Capella 2024.1 Virtual Features Guide

Capella 2024.1 Virtual Features Guide

Contents

List of Figures

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 2024 requires the following products.

- Microsoft® Excel® 2010 Service Pack 2, Excel 2016 or Excel 2019.
- 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 specifed 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.

If you are using IBM® Rational® Rhapsody® with MapleMBSE, the following versions are supported:

- Rational Rhapsody Version 8.1.5, 8.3 and 8.4
- Teamwork Cloud™ server 19.0 SP4 or 2021.x

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

- 5.1.0 or 5.2.0

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

- 2020-3

Note that the architecture of the supported non-server products is 64-bit.

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 either on your MapleMBSE installation DVD or the folder where you installed MapleMBSE.
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, Capella ecore and Model-based Systems Engineering concepts and who intend to create their own MapleMBSE configuration files.

1.2 Perquisite Knowledge

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

- Basic understanding of Capella and UML ecore to know the Classifiers, attributes and references. 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

Capella manages different architectural level and updates the models consistently maintaining the model complexity and traceability between the elements. Maintaining the same using MapleMBSE requires users to enter various information that might not be required or relevant when working with Capella directly.

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 is interested in taking advantage of the modeling capabilities of ARCADIA, 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 Capella Ecore, they themselves are not part of Capella Ecore.

With the right choice of labels within an Excel template and a well-designed configura-

tion(.mse) file that implements MapleMBSE virtual features, an end user can enter a couple of inputs in a spreadsheet and create Capella Elements and linking them, or Ports and other combinations of elements

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 Capella ecore:
"http://www.polarsys.org/capella/core/modeller/5.0.0"
"http://www.polarsys.org/capella/core/ctx/5.0.0"
"http://www.polarsys.org/capella/core/la/5.0.0"
"http://www.polarsys.org/capella/core/cs/5.0.0"
"http://www.polarsys.org/capella/core/information/5.0.0"
"http://www.polarsys.org/capella/core/core/5.0.0"
"http://www.polarsys.org/capella/core/oa/5.0.0"
"http://www.polarsys.org/capella/core/fa/5.0.0"
"http://www.polarsys.org/capella/core/pa/5.0.0"

To specify the MapleMBSE ecore:
"http://maplembse.maplesoft.com/common/1.0"
"http://maplembse.maplesoft.com/capella/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/capella/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, abstract-TraceTargetElement.

2 Relations

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

2.1 abstractTraceTargetElement

Description

An `abstractTraceTargetElement` is used between two model elements to represent a relationship where the owner of the element is the source and the target element is given by the user.

Syntax

The general syntax for using the abstractTraceTargetElement virtual feature is as follows:
`/mse:: abstractTraceTargetElement`
This virtual feature is used while querying a model element that has to be assigned as target to the relation that is being created and is used in a dimension where the relation is being querried.

Using the abstractTraceTargetElement Virtual Feature

In general, the following steps outline how to use abstractTraceTargetElement:

1. It should be used when a relationship with sourceElement and targetElement is queried.
2. When querying the model element with mse:: abstractTraceTargetElement, the reference decomposition should be to a target element. The element in the previous dim is assigned as source.

