Modelica Change Proposal MCP-0014
Conversion
Status: Scheduled for Modelica 3.4

Proposed Changes to the Modelica Language Specification
Version 3.3 Revision 1

Table of Contents

Preface ... 3
Chapter 1 Introduction ... 3
Chapter 2 Lexical Structure ... 3
Chapter 3 Operators and Expressions ... 3
Chapter 4 Classes, Predefined Types, and Declarations ... 3
Chapter 5 Scoping, Name Lookup, and Flattening ... 3
Chapter 6 Interface or Type Relationships ... 4
Chapter 7 Inheritance, Modification, and Redeclaration ... 4
Chapter 8 Equations ... 4
Chapter 9 Connectors and Connections ... 4
Chapter 10 Arrays ... 4
Chapter 11 Statements and Algorithm Sections ... 5
Chapter 12 Functions ... 5
Chapter 13 Packages ... 5
Chapter 14 Overloaded Operators ... 5
Chapter 15 Stream Connectors ... 5
Chapter 16 Synchronous Language Elements ... 6
Chapter 17 State Machines ... 6
Chapter 18 Annotations ... 6
Chapter 19 Unit Expressions ... 8
Chapter 20 The Modelica Standard Library ... 8
Appendix A Glossary ... 8
Appendix B Modelica Concrete Syntax ... 8
Appendix C Modelica DAE Representation ... 9
Appendix D Derivation of Stream Equations ... 9

Preface

Chapter 1

Introduction

Chapter 2

Lexical Structure

Chapter 3

Operators and Expressions

Chapter 4

Classes, Predefined Types, and Declarations

Chapter 5

Scoping, Name Lookup, and Flattening

Chapter 6

Interface or Type Relationships

Chapter 7

Inheritance, Modification, and Redeclaration

Chapter 8

Equations

Chapter 9

Connectors and Connections

Chapter 10

Arrays

Chapter 11

Statements and Algorithm Sections

Chapter 12

Functions

Chapter 13

Packages

Chapter 14

Overloaded Operators

Chapter 15

Stream Connectors

Chapter 16

Synchronous Language Elements

Chapter 17

State Machines

Chapter 18

Annotations

18.8.2 Version Handling

Update 3rd bullet to the following, and the rest of the text after bulleted list:

- `conversion ( from (version = `*`Versions`*`, [to=VERSION-NUMBER,] `*`Convert`*`) )`
  where *Versions* is `VERSION-NUMBER` | `{VERSION-NUMBER,VERSION-NUMBER,…}`
  and *`Convert`* `is script="…" | change={conversionRule(), …, conversionRule()}`
  Defines that user models using the `VERSION-NUMBER` or any of the given `VERSION-NUMBER` can be upgraded to the given `VERSION-NUMBER` (if the to-tag is missing this is the `CURRENT-VERSION-NUMBER`) of the current class by applying the given conversion rules. The script consist of an unordered sequence of conversionRule(); (and optionally Modelica comments). The conversionRule functions are defined in 18.8.2.1. *[The to-tag is added for clarity and optionally allows a tool to convert in multiple steps.]*

The from-version-tags should be unique among the different conversion-elements, indicating that based on an existing library version it is possible to uniquely determine which conversion to use.

18.8.2.1 Conversion rules

There are a number of functions: convertClass, convertClassIf, convertElement, convertModifiers, convertMessage defined as follows. The calls of these functions do not directly convert, instead they define conversion rules as below. The order between the function calls does not matter, instead the longer paths (in terms number of hierarchical names) are used first as indicated below, and it is an error if there are any ambiguities.

These functions can be called with literal strings or array of strings and vectorize according to 12.4.6.

Both convertElement and convertModifiers only use inheritance among user models, and not in the library that is used for the conversion – thus conversions of base-classes will require multiple conversion-calls; this ensures that the conversion is independent of the new library structure. The class-name used as argument to convertElement and convertModifiers is similarly the old name of the class, i.e. the name before it is possibly converted by convertClass. [*This allows the conversion to be done without access to the old version of the library (by suitable modifications of the lookup). Another alternative is to use the old version of the library during the conversion.*]

convertClass("OldClass","NewClass")

Convert class `OldClass` to `NewClass`.
Match longer path first, so if converting both A to C and A.B to D then A.F is converted to C.F and A.B.E to D.E. This is considered before convertMessage for the same `OldClass`.

convertClassIf("OldClass", "oldElement", "whenValue", "NewClass")

Convert class `OldClass` to `NewClass` if the literal modifier for `oldElement` has the value `whenValue`, and also remove the modifier for `oldElement`.

These are considered before `convertClass` and `convertMessage` for the same `OldClass`.

convertElement("OldClass","OldName","NewName")

In `OldClass` convert element `OldName` to `NewName`. Both `OldName` and `NewName` normally refer to components – but they may also refer to class-parameters, or hierarchical names. For hierarchical names the longest match is used first.

convertModifiers

```
convertModifiers("OldClass",{"OldModifier1=default1","OldModifier2=default2",...},{"
NewModifier1=...%OldModifier1%"} [, simplify=true] )
```

Normal case; if any modifier among `OldModifier` exist then replace all of them with the `NewModifiers`. The defaults (if present) are used if there are multiple `OldModifier` and not all are set in the component instance

If `simplify` is specified and true then perform obvious simplifications to clean up the new modifier; otherwise leave as is. *Note: `simplify` is primarily intended for converting enumerations and emulated enumerations that naturally lead to large nested if-expressions. The simplifications may also simplify parts of the original expression.*

Behaviour in unusal cases:

1. if NewModifier list is empty then the modifier is just removed
2. If OldModifer list is empty it is added for all uses of the class
3. If OldModifier_i is cardinality(a)=0 the conversion will only be applied for a component comp if there is no inside connection to comp.a. This can be combined with other modifiers that are handled in the usual way.
4. If OldModifier_i is cardinality(a)=1 the conversion will only be applied for a component comp if there is are any inside connections to comp.a

Converting modifiers with cardinality is used to remove the deprecated operator cardinality from model libraries, and replace tests on cardinality in models by parameters explicitly enabling the different cases. [*I.e. instead of model A internally testing if its connector B is connected, there will be a parameter for enabling connector B, and the conversion ensures that each component of model A will have this parameter set accordingly*.] The case where the old class is used as a base-class, and there are any outside connections to a, and there is convertModifiers involving the cardinality of a is not handled. [*In case a parameter is simply renamed it is preferable to use convertElement, since that also handles e.g. binding equations using the parameter*.]

convertMessage("OldClass", "Failed Message");

For any use of `OldClass` (or element of `OldClass`) report that conversion could not be applied with the given message. [*This may be useful if there is no possibility to convert a specific class. An alternative is to construct ObsoloteLibraryA for problematic cases, which may be more work but allows users to directly run the models after the conversion and later convert them.*]

Chapter 19

Unit Expressions

Chapter 20

The Modelica Standard Library

Appendix A

Glossary

Appendix B

Modelica Concrete Syntax

Appendix C

Modelica DAE Representation

Appendix D

Derivation of Stream Equations