

SBML Level 3 Package: Arrays ('arrays')

Leandro Watanabe

l.watanabe@utah.edu

Dept. of Elec. and Comp. Eng.
University of Utah
Salt Lake City, UT, USA

Chris J. Myers

myers@ece.utah.edu

Dept. of Elec. and Comp. Eng.
University of Utah
Salt Lake City, UT, USA

Lucian P Smith

lpsmith@uw.edu

California Institute of Technology
Seattle, WA, USA

TODO: ADD OTHER AUTHORS

Version 0.3 α (Draft)

September 24, 2015

This is a draft specification for the package 'arrays' and not a normative document. Please send feedback to the Package Working Group mailing list at sbml-arrays@lists.sourceforge.net

The latest release, past releases, and other materials related to this specification are available at
http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/Arrays

This release of the specification is available at
http://sbml.org/Documents/Specifications/arrays_Level_1_Version_1



Contents

1 Introduction and motivation	3
1.1 Proposal corresponding to this package specification	3
1.2 Package dependencies	3
1.3 Document conventions	3
2 Background	5
2.1 Problems with current SBML approaches	5
2.2 Package history	5
2.3 Array notation used in this document	5
3 Proposed syntax and semantics	6
3.1 Namespace URI and other declarations necessary for using this package	6
3.2 Primitive data types	6
3.3 Dimensions	6
3.4 Indices	8
3.5 Mathematical formulas	11
4 Examples	13
4.1 Array of initial assignments	13
4.2 Array of rate rules	14
4.3 Array of reactions	15
4.4 Array of events	17
A Validation Rules	19
B Example algorithm for producing a “flattened” model	24
Acknowledgments	26
References	27

1 Introduction and motivation

This document describes a package for arrays to be used with the Systems Biology Markup Language (SBML) Level 3 Version 1. This package enables SBML elements such as compartments, parameters, species, reactions, etc. to be organized as arrays. While this package proposal is independent of all other package proposals, it is designed to work with and complement the hierarchical model composition and distributions packages. Indeed, the need to specify probability mass functions is a significant motivator for needing arrays.

1.1 Proposal corresponding to this package specification

TODO: DO WE NEED A PROPOSAL?

1.2 Package dependencies

The arrays package has no dependencies on other SBML Level 3 packages. It is also designed with the goal of being able to work seamlessly with other SBML Level 3 packages. Namely, all objects are entirely encapsulated and all extensions to existing SBML classes are defined as optional.

1.3 Document conventions

Following the precedent set by the SBML Level 3 Core specification document, we use UML 1.0 (Unified Modeling Language; [Eriksson and Penker 1998](#); [Oestereich 1999](#)) class diagram notation to define the constructs provided by this package. We also use color in the diagrams to carry additional information for the benefit of those viewing the document on media that can display color. The following are the colors we use and what they represent:

- **Black:** Items colored black in the UML diagrams are components taken unchanged from their definition in the SBML Level 3 Core specification document.
- **Green:** Items colored green are components that exist in SBML Level 3 Core, but are extended by this package. Class boxes are also drawn with dashed lines to further distinguish them.
- **Blue:** Items colored blue are new components introduced in this package specification. They have no equivalent in the SBML Level 3 Core specification.

We also use the following typographical conventions to distinguish the names of objects and data types from other entities; these conventions are identical to the conventions used in the SBML Level 3 Core specification document:

AbstractClass: Abstract classes are classes that are never instantiated directly, but rather serve as parents of other object classes. Their names begin with a capital letter and they are printed in a slanted, bold, sans-serif typeface. In electronic document formats, the class names defined within this document are also hyperlinked to their definitions; clicking on these items will, given appropriate software, switch the view to the section in this document containing the definition of that class. (However, for classes that are unchanged from their definitions in SBML Level 3 Core, the class names are not hyperlinked because they are not defined within this document.)

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. In electronic document formats, the class names are also hyperlinked to their definitions in this specification document. (However, as in the previous case, class names are not hyperlinked if they are for classes that are unchanged from their definitions in the SBML Level 3 Core specification.)

Something, otherThing: Attributes of classes, data type names, literal XML, and generally all tokens *other* than SBML UML class names, are printed in an upright typewriter typeface. Primitive types defined by SBML begin with a capital letter; SBML also makes use of primitive types defined by XML Schema 1.0 ([Biron and Malhotra, 2000](#); [Fallside, 2000](#); [Thompson et al., 2000](#)), but unfortunately, XML Schema does not follow any

capitalization convention and primitive types drawn from the XML Schema language may or may not start with a capital letter.

For other matters involving the use of UML and XML, we follow the conventions used in the SBML Level 3 Core specification document.

2 Background

2.1 Problems with current SBML approaches

All mathematical operations in SBML are currently restricted to operations on scalar values. There is currently no way to express an array of values. If one desires such a structure, they must first flatten it into individual scalar values. This approach is not appealing for expressing things such as a probability mass function for a distribution. A second problem in SBML is that regular structures cannot be represented efficiently. While the hierarchical modeling package has made this somewhat easier, it is still necessary to instantiate each submodel individually and make connections between the submodels explicitly.

2.2 Package history

The idea of adding arrays to SBML has been around for more than 10 years when it was originally proposed by Andrew Finney, Victoria Gor, Ben Bornstein, and Eric Mjolsness in September 2003. Bruce Shapiro, Victoria Gor, and Eric Mjolsness revised this proposal in December 2004 to support “dynamic” arrays. This specification adopts several of the ideas described in these proposals with some exceptions. In particular, this specification restricts arrays to be statically sized, and it requires all arrays to be explicitly defined.

This specification evolved through discussions at various COMBINE and HARMONY workshops beginning with a presentation by Chris Myers at COMBINE 2012 at the University of Toronto. Over the past several years, Sarah Keating, Lucian Smith, Mike Hucka, Nicolas LeNovere, and Stuart Moodie, among others, made significant contributions to these discussions. As part of the 2014 Google Summer of Code, Leandro Watanabe under the direction of Nicolas Rodriguez implemented a JSBML version of the arrays package, and during this time made significant revisions to this specification document. Around the same time, Sarah Keating implemented this package within libSBML, as well. Support for the arrays package has been incorporated into the iBioSim software tool with the help of Leandro Watanabe, Scott Glass, and Chris Myers.