Example

```
import-ecore "http://www.polarsys.org/capella/core/modeller/5.0.0"
import-ecore "http://www.polarsys.org/capella/core/ctx/5.0.0" as sa
import-ecore "http://www.polarsys.org/capella/core/la/5.0.0" as la
import-ecore "http://www.polarsys.org/capella/core/cs/5.0.0" as cs
import-ecore "http://www.polarsys.org/capella/core/information/5.0.0" as info
import-ecore "http://www.polarsys.org/capella/core/core/5.0.0" as core
import-ecore "http://www.polarsys.org/capella/core/oa/5.0.0" as oa
import-ecore "http://www.polarsys.org/capella/core/fa/5.0.0" as fa
import-ecore "http://www.polarsys.org/capella/core/pa/5.0.0" as pa

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

workbook {
worksheet SALAFunctionsTraceability(SALAFunctionTraceability,logicalFunctionsTable,systemFunctionsTable)
}

//Functional Realization between System Functions & Logical Functions

data-source Root[Project]
data-source SystemAnalysis = Root/ownedModelRoots[SystemEngineering]/ownedArchitectures[sa::SystemAnalysis]
data-source SAFunctionPkg = SystemAnalysis/ownedFunctionPkg[sa::SystemFunctionPkg]
data-source RootSystemFunction = SAFunctionPkg/ownedSystemFunctions[sa::SystemFunction]
data-source systemFunctions = RootSystemFunction/mse::recursiveElements[sa::SystemFunction]
data-source LogicalArchitecture = Root/ownedModelRoots[SystemEngineering]/ownedArchitectures[la::LogicalArchitecture]
data-source LAFunctionPkg = LogicalArchitecture/ownedFunctionPkg[la::LogicalFunctionPkg]
data-source RootLogicalFunction = LAFunctionPkg/ownedLogicalFunctions[la::LogicalFunction]
data-source logicalFunctions = RootLogicalFunction/mse::recursiveElements[la::LogicalFunction]

synctable-schema SFunctionsTable {
record dim [sa::SystemFunction] {
key column /name as SFName1
column /description as SFDesc1
}
}

synctable-schema LFunctionsTable {
record dim [la::LogicalFunction] {
key column /name as LFName1
column /description as LFDesc1
}
}

synctable systemFunctionsTable = SFunctionsTable<systemFunctions>
synctable logicalFunctionsTable = LFunctionsTable<logicalFunctions>

synctable-schema FunctionTraceability (abc: SFunctionsTable) {
dim [la::LogicalFunction] {
key column /name as LFName1
}
record dim /ownedFunctionRealizations[fa::FunctionRealization]{
column /id as test
key reference-query .mse::abstractTraceTargetElement[sa::SystemFunction] @ functions //Using abstractTraceTargetElement Virtual
Feature
reference-decomposition functions = abc {
foreign-key column SFName1 as SFName1
}
}
}

synctable SALAFunctionTraceability = FunctionTraceability<logicalFunctions>(systemFunctionsTable)

worksheet-template SALAFunctionsTraceability(mat: FunctionTraceability, lf: LFunctionsTable, sf: SFunctionsTable ) {
matrix Matrix1 at (4,4) = mat {
const-field "X"
row-index = lf {
unmapped-field
key field LFName1
unmapped-field
sort-keys LFName1
}
column-index = sf {
unmapped-field
key field SFName1
unmapped-field
sort-keys SFName1
}
}
}
```

Figure 2.1: abstractTraceTargetElement Example

2.2 referencingElements

Description

It is only possible to query elements from an owned relation to the target elements. This virtual feature can query the opposite reference of a relation such as Component Exchange, Functional Exchange etc.,. Using this feature you can create queries to get to the source of a relation from the target element. The types of Capella
relations supported by this feature are information flows, requirement trace and activity edges. This feature can be used to only query the elements from the model adding new elements with relations are not supported.

Syntax

alias::referencingElements

Using the referencingElements Virtual Feature

1. Referencing elements should be used only when querying the opposite references for elements linked with relations.
2. It is necessary to add the Classifier in the query for referencing elements to filter based on the element type should be displayed

Example

2.3 capellaIncomingRelation

Description

Adding new Capella relations between elements requires users to provide both the source and target of the relations. Even though this is desirable in some cases most Capella relations have the owner as the source element. The capellaIncomingRelation virtual feature will be always assigned the owner of the relation to be the source. With this feature the user is not required to add redundant information about the source when adding new relations.

Syntax

.alias::capellaIncomingRelation

Using the capellaIncomingRelation Virtual Feature

1. This feature has to be used with the CapellaIncomingRelation dimension as shown in example.
2. Add reference-query and reference decomposition to add/edit the relations.

