Modelica Compliance Library Guide

March 11, 2013

1 Test case structure

Each test case should consist of a separate file that contains one model marked as a test case. The test case model does not need to be the top-level class in the file, so a file can e.g. have a package with multiple classes defined in it, but each file should only contain one test case model. Models are marked as test cases by the test case annotation:

```
annotation(__ModelicaAssociation(TestCase(shouldPass=true|false)))
```

The shouldPass property defines whether the test case is expected to succeed or fail. Each test case should also extend the Icons.TestCase model which supplies a graphical icon annotation, a Documentation annotation with HTML documentation, and an experiment annotation for simulation properties. Given below is an example of a test case which is expected to succeed:

```
model SimpleDeclaration
  extends Icons.TestCase;
  Real x = 3;
  Real y = x;
  annotation(
    __ModelicaAssociation(TestCase(shouldPass=true)),
    experiment(StopTime=0.01),
    Documentation(
      info="<html>Tests simple component declarations.</html>"));
end SimpleDeclaration;
```

And a test case which is expected to fail:

```
model DoubleDeclaration
  extends Icons.TestCase;
  Real x;
  Real x "Double declaration of x.";
  annotation(
    __ModelicaAssociation(TestCase(shouldPass=false)),
    experiment(StopTime=0.01),
    Documentation(
      info="<html>Tests that double declaration of elements is not
          allowed, according to section 4.2.</html>"));
end DoubleDeclaration;
```

2 Library structure

The compliance library uses a package hierarchy to divide the test cases into suitable categories, as defined by the table below. References are for the Modelica 3.3 specification (since I haven't seen the mythical 3.2rev2). Section references in the table uses intervals and wildcards, where 1.2-1.4 means from section 1.2 to 1.4 and 1.1.* means section 1.1 and its subsections. Section references does not otherwise include subsections, so section 1.1 means *only* 1.1.

Components	
Declarations	Section 4.2-4.4.2.1, 4.4.3.
Conditional	Conditional components, section 4.4.5.
Prefixes	Component prefixes, section 4.4.2.2, 4.4.4.*.
Variability	Variability prefixes, section 3.8.*, 4.4.4.
Time	The built-in variable time, section 3.6.7.
Classes	
Declarations	
Long	"Long" declarations, section 4.5, 4.5.2, 4.5.3.
Short	"Short" declarations, section 4.5.1.
Specialized	Restrictions on specialized classes, section 4.6.
Prefixes	Class prefixes, section 4.4.2.2.
Balancing	Balance checking, section 4.7.
Predefined	The predefined types, section 4.8-4.8.4, 4.8.8.1.
Enumeration	Enumerations, section 4.8.5.
Scoping	
MemberAccess	The member access operator, section 3.6.6.
Visibility	Public and protected elements, section 4.1.
NameLookup	
Simple	Section 5.3.1.
Composite	Section 5.3.2.
Global	Section 5.3.3.
Imports	Section 13.2.1.*.
InnerOuter	Section 5.4-5.5.
Operators	
Arithmetic	Arithmetic operators $\hat{}\ * \ / + -$, section 3.4.
Relational	Relational operators ==, <>, etc. Section 3.5 and 10.6.10.
Logical	Logical operators **not**, **and**, **or** (section 3.5).
String	String concatenation, section 3.6.1.
Mathematical	Operators in 3.7.1.*, except Integer and String.
Conversion	Conversion operators in 3.7.1, Integer and String.
Events	Event-related operators in 3.7.3.*.
Special	Special purpose operators in 3.7.2.*, except connection operators.
If	If-expressions, section 3.3, 3.6.5.
Precedence	Precedence rules in section 3.2.
Associativity	Associativity rules in in section 3.2.
Overloading	Overloaded operators, chapter 14.