TODO: IS THERE ANYTHING MORE THAT SHOULD BE SAID HERE? ANYONE LEFT OUT?

2.3 Array notation used in this document

Elements of 1-dimensional arrays are referenced as $X[d0]$ in this document to refer to the $d0^{th}$ element of the 1-dimensional array X while $X[d1][d0]$ is used to refer to an individual element in a 2-dimensional array where $d1$ selects the highest dimension and $d0$ the lowest dimension. Higher dimension arrays can be indexed in a similar fashion. This notation is used to describe the meaning of certain features, but it is not intended to be used explicitly anywhere in SBML. In some cases, this specification may refer to the i^{th} element of an array of objects (such as rules) that do not have object ids and cannot be referenced in SBML.

3 Proposed syntax and semantics

In this section, we define the syntax and semantics of the Arrays package for SBML Level 3 Version 1. We describe the various data types and constructs defined in this package, then in [Section 4 on page 13](#), we provide complete examples of using the constructs in example SBML models.

3.1 Namespace URI and other declarations necessary for using this package

Every SBML Level 3 package is identified uniquely by an XML namespace URI. For an SBML document to be able to use a given SBML Level 3 package, it must declare the use of that package by referencing its URI. The following is the namespace URI for this version of the Arrays package for SBML Level 3 Version 1:

```
"http://www.sbml.org/sbml/level3/version1/arrays/version1"
```

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model, or whether the package is optional. This is done using the attribute **required** on the `<sbml>` element in the SBML document. For the Arrays package, the value of this attribute must be set to **"true"**. The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 1 and this version of the Arrays package:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
  xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1" arrays:required="true">
```

3.2 Primitive data types

Section 3.1 of the SBML Level 3 Version 1 Core specification defines a number of primitive data types and also uses a number of XML Schema 1.0 data types ([Biron and Malhotra, 2000](#)). More specifically, we make use of **Sid**, **string**, **int**, and **SidRef**.

3.3 Dimensions

As shown in [Figure 1](#), the arrays package extends the **SBase** class from core SBML with the addition of a list of **Dimension** objects. An object with a list of dimensions is an array, and the number of dimensions is equivalent to the number of **Dimension** objects in the list.

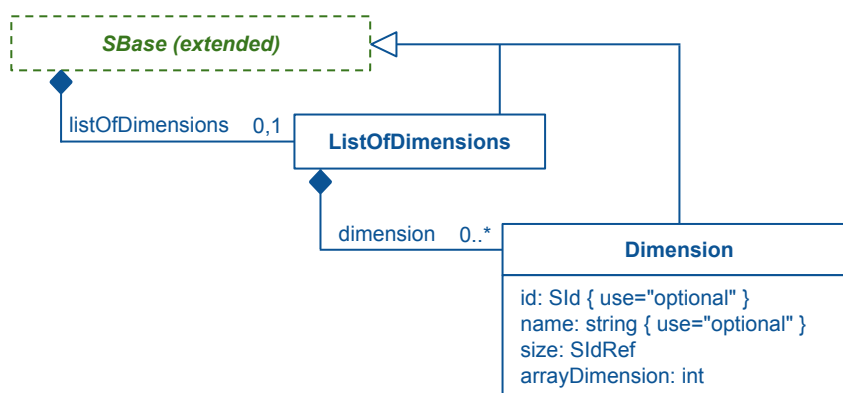


Figure 1: A UML representation of the extended **SBase**, **ListOfDimensions**, and **Dimension** classes. See [Section 1.3](#) for conventions related to this figure.

The id attribute

The **Dimension** object has an optional attribute, **id**, used to give the dimension an identifier. The value of **id** must conform to the syntax permitted by the **SId** data type described in Section 3.1.7 of the SBML Level 3 Version 1 Core specification. The scope of the **id** attribute is local to the enclosing object definition and is not visible outside the object definition. It may therefore be used in mathematical expressions within the enclosing object definition, but only by reference—it may not be assigned a value by a **Rule** or other assignment. When used, it is similar to the time variable from SBML core, or the **CoordinateComponent** element from the Spatial specification: it represents a 'domain' over which the model is defined. In context, then, it represents the set of all integers from 0 to the **Dimension**'s size.


The value of the **id** attribute must be unique among the **SId** values both local to the object and global to the **Model**. This means that if a **Parameter** in the same **Model** has an **id** of "a0", and the same parent object has another **Dimension** with an **id** of "b0", this **Dimension** must not have a value of either "a0" or "b0". Similarly, if a parent **Reaction** of a **Dimension** has a **LocalParameter** child of its **KineticLaw** with an **id** of "c0", that **Dimension** may not be given an **id** of "c0". However, two different **AssignmentRule** elements in the same **Model** may each be given a **Dimension** with an **id** of "d0", without semantic overlap. In general, the **id** attribute must be used as described in Section 3.3 of the SBML Level 3 Version 1 Core specification.

The name attribute

Dimension also has an optional **name** attribute, of type **string** (see Section 3.1.1 and Section 3.3 of the SBML Level 3 Version 1 Core specification).

The size attribute

Each dimension of an array has a fixed size which is set with the **size** attribute. The size attribute is of type **SIdRef** (see Section 3.1.8 of the SBML Level 3 Version 1 Core specification) and must refer to a **Parameter** object instance defined in the model. The **Parameter** referenced must be **constant**, scalar (that is, it may not itself have a **Dimension** child), and have a defined initial value that must be a positive non-zero integer.

 **Lucian:** Check to see if 'non-zero' is a requirement. It also might still be worth having a discussion about the 'defined initial value'.

The arrayDimension attribute

The **arrayDimension** attribute specifies which dimension is defined by this **Dimension** object. The only allowed values for **arrayDimension** are non-negative integer values (e.g. 0, 1, 2, ..., n). Also, there can be at most one **Dimension** for each possible value of **arrayDimension**. Finally, if a list of dimensions includes a **Dimension** with **arrayDimension** value of n , it must also include **Dimension** objects with value $n - 1, n - 2, \dots, 1, 0$.

Example

As an example, a 10×10 array of compartments "Cell" could be defined as:

```
<listOfCompartments>
  <compartment id="Cell" constant="true" spatialDimensions="3" size="1">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
      <arrays:dimension arrays:id="d1" arrays:size="n" arrays:arrayDimension="1"/>
    </arrays:listOfDimensions>
  </compartment>
</listOfCompartments>
<listOfParameters>
  <parameter id="n" constant="true" value="10"/>
</listOfParameters>
```

Dimension usage

In SBML Level 3 Core, the following objects are permitted to have a **ListOfDimensions**:

- | | | |
|---------------------------|-----------------------------|----------------------------|
| ■ Compartment, | ■ InitialAssignment, | ■ Species, and |
| ■ Constraint, | ■ Parameter, | ■ SpeciesReference. |
| ■ Event, | ■ Reaction, | |
| ■ EventAssignment, | ■ Rule, | |

All other SBML Level 3 Core objects are not permitted to have a **ListOfDimensions**:

- | | | |
|---------------------------------|------------------------------------|--------------------------|
| ■ Delay, | ■ ListOfParameters, | ■ LocalParameter, |
| ■ FunctionDefinition, | ■ ListOfInitialAssignments, | ■ Model, |
| ■ KineticLaw, | ■ ListOfRules, | ■ Priority, |
| ■ ListOfUnits, | ■ ListOfConstraints, | ■ Trigger, |
| ■ ListOfUnitDefinitions, | ■ ListOfReactions, | ■ Unit, and |
| ■ ListOfLocalParameters, | ■ ListOfSpeciesReferences, | ■ UnitDefinition. |
| ■ ListOfCompartments, | ■ ListOfEvents, | |
| ■ ListOfSpecies, | ■ ListOfEventAssignments, | |

All SBML objects defined by packages that inherit from **SBase** are permitted to have a **ListOfDimensions** unless it is explicitly disallowed in the corresponding package specification, with the exception of any **ListOf_____** class. None of these arrays package classes (**Dimension**, **Index**, **ListOfDimensions**, and **ListOfIndices**) are permitted to have a **ListOfDimensions**.

It is important to note that it is assumed that all elements of arrays created in this way share the same attribute values. For example, if a **Species** is set **boundaryCondition**="true", all **Species** in the array are then defined as boundary species. Similarly, every element in the array will have the same units. If a size is specified for a **Compartment**, or a value for a **Parameter**, then every element of the array takes that initial value. To specify different initial values to different elements of an array, an **InitialAssignment** can be used, but generally, there is no way to alter the values of individual attributes of elements in an array.

3.4 Indices

As shown in Figure 2, the arrays package extends the **SBase** class from core SBML with the addition of a list of indices. Each index in this list is used to specify an array index. The array index is used to reference an arrayed object specified in an attribute for this SBML element. Each arrayed object specified in an attribute can have at most one index for each dimension specified for that object.

Note that **SBase** objects containing **SIRef** objects with a **ListOfDimensions** are required to have a **ListOfIndices**, where the number of **Index** objects for each particular object you are indexing is required to match the number of dimensions of the referenced arrayed object. For example, assume there is an **AssignmentRule** that is assigning a value to **Parameter**. Furthermore, assume that the **Parameter** is a two-dimensional array. Then, it must be the case that the **Rule** has two **Index** objects to reference a single value of the **Parameter**.

The referencedAttribute attribute

The **referencedAttribute** attribute specifies the attribute of the SBML object to which this index is referencing.

The arrayDimension attribute

The **arrayDimension** attribute specifies which dimension this index corresponds to. The only allowed values for **arrayDimension** are non-negative integer values. Also, there can be at most one **Index** for each possible pair of val-

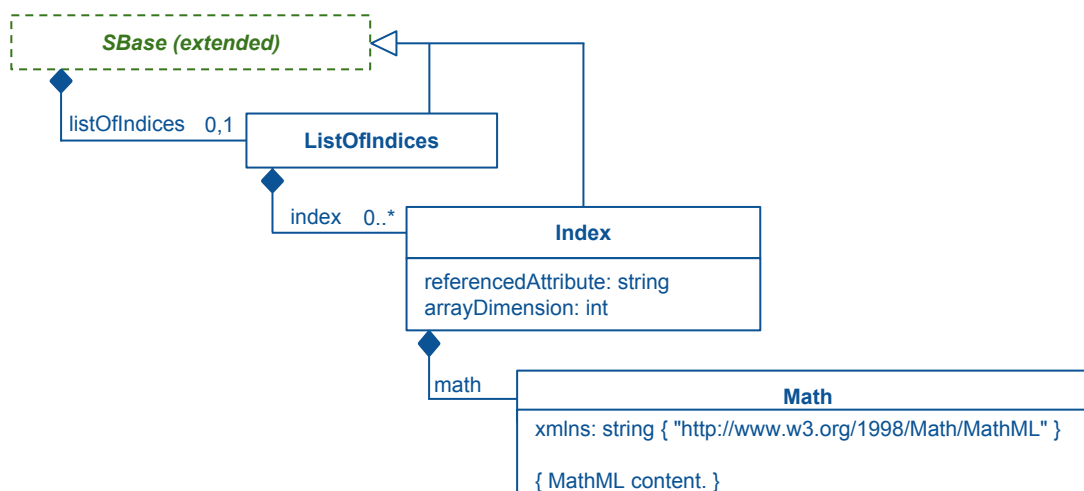


Figure 2: A UML representation of the extended **SBase**, **ListOfIndices**, and **Index** classes. See [Section 1.3](#) for conventions related to this figure.

ues **arrayDimension** and **referencedAttribute**. Finally, if a list of indices includes a **Index** with **arrayDimension** value of n , it must also include **Index** objects with value $n - 1, n - 2, \dots, 1, 0$ for each **referencedAttribute** attribute.

The **math** element

The **math** element provides a mathematical expression which is evaluated to determine the index value for the reference. Note that arrays are 0-based which means that an object of size n can have index values between 0 and $n - 1$. When an index value computes to a value outside this range during a **simulation or analysis**, this is an **array out-of-bounds** error and **the program** should terminate and report this error to the user.

Example

The example below copies a reverse copy of the array in parameter **x** into parameter **y**.