Example

```
import-ecore "http://www.polarsys.org/capella/core/modeller/5.0.0"
import-ecore "http://www.polarsys.org/capella/core/ctx/5.0.0" as sa
import-ecore "http://www.polarsys.org/capella/core/la/5.0.0" as la
import-ecore "http://www.polarsys.org/capella/core/cs/5.0.0" as cs
import-ecore "http://www.polarsys.org/capella/core/information/5.0.0" as info
import-ecore "http://www.polarsys.org/capella/core/core/5.0.0" as core
import-ecore "http://www.polarsys.org/capella/core/oa/5.0.0" as oa
import-ecore "http://www.polarsys.org/capella/core/fa/5.0.0" as fa
import-ecore "http://www.polarsys.org/capella/core/pa/5.0.0" as pa
import-ecore "http://maplembse.maplesoft.com/capella/1.0" as mse

data-source Root[Project]
data-source AllLogicalFunctions*[la::LogicalFunction]
data-source AllLogicalComponent*[la::LogicalComponent]

//LFunction to allocated Component
synctable-schema LogicalFunctionAlSchema{
    record dim [la::LogicalFunction]{
        key column /name as lfunctionName
    }
    record dim .mse::referencingElements[la::LogicalComponent]{
        key column /name as compAlFn
    }
}

synctable logicalFunctionAlSchema = LogicalFunctionAlSchema<AllLogicalFunctions>

worksheet-template LogicalFunction(fn : LogicalFunctionAlSchema){
    vertical table tabl at (4,5) = fn{
        key field lfunctionName
        key field compAlFn
    }
}

workbook {
    worksheet LogicalFunction(logicalFunctionAlSchema){label ="LogicalFnAllocationTable"}
}
```

2.4 capellaOutgoingRelation

Description

Adding new Capella relations between elements requires users to provide, both the source and target of the relations. Even though this is desirable in some cases, most Capella relations

have the owner as the source element. The capellaOutgoingRelation virtual feature will be always assigned the owner of the relation to be the source. With this feature the user is not required to add redundant information about the source when adding new relations.

Syntax

.alias::capellaOutgoingRelation

Using the capellaOutgoingRelation Virtual Feature

1. This feature has to be used with the CapellaOutgoingRelation dimension as shown in example.
2. Add reference-query and reference decomposition to add/edit the relations.

Example

```
import-ecore "http://www.polarsys.org/capella/core/information/5.0.0" as info
import-ecore "http://www.polarsys.org/capella/core/core/5.0.0" as core
import-ecore "http://www.polarsys.org/capella/core/pa/5.0.0" as pa
// Requirements Plug-in
import-ecore "http://www.polarsys.org/capella/requirements" as reqcore
import-ecore "http://www.polarsys.org/kitalpha/requirements" as reqs

import-ecore "http://maplembse.maplesoft.com/capella/1.0" as mse
data-source Root[Project]
data-source physarchitecture = Root/ownedModelRoots[SystemEngineering]/ownedArchitectures[pa::PhysicalArchitecture]
data-source requirements = physarchitecture/ownedExtensions[reqcore::CapellaModule]/ownedRequirements[reqs::Requirement]
data-source AllPhyComp*[pa::PhysicalComponent]

synctable-schema PhysicalComponentSchema{
    dim[pa::PhysicalComponent]{
        key column /name as phyCompName
    }
}

synctable-schema RequirementTable(pcs: PhysicalComponentSchema){
    record dim[reqs::Requirement]{
        key column /ReqIFLongName as requirementName
        column /ReqIFIdentifier as reqId
        column /ReqIFText as reqText
        column /ownedAttributes[reqs::StringValueAttribute]/value as usercomment
    }
    dim /ownedRelations[reqcore::CapellaOutgoingRelation]{
        key reference-query .mse::CapellaOutgoingRelation[pa::PhysicalComponent] @tgtref
        reference-decomposition tgtref = pcs{
            foreign-key column phyCompName as phyCompName
        }
    }
}

synctable physicalComponentsSchema = PhysicalComponentSchema<AllPhyComp>
synctable requirementsTable = RequirementTable<requirements>(physicalComponentsSchema)

worksheet-template Requirements(rq : RequirementTable){
    vertical table tabl at (4,5) = rq{
        field reqId
        key field requirementName
        field reqText
        key field phyCompName
        sort-keys reqId, requirementName
    }
}
```

3 Recursion

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

3.1 recursiveElements

Description

The `recursiveElements` virtual feature works as a chained data source, traversing all sub elements recursively under the owner data source or QPE and then filters out elements matching the qualifier and filter.
Note: This virtual feature is read-only in the sense new elements cannot be added, but useful to set and remove references.

