

The Distributions Package for SBML Level 3

Authors

Stuart L Moodie
moodie@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Lucian P Smith
lpsmith@uw.edu
California Institute of Technology
Seattle, WA, USA

Contributors

Nicolas Le Novère
lenov@babraham.ac.uk
Babraham Institute
Babraham, UK

Darren Wilkinson
darren.wilkinson@ncl.ac.uk
University of Newcastle
Newcastle, UK

Maciej Swat
mjswat@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Sarah Keating
skeating@ebi.ac.uk
EMBL-EBI
Hinxton, UK

Colin Gillespie
c.gillespie@ncl.ac.uk
University of Newcastle
Newcastle, UK

Version 0.11 (Draft)

17 May, 2013

Disclaimer: This is a working draft of the SBML Level 3 “distib” package proposal. It is not a normative document. Please send comments and other feedback to the mailing list:
sbml-distrib@lists.sourceforge.net.



Contents

1	Introduction and motivation	4
1.1	What is it?	4
1.2	Scope	4
1.3	This Document	4
1.4	Conventions used in this document	5
2	Background	6
2.1	Problems with current SBML approaches	6
2.2	Past work on this problem or similar topics	6
2.2.1	The Newcastle Proposal	6
2.2.2	Seattle 2010	6
2.2.3	Hinxton 2011	7
2.2.4	HARMONY 2012: Maastricht	8
2.2.5	COMBINE 2012: Toronto	8
2.2.6	2013 Package Working Group discussions	8
3	Proposed syntax and semantics	9
3.1	Overview	9
3.2	Namespace URI and other declarations necessary for using this package	9
3.3	Primitive data types	10
3.3.1	Type <code>UncertId</code>	10
3.3.2	Type <code>UncertIdRef</code>	10
3.4	Defining Distributions	10
3.4.1	The approach	10
3.4.2	Design Overview	11
3.5	The extended <code>FunctionDefinition</code> class	11
3.6	The <code>DrawFromDistribution</code> class	12
3.7	The <code>Input</code> class	12
3.8	The <code>DistributionOrSample</code> class	12
3.9	The <code>ExplicitPDF</code> class	13
3.10	Discrete vs. continuous sampling	13
3.11	Truncation	14
3.12	Examples using the extended <code>FunctionDefinition</code>	14
3.12.1	Defining and using a normal distribution with <code>UncertML</code>	14
3.12.2	Defining a normal distribution with an explicit PDF	15
3.12.3	Defining a 'die roll' PMF with <code>UncertML</code>	16
3.12.4	Defining a 'pick one' sample with <code>UncertML</code>	17
3.13	Equivalence with Fallback Function	18
3.14	The extended <code>SBase</code> class	18
3.15	The <code>Uncertainty</code> class	19
3.16	The <code>UncertML</code> class	20
3.17	The <code>Math</code> class	20
3.18	Examples using extended <code>SBase</code>	20
3.18.1	Basic <code>UncertML</code> example	20
3.18.2	Combining <code>UncertML</code> and <code>Math</code>	21
3.18.3	Basic <code>Math</code> example	22
3.18.4	An <code>Math</code> example using indices	23
4	Interaction with other packages	25
5	Use-cases and examples	26
5.1	Sampling from a distribution: PK/PD Model	26
5.2	Truncated distribution	28
5.3	Multivariate distribution	29
5.4	User-defined continuous distribution	31
5.5	User-defined discrete distribution	33
6	Prototype implementations	35
7	Unresolved issues	36
8	Acknowledgements	38
A	Distributions incorporated from <code>UncertML</code>	39
	References	41