```

<listOfParameters>
  <parameter id="n" constant="true" value="10"/>
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
  <parameter id="Y" constant="false" value="0">
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
</listOfParameters>
<listOfRules>
  <assignmentRule metaid="rule0" variable="Y">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <selector/>
        <ci> X </ci>
        <ci> d0 </ci>
      </apply>
    </math>
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">

```

```

<arrays:index arrays:referencedAttribute="variable" arrays:arrayDimension="0">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <minus/>
      <apply>
        <minus/>
        <ci> n </ci>
        <cn type="integer"> 1 </cn>
      </apply>
      <ci> d0 </ci>
    </apply>
  </math>
</arrays:index>
</arrays:listOfIndices>
</assignmentRule>
</listOfRules>


```

Basically, this example is equivalent to:

```

n = 10;
for (d0=0; d0 < n; d0++) {
  Y[(n-1) - d0] = X[d0];
}

```

 **Lucian:** I removed the IDs of X and Y's Dimension because they made this example more confusing than it needed to be.

Index usage

In SBML core, only objects that have defined attributes of **SIIDRef** type can have a list of indices, since that's the only type of ID reference used, since units are disallowed from being arrays. For SBML Level 3 Core, the following objects can have a list of indices:

- **Model** for its **ConversionFactor** attribute,
- **Species** for its **Compartment** and **ConversionFactor** attributes,
- **Reaction** for its **Compartment** attribute,
- **InitialAssignment** for its **Symbol** attribute,
- **Rule** for its **Variable** attribute,
- **SpeciesReference** for its **Species** attribute, and
- **EventAssignment** for its **Variable** attribute.

Similarly, for packages, any objects that have defined attributes of **SIIDRef** type can also have indices. In addition, elements that define new ID types on elements that can be arrayed, and then provide references to those arrays (such as the **PortID** and **PortIdRef** types from the Hierarchical Model Composition package), can also have **Index** children. In all cases, the object referenced by an index must, of course, be an arrayed object (i.e., have a specified dimension), and it must include the dimension being indexed. Finally, the **math** element must be statically computable. In other words, any identifier that appears in the **math**, other than a **Dimension** id for this object, must be a constant. For each possible value of each **Dimension** id (i.e., 0 to size-1 of the **Dimension** referred to) that appears in the **math** element, there should be no array out-of-bounds problems. Namely, it must evaluate to a non-negative integer that is less than the size of the corresponding **Dimension** for the object being indexed.

Consider again the example above. The **Index** refers to the variable in the **AssignmentRule** which is indeed an array. Since it has **arrayDimension** 0, the index for **arrayDimension** 0 is valid. The **math** element is $(n - 1) - d0$ which is statically computable because n is a constant **Parameter** while $d0$ is the **Dimension** id for the **AssignmentRule**. Since n is 10 and $d0$ can take values of 0 to 9, this **math** formula evaluates to 9 down to 0. These values are always non-negative and less than the size of the 0 dimension for Y. Therefore, there are no array out-of-bounds problems.

3.5 Mathematical formulas



Lucian: This bit will have to change after we finally get the 'MathML packages' up and running, but we might as well leave it for now.

Section 3.4 of the SBML Level 3 Version 1 Core specification defines how mathematical expressions are expressed in SBML using a subset of MathML 2.0 (W3C, 2000). The arrays package extends the supported MathML subset to include **vector** and **selector**. The first argument of a MathML **selector** must be a MathML **vector** object or a valid identifier to an **SBase** object extended with a list of **Dimension** objects. The number of dimensions that the first argument has must be equal to the number of arguments of the **selector** minus one. The remaining arguments of a MathML **selector** must evaluate to scalars that are non-negative integers, and it must be statically computable. In other words, any identifier that appears in the argument, other than a **Dimension id** for this object, must be a constant. Also, for each possible value of each **Dimension id** (i.e., 0 to size-1 of the **Dimension** referred to) that appears in the second and later arguments of the **selector**, there should be no array out-of-bounds problems. Namely, it must evaluate to a non-negative integer that is less than the size of the corresponding **Dimension** for the object being indexed where the last argument refers to dimension 0, next to last to dimension 1, etc.

An example that uses both **vector** and **selector** in an **InitialAssignment** is shown below. In this example, X is an array of size 3. Therefore, the **InitialAssignment** for X is also size 3 which effectively creates three initial assignments to assign each value of the array. The assignment is $\{3, 2, 1\}[d_0]$ where the vector $\{3, 2, 1\}$ is the first argument to the selector, and the **InitialAssignment** dimension id, d_0 , is the second argument to the selector. This assignment sets the initial values as follows: $X[0] = 3$, $X[1] = 2$, and $X[2] = 1$.

```
<listOfParameters>
  <parameter id="n" constant="true" value="3"/>
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="X">
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="symbol" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <selector/>
        <vector>
          <cn type="integer"> 3 </cn>
          <cn type="integer"> 2 </cn>
          <cn type="integer"> 1 </cn>
        </vector>
        <ci> d0 </ci>
      </apply>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
```

Note that it is invalid for any SBase object to have a value of type **vector**. Every mathematical operation must result in a scalar value. Although operations on **vector** nodes could be allowed, they are not needed since **vector** objects must be reduced to scalars. In fact, it is more efficient to work with operations on scalars rather than vectors. For

instance, $\{\{1,2\},\{4,5\}\} + \{\{5,6\},\{7,8\}\}[0][1]$ is equivalent to $\{\{1,2\},\{4,5\}\}[0][1] + \{\{5,6\},\{7,8\}\}[0][1]$, so only the later one without vector math is considered to be valid.



Lucian: I don't know what you mean here by 'invalid to have a value of type vector'. In fact, it seems to me that things could be a ton simpler if you allowed simple vector assignment, a la:

```
<listOfParameters>
  <parameter id="n" constant="true" value="3"/>
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="X">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <vector>
        <cn type="integer"> 3 </cn>
        <cn type="integer"> 2 </cn>
        <cn type="integer"> 1 </cn>
      </vector>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
```



Lucian: Is there a reason you don't want this? This (or something similar) will also be necessary in the case of distrib, where it needs vector input and output.

4 Examples

This section contains a variety of examples of SBML Level 3 Version 1 documents employing the Arrays package.

4.1 Array of initial assignments

This example uses an **InitialAssignment** array to give an initial value to an array *X* of size 10, in which half of the elements are given value 5.7 and the other half value 3.2.