Inheritance	
Flattening	Flattening of extends, section 5.6.1, 7.1-7.1.2.
Restrictions	Base class restrictions, section 7.1.3-7.1.4.
Modification	
Flattening	Flattening of modifications, section 7.2-7.2.3, 7.2.5.
Restrictions	Restrictions on modifications, section 7.2.4, 7.2.6.
Redeclare	
Flattening	Flattening of redeclares, section 7.3, 7.3.4.
ConstrainingType	Constraining types, section 7.3.2.
ClassExtends	Class extends, section 7.3.1.
Restrictions	Restrictions on redeclares, 7.3.3.
Equations	
Equality	Section 8.3.1.
For	Section 8.3.2.*.
If	Section 8.3.4.
When	Section 8.3.5.*.
Reinit	Section 8.3.6.
Assert	Section 8.3.7.
Terminate	Section 8.3.8.
Events?	Section 8.5.
Initialization?	Section 8.6.
Algorithms	
Assignment	Section 11.2.1.*.
For	Section 11.2.2.*.
If	Section 11.2.6.
When	Section 11.2.7.*.
While	Section 11.2.3.
Break	Section 11.2.4.
Return	Section 12.1.2.
Reinit	Section 11.2.8.1.
Assert	Section 11.2.8.2.
Terminate	Section 11.2.8.3.
Connections	
Declarations	Basic connect equations, section 9.1.
Operators	Section 3.7.2, 15.2-15.3
Expandable	Expandable connectors, section 9.1.3.
Stream	Stream connectors, chapter 15.
Restrictions	The restrictions in section 9.3.*, 15.1.
Overconstrained	Section 9.4.

Arrays	
Declarations	Array declarations, section 4.4.2, 10.1.*, 10.7.
Flexible	Flexible arrays, section 12.4.5.
Indexing	Array indexing, slicing, section 10.5.*, 10.6.9.
Functions	
Size	ndims and size, section 10.3.1.
Construction	Array construction, section 10.3.3 and 10.4.*.
Conversion	Dimensionality conversion functions, section 10.3.2.
Reductions	Reduction expressions from section 10.3.4.*.
Algebra	Matrix and vector algebra functions, section 10.3.5.
Operations	
Equality	Array equality and assignment, section 10.6.1.
Arithmetic	Arithmetic operators, section 10.6.2-10.6.3, 10.6.5-10.6.7.
MatrixProduct	Matrix multiplication, section 10.6.4, 10.6.8.
Relational	Relational operators, section 10.6.10.
Logical	Logical operators, section 10.6.11.

Functions	
Declarations	Function declarations, section 12.1-12.1.1, 12.1.3.
Restrictions	Function restrictions, section 12.2.
Calls	Function calls, section 12.4.1, 12.4.3-12.4.4, 12.4.7.
Vectorization	Vectorization of scalar functions, section 12.4.6.
HigherOrder	Higher order functions, 12.4.2.*.
Records	Record constructor functions, section 12.6.
External	External functions, section 12.9.*.
Derivative	Function derivatives, section 12.7.*
Inverse	Function inverses, section 12.8.

Packages	Chapter 13, except imports.
Annotations	Chapter 18.

3 Style guide

The test cases should follow the style of the specification and MSL to the extent that a consistent style is used. This means that class names should be UpperCamelCase, while component names should be lowerCamelCase. The contents of classes and control statements should be indented with two spaces, and tabs should not be used.

```
package TestPackage
  type TestType = Real;

  model TestModel
    TestType testComponent;
  end TestModel;
end TestPackage;
```

Control statements should be written as in the specification with the beginning and end parts on separate lines and the content indented.

```
for i in 1:3 loop
  if i == 1 then
    x[i] = 1;
  elseif i == 2 then
    x[i] = 2;
  else
    x[i] = 3;
  end if;
end loop;
```

Expressions and operators should usually be separated with spaces. Notable exceptions are brackets and the range operator.

```
model Test
  Real x = 2.0;
  Real y, z;
  Integer u[1, 3] = {{1, 2, 3}};
equation
  y = x * (2 + z);
  z = sum(u[1, i] for i in 1:size(u, 1));
end Test;
```