Revision History

Version	Date	Author	Comments
0.1 (Draft)	15 Oct 2011	Stuart Moodie	First draft
0.2 (Draft)	16 Oct 2011	Stuart Moodie	Added introductory text and background info. Other minor changes etc.
0.3 (Draft)	16 Oct 2011	Stuart Moodie	Filled empty invocation semantics section.
0.4 (Draft)	4 Jan 2012	Stuart Moodie	Incorporated comments from NIN, MS and SK. Some minor revisions and corrections.
0.5 (Draft)	6 Jan 2012	Stuart Moodie	Incorporated addition comments on aim of package from NIN.
0.6 (Draft)	19 Jul 2012	Stuart Moodie	Incorporated revisions discussed and agreed at HARMONY 2012.
0.7 (Draft)	6 Aug 2012	Stuart Moodie	Incorporated review comments from Maciej Swat and Sarah Keating.
0.8 (Draft)	21 Dec 2012	Stuart Moodie	Incorporated changes suggested at combine and subsequently through list discussions.
0.9 (Draft)	9 Jan 2013	Stuart Moodie	Incorporated corrections and comments from Maciej Swat and Sarah Keating.
0.10 (Draft)	10 Jan 2013	Stuart Moodie	Modified based on comments from MS.
0.11 (Draft)	17 May 2013	Lucian Smith	Modified based on Stuart's proposals and PWG discussion.

1 Introduction and motivation

1.1 What is it?

The Distributions package (also affectionately known as *distrib* for short) provides an extension to SBML Level 3 that enables a model to encode and sample from both discrete and continuous probability distributions, and provide the ability to annotate elements with information about the distribution their values were drawn from. Applications of the package include for instance descriptions of population based models: an important subset of which are pharmacokinetic/pharmacodynamic (PK/PD) models¹, which are used to model the action of drugs.

Note that originally the package was called Distributions and Ranges, but Ranges and the use of probability distributions to describe statistical uncertainty are no longer in the scope, hence the name change.

1.2 Scope

The Distributions package adds support to SBML for sampling from a probability distribution. In particular the following are in scope:

- Sampling from a continuous distribution.
- Sampling from a discrete distribution.
- Sampling from user-defined probability density function.
- Sampling from user-defined discrete probability density function.
- The specification of descriptive statistics (mean, standard deviation, standard error, etc.).

At one point the following were considered for inclusion in this package but are now **out of scope**:

- Stochastic differential equations.
- Other functions used to characterise a probability distribution, such as cumulative distribution functions (CDF) or survival functions, etc.

1.3 This Document

This proposal describes the consensus view of workshop participants and subscribers to the sbml-distrib mailing list. Although it was written by the listed author(s) it does not solely reflect their views nor is it their proposal. Rather, it is their understanding of the consensus view of what the Distributions package should do and how it should do it. The contributors listed have made significant contributions to the development and writing of this specification and are credited accordingly, but a more comprehensive attribution is provided in the acknowledgements (Section 8 on page 38).

Finally, there are issues that have arisen during the writing of this document (Section 7). It is important that they are considered and resolved and so the author(s) would encourage the reader to consider them and contribute their ideas or comments — indeed any feedback about this proposal — to the *distrib* discussion list².

Once the proposal is finalised this will be the first step towards the formal adoption of the *distrib* as a package in SBML Level 3. After this, two implementations based on this proposal are required and then a vote on its adoption by the SBML community. The proposal will then provide the basis for a future package specification document. More details of the SBML package adoption process can be found at: http://sbml.org/Documents/SBML_Development_Process.

¹for more information see: <http://www.pharmpk.com/>.

²sbml-distrib@lists.sourceforge.net

1.4 Conventions used in this document

As we are early in the package proposal process there will be some parts of this proposal where there is no clear consensus on the correct solution or only recent agreement or agreement by a group which may not be representative of the SBML community as a whole. These cases are indicated by the question mark in the left margin (illustrated). The reader should pay particular attention to these points and ideally provide feedback, especially if they disagree with what is proposed. Similarly there will be points — especially as the proposal is consolidated — which are agreed, but which the reader should take note of and perhaps read again. These points are emphasised by the hand pointer in the left margin (illustrated).

?



2 Background

2.1 Problems with current SBML approaches

SBML Level 3 Core has no direct support for encoding random values within a model. Currently there is no workaround within the core language itself, although it is possible to define such information using annotations within SBML itself. Frank Bergmann had proposed such an semi-formalised extension for use with SBML L2 [REF?].

2.2 Past work on this problem or similar topics

2.2.1 The Newcastle Proposal

In 2005 there was a proposal from Colin Gillespie and others³ to introduce support for probability distributions in the SBML core specification. This was based on their need to use such distributions to represent the models they were creating as part of the BASIS project (<http://www.basis.ncl.ac.uk>).