```
<listOfParameters>
  <!-- Create an array X of size n -->
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
        <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
      </arrays:listOfDimensions>
    </parameter>
    <!-- Set size n=10 -->
    <parameter id="n" constant="true" value="10"/>
  </listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="X">
    <math
      xmlns="http://www.w3.org/1998/Math/MathML">
      <piecewise>
        <piece>
          <cn> 5.7 </cn>
          <apply>
            <lt/>
            <ci> d0 </ci>
            <cn type="integer"> 5 </cn>
          </apply>
        </piece>
        <otherwise>
          <cn> 3.2 </cn>
        </otherwise>
      </piecewise>
    </math>
    <arrays:listOfDimensions>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
        <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
      </arrays:listOfDimensions>
    <arrays:listOfIndices>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
        <arrays:index arrays:referencedAttribute="symbol" arrays:arrayDimension="0">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> d0 </ci>
          </math>
        </arrays:index>
      </arrays:listOfIndices>
    </initialAssignment>
  </listOfInitialAssignments>
```

This could also be done using the vector and selector operators.

```
<listOfParameters>
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
        <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
      </arrays:listOfDimensions>
```

```

</parameter>
<parameter id="n" constant="true" value="10"/>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="X">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <selector/>
        <vector>
          <cn> 5.7 </cn>
          <cn> 5.7 </cn>
          <cn> 5.7 </cn>
          <cn> 5.7 </cn>
          <cn> 5.7 </cn>
          <cn> 5.7 </cn>
          <cn> 3.2 </cn>
          <cn> 3.2 </cn>
          <cn> 3.2 </cn>
          <cn> 3.2 </cn>
          <cn> 3.2 </cn>
        </vector>
        <ci> d0 </ci>
      </apply>
    </math>
    <arrays:listOfDimensions>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="symbol" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
  </initialAssignment>
</listOfInitialAssignments>

```

4.2 Array of rate rules

This example creates a **RateRule** array to compute the following:

$$\frac{dX[d0]}{dt} = \begin{cases} y, & i = 0, 1, 2, 3, 4 \\ 2y, & i = 5, 6, 7 \end{cases}$$

```

<listOfParameters>
  <!-- Create size variables for arrays -->
  <parameter id="n" constant="true" value="8"/>
  <!-- Create array X of size n -->
  <parameter id="X" constant="false" value="0">
    <arrays:listOfDimensions>
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
  <!-- Create scalar parameter y -->
  <parameter id="y" constant="false" value="0"/>
</listOfParameters>
<listOfRules>
  <rateRule metaid="rule" variable="X">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <piecewise>

```

```

        <piece>
          <ci> y </ci>
          <apply>
            <lt/>
            <ci> d0 </ci>
            <cn type="integer"> 5 </cn>
          </apply>
        </piece>
        <otherwise>
          <apply>
            <times/>
            <cn type="integer"> 2 </cn>
            <ci> y </ci>
          </apply>
        </otherwise>
      </piecewise>
    </math>
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="variable" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
  </rateRule>
</listOfRules>

```

4.3 Array of reactions

This example creates an array Cell of 100 compartments, arrays for species A, B, and C also of size 100 with each one placed in the corresponding compartment cell[i], and an array of 100 reactions with one within each Cell[d0] converting A[d0] plus B[d0] into C[d0] with kinetic rate law of $k \cdot A[d0] \cdot B[d0]$.

```

<listOfCompartments>
  <!-- Create an array of n compartments -->
  <compartment id="Cell" constant="true" spatialDimensions="3" size="1">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </compartment>
</listOfCompartments>
<listOfSpecies>
  <!-- Create array of n species A with A[d0] placed in Cell[d0] -->
  <species id="B" constant="false" initialAmount="0" hasOnlySubstanceUnits="true"
    boundaryCondition="false" compartment="Cell">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="compartment" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
  </species>

```

```

<!-- Create array of n species B with B[d0] placed in Cell[d0] -->
<species id="B" constant="false" initialAmount="0" hasOnlySubstanceUnits="true"
boundaryCondition="false" compartment="Cell">
  <arrays:listOfDimensions
    xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
    <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
  </arrays:listOfDimensions>
  <arrays:listOfIndices
    xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
    <arrays:index arrays:referencedAttribute="compartment" arrays:arrayDimension="0">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> d0 </ci>
      </math>
    </arrays:index>
  </arrays:listOfIndices>
</species>
<!-- Create array of n species C with C[d0] placed in cell[d0] -->
<species id="C" constant="false" initialAmount="0" hasOnlySubstanceUnits="true"
boundaryCondition="false" compartment="Cell">
  <arrays:listOfDimensions
    xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
    <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
  </arrays:listOfDimensions>
  <arrays:listOfIndices
    xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
    <arrays:index arrays:referencedAttribute="compartment" arrays:arrayDimension="0">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> d0 </ci>
      </math>
    </arrays:index>
  </arrays:listOfIndices>
</species>
</listOfSpecies>
<listOfParameters>
  <parameter id="k" constant="false" value="1"/>
  <!-- Specifies size of all arrays (i.e., n:=100) -->
  <parameter id="n" constant="true" value="100"/>
</listOfParameters>
<listOfReactions>
  <!-- Create array of n reactions r with r[d0] converting A[d0] and B[d0] into C[d0]-->
  <reaction id="r" reversible="false" fast="false" compartment="Cell">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <arrays:listOfIndices
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="species" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
    <listOfReactants>
      <speciesReference constant="true" species="B" stoichiometry="1">
        <arrays:listOfIndices
          xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
          <arrays:index arrays:referencedAttribute="species" arrays:arrayDimension="0">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
              <ci> d0 </ci>
            </math>
          </arrays:index>
        </arrays:listOfIndices>
      </speciesReference>
      <speciesReference constant="true" species="A" stoichiometry="1">
        <arrays:listOfIndices

```