Syntax

The general syntax for using the recursiveElements virtual feature is as follows:
`/mse:: recursiveElements`

Using the recursiveElements Virtual Feature

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

1. It should be used when a flat list of all the sub-elements from a top-level element or Architecture should be displayed. Example to list all System Functions from System Architecture level or from Root System Function.
2. When querying the model element with `mse:: recursiveElements` it can be used at the data-source Level or inside a dim.
3. When using recursiveElements specify the type of element to be displayed in the Qualifier. For example, `/mse::recursiveElements[la::LogicalFunction]`

Example

```
import-ecore "http://www.polarsys.org/capella/core/modeller/5.0.0"
import-ecore "http://www.polarsys.org/capella/core/ctx/5.0.0" as sa
import-ecore "http://www.polarsys.org/capella/core/la/5.0.0" as la
import-ecore "http://www.polarsys.org/capella/core/cs/5.0.0" as cs
import-ecore "http://www.polarsys.org/capella/core/information/5.0.0" as info
import-ecore "http://www.polarsys.org/capella/core/core/5.0.0" as core
import-ecore "http://www.polarsys.org/capella/core/oa/5.0.0" as oa
import-ecore "http://www.polarsys.org/capella/core/fa/5.0.0" as fa
import-ecore "http://www.polarsys.org/capella/core/pa/5.0.0" as pa

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

workbook {
worksheet SALAFunctionsTraceability(SALAFunctionTraceability,logicalFunctionsTable,systemFunctionsTable)
}

//Functional Realization between System Functions & Logical Functions

data-source Root[Project]
data-source SystemAnalysis = Root/ownedModelRoots[SystemEngineering]/ownedArchitectures[sa::SystemAnalysis]
data-source SAFunctionPkg = SystemAnalysis/ownedFunctionPkg[sa::SystemFunctionPkg]
data-source RootSystemFunction = SAFunctionPkg/ownedSystemFunctions[sa::SystemFunction]

data-source systemFunctions = RootSystemFunction/mse::recursiveElements[sa::SystemFunction] //Using recursiveElements Virtual
Feature

data-source LogicalArchitecture = Root/ownedModelRoots[SystemEngineering]/ownedArchitectures[la::LogicalArchitecture]
data-source LAFunctionPkg = LogicalArchitecture/ownedFunctionPkg[la::LogicalFunctionPkg]
data-source RootLogicalFunction = LAFunctionPkg/ownedLogicalFunctions[la::LogicalFunction]

data-source logicalFunctions = RootLogicalFunction/mse::recursiveElements[la::LogicalFunction] //Using recursiveElements Virtual
Feature

synctable-schema SFunctionsTable {
record dim [sa::SystemFunction] {
key column /name as SFName1
column /description as SFDesc1
}
}

synctable-schema LFunctionsTable {
record dim [la::LogicalFunction] {
key column /name as LFName1
column /description as LFDesc1
}
}

synctable systemFunctionsTable = SFunctionsTable<systemFunctions>
synctable logicalFunctionsTable = LFunctionsTable<logicalFunctions>

synctable-schema FunctionTraceability (abc: SFunctionsTable) {
dim [la::LogicalFunction] {
key column /name as LFName1
}
record dim /ownedFunctionRealizations[fa::FunctionRealization]{
column /id as test
key reference-query .mse::abstractTraceTargetElement[sa::SystemFunction] @ functions
reference-decomposition functions = abc {
foreign-key column SFName1 as SFName1
}
}
}

synctable SALAFunctionTraceability = FunctionTraceability<logicalFunctions>(systemFunctionsTable)

worksheet-template SALAFunctionsTraceability(mat: FunctionTraceability, lf: LFunctionsTable, sf: SFunctionsTable ) {
matrix Matrix1 at (4,4) = mat {
const-field "X"
row-index = lf {
unmapped-field
key field LFName1
unmapped-field
sort-keys LFName1
}
column-index = sf {
unmapped-field
key field SFName1
unmapped-field
sort-keys SFName1
}
}
}
```

Figure 3.1: recursiveElements Example

Index

R