79

U

util, 41
multiplicityProperty, 41
description, 41
syntax, 41
using, 41