```

    xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
    <arrays:index arrays:referencedAttribute="species" arrays:arrayDimension="0">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> d0 </ci>
      </math>
    </arrays:index>
  </arrays:listOfIndices>
</speciesReference>
</listOfReactants>
<listOfProducts>
  <speciesReference constant="true" species="C" stoichiometry="1">
    <arrays:listOfIndices
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:index arrays:referencedAttribute="species" arrays:arrayDimension="0">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> d0 </ci>
        </math>
      </arrays:index>
    </arrays:listOfIndices>
  </speciesReference>
</listOfProducts>
<kineticLaw>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <times/>
      <apply>
        <times/>
        <ci> k </ci>
        <apply>
          <selector/>
          <ci> A </ci>
          <ci> d0 </ci>
        </apply>
      </apply>
    </math>
  </kineticLaw>
</reaction>
</listOfReactions>

```

4.4 Array of events

This example creates an **Event** array with the following behavior:

$$\text{If } X[d0] > 1 \text{ then set } X[d0] = \begin{cases} 0.5, & i = 0, 1, 2, 3, 4 \\ 0.75, & i = 6, 7, 8 \end{cases}$$

```

<listOfParameters>
  <!-- Create size variables for arrays -->
  <parameter id="n" constant="true" metaid="iBioSim2" value="8"/>
  <!-- Create array x of size n -->
  <parameter id="X" constant="false" metaid="iBioSim1" value="0">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
  </parameter>
</listOfParameters>

```

```

<listOfEvents>
  <event id="event0" useValuesFromTriggerTime="false">
    <arrays:listOfDimensions
      xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
      <arrays:dimension arrays:id="d0" arrays:size="n" arrays:arrayDimension="0"/>
    </arrays:listOfDimensions>
    <trigger persistent="false" initialValue="false">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <gt/>
          <apply>
            <selector/>
            <ci> X </ci>
            <ci> d0 </ci>
          </apply>
          <cn type="integer"> 1 </cn>
        </apply>
      </math>
    </trigger>
    <listOfEventAssignments>
      <eventAssignment variable="X">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <piecewise>
            <piece>
              <cn> 0.5 </cn>
              <apply>
                <lt/>
                <ci> d0 </ci>
                <cn type="integer"> 5 </cn>
              </apply>
            </piece>
            <otherwise>
              <cn> 0.75 </cn>
            </otherwise>
          </piecewise>
        </math>
        <arrays:listOfIndices
          xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1">
          <arrays:index arrays:referencedAttribute="variable" arrays:arrayDimension="0">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
              <ci> d0 </ci>
            </math>
          </arrays:index>
        </arrays:listOfIndices>
      </eventAssignment>
    </listOfEventAssignments>
  </event>
</listOfEvents>

```

A Validation Rules

This section summarizes all the conditions that must (or in some cases, at least *should*) be true of an SBML Level 3 Version 1 model that uses the Arrays package. We use the same conventions as are used in the SBML Level 3 Version 1 Core specification document. In particular, there are different degrees of rule strictness. Formally, the differences are expressed in the statement of a rule: either a rule states that a condition *must* be true, or a rule states that it *should* be true. Rules of the former kind are strict SBML validation rules—a model encoded in SBML must conform to all of them in order to be considered valid. Rules of the latter kind are consistency rules. To help highlight these differences, we use the following three symbols next to the rule numbers:

- ☑ A checked box indicates a *requirement* for SBML conformance. If a model does not follow this rule, it does not conform to the Arrays specification. (Mnemonic intention behind the choice of symbol: “This must be checked.”)
- ▲ A triangle indicates a *recommendation* for model consistency. If a model does not follow this rule, it is not considered strictly invalid as far as the Arrays specification is concerned; however, it indicates that the model contains a physical or conceptual inconsistency. (Mnemonic intention behind the choice of symbol: “This is a cause for warning.”)
- ★ A star indicates a strong recommendation for good modeling practice. This rule is not strictly a matter of SBML encoding, but the recommendation comes from logical reasoning. As in the previous case, if a model does not follow this rule, it is not considered an invalid SBML encoding. (Mnemonic intention behind the choice of symbol: “You’re a star if you heed this.”)

The validation rules listed in the following subsections are all stated or implied in the rest of this specification document. They are enumerated here for convenience. Unless explicitly stated, all validation rules concern objects and attributes specifically defined in the Arrays package.

For convenience and brevity, we use the shorthand “**arrays:x**” to stand for an attribute or element name **x** in the namespace for the Arrays package, using the namespace prefix **arrays**. In reality, the prefix string may be different from the literal “**arrays**” used here (and indeed, it can be any valid XML namespace prefix that the modeler or software chooses). We use “**arrays:x**” because it is shorter than to write a full explanation everywhere we refer to an attribute or element in the Arrays package namespace.

General rules about the Arrays package

- arrays-10101** ☑ To conform to Version 1 of the Arrays package specification for SBML Level 3, an SBML document must declare the use of the following XML Namespace:
“<http://www.sbml.org/sbml/level3/version1/arrays/version1>”. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.1 on page 6.](#))
- arrays-10102** ☑ Wherever they appear in an SBML document, elements and attributes from the Arrays package must be declared either implicitly or explicitly to be in the XML namespace
“<http://www.sbml.org/sbml/level3/version1/arrays/version1>”. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.1 on page 6.](#))

Rules for the extended SBML class

- arrays-10201** ☑ In all SBML documents using the Arrays package, the **SBML** object must include a value for the attribute **arrays:required** attribute. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)
- arrays-10202** ☑ The value of attribute **arrays:required** on the **SBML** object must be of the data type **boolean**. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)
- arrays-10203** ☑ The value of attribute **arrays:required** on the **SBML** object must be set to “**true**” (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.1 on page 6.](#))

General rules about MathML content in the Arrays Package

- arrays-10204** ✓ Wherever MathML content appears in an SBML document, the MathML content must be placed within a math element, and that math element must be either explicitly or implicitly declared to be in the XML namespace “<http://www.w3.org/1998/Math/MathML>”. (Reference: SBML Level 3 Version 1 Core, Section 3.4.)
- arrays-10205** ✓ The following is a list of the additional MathML 2.0 elements permitted in the Arrays package: **vector** and **selector**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10206** ✓ The first argument of a MathML **selector** must be a MathML **vector** object or a valid identifier to an **SBase** object extended with a list of **Dimension** objects. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10207** ✓ The first argument of a MathML **selector** must have a number of dimensions equal to the number of arguments to the **selector** minus 1. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10208** ✓ The arguments of a MathML **selector** other than the first argument must be statically computable. In other words, any identifier that appears in an argument, other than a **Dimension id** for the corresponding object, must be a constant. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10209** ✓ The arguments of a MathML **selector** other than the first argument must be evaluated to a scalar value. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10210** ✓ For each possible value of each **Dimension id** (i.e., 0 to size-1 of the **Dimension** referred to) that appears in the second and later arguments of the **selector**, there should be no array out-of-bounds problems. Namely, it must evaluate to a non-negative integer that is less than the size of the corresponding **Dimension** for the object being indexed where the last argument refers to dimension 0, next to last to dimension 1, etc. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10211** ✓ All mathematical operations must be performed on scalar values rather than vectors. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10212** ✓ For MathML operations with two or more operands involving MathML **vectors** or **SBase** objects with a list of **Dimension** objects, the number of dimensions and their size must agree for all operands unless the operand is a scalar type (i.e., it does not have a list of **Dimension** objects). (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))
- arrays-10213** ✓ No **SBase** is allowed to have value of type **vector**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.5 on page 11.](#))

Rules for the extended SBase abstract class

- arrays-20101** ✓ Any object derived from the extended **SBase** class (defined in the Arrays package) may contain at most one instance of a **ListOfDimensions**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))
- arrays-20102** ✓ Apart from the general notes and annotation permitted on all SBML objects, a **ListOfDimensions** container object may only contain **Dimension** objects. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))

- arrays-20103 ✓ The **ListOfDimensions** associated with an **SBase** object must have a **Dimension** object with **arrays:arrayDimension** attribute set to $0, 1, \dots, n-1$ before adding a **Dimension** object with **arrays:arrayDimension** attribute set to n . (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))
- arrays-20104 ✓ The **ListOfDimensions** associated with an **SBase** object must not have multiple **Dimension** objects with the same **arrays:arrayDimension** attribute. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))
- arrays-20105 ✓ A **ListOfDimensions** object may have the optional SBML core attributes **metaid** and **sboTerm**. No other attributes from the SBML Level 3 Core namespace or the Arrays namespace are permitted on a **ListOfDimensions** object. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))
- arrays-20106 ✓ **ListOf** objects are not permitted to have a **ListOfDimensions**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 8.](#))
- arrays-20107 ✓ In SBML Level 3 Core, **Models**, **FunctionDefinitions**, **Units**, **UnitDefinitions**, **KineticLaws**, **LocalParameters**, **Triggers**, **Priorities**, and **Delays** are not permitted to have a **ListOfDimensions**. All other SBML Level 3 Core objects are permitted to have a **ListOfDimensions** including: **Compartments**, **Species**, **Parameters**, **Initial assignments**, **Rules**, **Constraints**, **Reactions**, **Species references**, **Events**, and **Event assignments**. All SBML objects defined by packages that inherit from **SBase** are permitted to have a **ListOfDimensions** unless it is explicitly disallowed in the corresponding package specification. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 8.](#))
- arrays-20108 ✓ The **Dimension** and **Index** objects are not permitted to have a **ListOfDimensions**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 8.](#))
- arrays-20109 ✓ Any object derived from the extended **SBase** class (defined in the Arrays package) may contain at most one instance of a **ListOfIndices**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))
- arrays-20110 ✓ Apart from the general notes and annotation subobjects permitted on all SBML objects, a **ListOfIndices** container object may only contain **Index** objects. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))
- arrays-20111 ✓ The **ListOfIndices** associated with an **SBase** object must have a **Index** object with **arrays:arrayDimension** attribute set to $0, 1, \dots, n-1$ before adding a **Index** object with **arrays:arrayDimension** attribute set to n for every **arrays:referencedAttribute** that are being indexed. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))
- arrays-20112 ✓ The **ListOfIndices** in an **SBase** object must not have multiple **Index** objects with the same pair of values **arrays:arrayDimension** and **arrays:referencedAttribute** attributes. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))
- arrays-20113 ✓ A **ListOfIndices** object may have the optional SBML core attributes **metaid** and **sboTerm**. No other attributes from the SBML Level 3 Core namespace or the Arrays namespace are permitted on a **ListOfIndices** object. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))
- arrays-20114 ✓ Only SBML objects that include defined attributes of type **SIdRef** are permitted to have a **ListOfIndices**. For SBML Level 3 Core, this includes **Model** to reference a **conversionFactor** element, **Species** to reference a **compartment** or a **conversionFactor** element, **Reactions**

to reference a **compartment**, **Initial assignments** to reference a **symbol**, **Rules** to reference a **variable**, **Species references** to reference a **species**, and **Events assignments** to reference a **variable**. In addition to these, any SBML object in a package with a defined attribute of type **SIidRef** may also have a **ListOfIndices**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

arrays-20115 ✓ **SBase** objects containing **SIidRef** must have a **ListOfIndices** if the referenced **SBase** is an array. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

arrays-20116 ✓ **SBase** objects containing **SIidRef** with a **ListOfDimensions** should have a **ListOfIndices** containing as many **Index** objects for this particular **arrays:referencedAttribute** as the number of **Dimension** objects the referenced object contains. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

Rules for Dimension objects

arrays-20201 ✓ A **Dimension** object may have the optional SBML Level 3 Core attributes **metaid** and **sboTerm**. No other attributes from the SBML Level 3 Core namespace are permitted on a **Dimension** object. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

arrays-20202 ✓ A **Dimension** object must have a value for the attributes **arrays:arrayDimension** and **arrays:size**, and may additionally have the attributes **arrays:id** and **arrays:name**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))

arrays-20203 ✓ The value of the **arrays:arrayDimension** attribute on a given **Dimension** object, must be a non-negative integer value. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))

arrays-20204 ✓ The value of the **arrays:size** attribute on a given **Dimension** object, must be a valid **SIidRef** to an object of type **Parameter**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))

arrays-20205 ✓ The value of the **Parameter** referenced by the **arrays:size** attribute must be a non-negative scalar constant integer. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.3 on page 6.](#))

Rules for Index objects

arrays-20301 ✓ An **Index** object may have the optional SBML Level 3 Core attributes **metaid** and **sboTerm**. No other attributes from the SBML Level 3 Core namespace are permitted on a **Index** object. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

arrays-20302 ✓ An **Index** object must have a value for the attributes **arrays:arrayDimension**, and **arrays:referencedAttribute**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

arrays-20303 ✓ The value of the **arrays:referencedAttribute** attribute on a given **Index** object, must be an existing attribute of type **SIidRef** with a value that references a valid **SIid**. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

arrays-20304 ✓ The value of the **arrays:arrayDimension** attribute on a given **Index** object, must be a non-negative integer value. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, [Section 3.4 on page 8.](#))

arrays-20305 ✓ The object referenced by the **SIidRef** indicated by the **arrays:referencedAttribute** attribute must include an **arrays:arrayDimension** matching the **arrays:arrayDimension** for

	the Index . (Reference: SBML Level 3 Package Specification for Arrays, Version 1, Section 3.4 on page 8.)	1
		2
arrays-20306 ✓	An Index object must have exactly one MathML math element. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, Section 3.4 on page 8.)	3
		4
arrays-20307 ✓	The MathML math element in an Index object must be statically computable. In other words, any identifier that appears in the math element, other than a Dimension id for the object with this Index , must be a constant. (Reference: SBML Level 3 Package Specification for Arrays, Version 1, Section 3.4 on page 8.)	5
		6
		7
		8
arrays-20308 ✓	For each possible value of each Dimension id (i.e., 0 to size-1 of the Dimension referred to) that appears in the MathML math element, there should be no array out-of-bounds problems. Namely, it must evaluate to a non-negative integer that is less than the size of the corresponding Dimension for the object being indexed.	9
		10
		11
		12

B Example algorithm for producing a “flattened” model

The algorithms below provide a method for interpreting the Arrays constructs and creating a “flattened” SBML Level 3 Version 1 Core document. Algorithm 1 considers each **SBase** object in a SBMLDocument. First, it removes the **SBase** object from the document. Then, it calls Algorithm 2 recursively to expand each dimension. If the *currentDimension* is greater than or equal to 0, Algorithm 2 creates a clone of the object for each member of the array within the current dimension in which the id is given a unique value based on its position in that array dimension. It also replaces the current dimension id with this value, as well. The metaid of the SBase is also updated and given a unique value. Finally, it recursively calls Algorithm 2 to expand the next dimension. Once all dimensions have been expanded, this algorithm evaluates the indices. It considers each attribute being indexed in turn. For each index dimension in reverse order, it updates the id being referenced by that attribute. After expanding the indices, it adds the final flattened object to the document. Finally, once all arrayed objects are flattened, Algorithm 1 evaluates all selectors and replaces them with the corresponding flattened ids.

Algorithm 1: flattenArrays(SBMLDocument document)

```
1 foreach sBase in document do
2   remove(document, sBase);
3   expandDimensions(document, sBase, dimensionCount(sBase) - 1);
4 Evaluate all selectors and replace with new ids;
```

Algorithm 2: expandDimensions(SBMLDocument document, SBase sBase, int currentDimension)

```
1 if currentDimension ≥ 0 then
2   for i := 0; i < size(sBase, currentDimension); i ++ do
3     sBase' := clone(sBase);
4     id(sBase', id(sBase) + “_” + i);
5     replace(sBase', dimensionId(currentDimension), i);
6     metaid(sBase', metaid(sBase) + “_” + i);
7     expandDimensions(document, sBase', currentDimension-1);
8 else
9   foreach attribute with an index for sBase do
10    for i := indexCount(sBase, attribute) - 1; i ≥ 0; i -- do
11      attribute(sBase) := attribute(sBase) + “_” + evaluate(indexMath(sBase, attribute, i));
12  add(document, sBase);
```

After performing the steps of the flattening algorithm, the result should be evaluated for validity according to the normal rules of SBML Level 3 Version 1 Core and (if applicable) the rules defined by any other Level 3 packages used in the model. However, it is our belief that a valid SBML document with arrays will always flatten into an valid SBML document without arrays.

It should be noted that the algorithm above assumes for simplicity that the new id's generated are unique. If there are id's already that end with “_” and a number, this may not be the case. In that situation, the algorithm will need to add additional underscores until it finds a unique id. It should also be noted that the algorithm above is further complicated when objects with dimensions have children which can also have dimensions. In SBML Level 3 Version 1 Core, the two types of objects that need special consideration are **Event** and **Reaction**. For an **Event**, it is necessary to expand its dimensions before expanding the dimensions of the **EventAssignment** objects that are its children. For a **Reaction**, it is necessary to expand its dimensions before expanding the dimensions of the **SpeciesReference**

objects that are its children. Furthermore, since the **SpeciesReference** ids are in the global namespace, they must be given unique ids as the **Reaction** dimensions are expanded.

TODO: SpeciesReferences in the global name space really complicate flatten. Is this really necessary?

Acknowledgments

We thank all the participants of the Harmony and Combine workshops and those involved in the Arrays Package Working Group who contributed ideas and discussion over the years.

In addition we would like to thank the authors (esp. Mike and Lucian) of the ‘contrib’ package whose text was used as an inspiration (and occasionally literally).

References

Biron, P. V. and Malhotra, A. (2000). XML Schema part 2: Datatypes (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-2/>.

Eriksson, H.-E. and Penker, M. (1998). *UML Toolkit*. John Wiley & Sons, New York.

Fallside, D. C. (2000). XML Schema part 0: Primer (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at <http://www.w3.org/TR/xmlschema-0/>.

Oestereich, B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*.

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures (W3C candidate recommendation 24 October 2000). Available online via the World Wide Web at the address <http://www.w3.org/TR/xmlschema-1/>.

W3C (2000). W3C's math home page. Available via the World Wide Web at <http://www.w3.org/Math/>.