They proposed that distributions could be referred to in SBML using the **csymbol** element in the MathML subset used by the SBML Core specification. An example is below:

```
<xmlns='http://www.w3.org/1998/Math/MathML' '>
<apply>
  <csymbol encoding='text'
    definitionURL='http://www.sbml.org/sbml/symbols/uniformRandom' '>
    uniformRandom
  </csymbol>
  <ci>mu</ci>
  <ci>sigma</ci>
</apply>
</math>
```

This required that a library of definitions be maintained as part of the SBML standard and in their proposal they defined an initial small set of commonly used distributions. The proposal was never implemented.

2.2.2 Seattle 2010

The “distrib” package was discussed at the Seattle SBML Hackathon⁴ and this section is an almost verbatim reproduction of Darren Wilkinson’s report on the meeting⁵. There Darren presented an overview of the problem^{6,7}, building on the old proposal from the Newcastle group (see above: Section 2.2.1). There was broad support at the meeting for development of such a package, and for the proposed feature set. Discussion following the presentation led to a consensus on the following points:

- There is an urgent need for such a package.
- It is important to make a distinction between a description of uncertainty regarding a model parameter and the mechanistic process of selecting a random number from a probability distribution, for applications such as parameter scans and experimental design
- It is probably worth including the definition of PMFs, PDFs and CDFs in the package
- It is worth including the definition of random distributions using particle representations within such a package, though some work still needs to be done on the precise representation

³http://sbml.org/Community/Wiki/SBML_Level3_Proposals/Distributions_and_Ranges

⁴http://sbml.org/Events/Hackathons/The_2010_SBML-BioModels.net_Hackathon

⁵<http://sbml.org/Forums/index.php?t=tree&goto=6141&rid=0>

⁶Slides: <http://sbml.org/images/3/3b/Djw-sbml-hackathon-2010-05-04.pdf>

⁷Audio: <http://sbml.org/images/6/67/Wilkinson-distributions-2010-05-04.mov>

- It could be worth exploring the use of xinclude to point at particle representations held in a separate file
- Random numbers must not be used in rate laws or anywhere else that is continuously evaluated, as then simulation behaviour is not defined
- Although there is a need for a package for describing extrinsic noise via stochastic differential equations in SBML, such mechanisms should not be included in this package due to the considerable implications for simulator developers
- We probably don't want to layer on top of UncertML (www.uncertml.org), as this spec is fairly heavy-weight, and somewhat tangential to our requirements
- A random number seed is not part of a model and should not be included in the package
- The definition of truncated distributions and the specification of hard upper and lower bounds on random quantities should be considered.

It was suggested that new constructs should be introduced into SBML by the package embedded as user-defined functions using the following syntax:

```
<listOfFunctionDefinitions>
  <functionDefinition id="myNormRand">
    <distrib:####>
      #### distrib binding information here ####
    </distrib:####>
    <math>
      <lambda>
        <bvar>
          <ci>mu</ci>
          <ci>sigma</ci>
        </bvar>
        <ci>mu</ci>
      </lambda>
    </math>
  </functionDefinition>
</listOfFunctionDefinitions>
```

which allows the use of a "default value" by simulators which do not understand the package (but simulators which do will ignore the <math> element). The package would nevertheless be "required", as it will not be simulated correctly by software which does not understand the package.

Informal discussions following the break-out covered topics such as:

- how to work with vector random quantities in the absence of the vector element in the MathML subset used by SBML
- how care must be taken with the semantics of random variables and the need to both:
 - reference multiple independent random quantities at a given time
 - make multiple references to the same random quantity at a given time.

2.2.3 Hinxton 2011

Detailed discussion was continued at the Statistical Models Workshop in Hinxton in June 2011⁸. There those interested in representing Statistical Models in SBML came together to work out the details of how this package would work in detail. Dan Cornford from the UncertML project⁹ attended the meeting and described how that resource could be used to describe uncertainty and in particular probability distributions. Perhaps the most

⁸http://sbml.org/Events/Other_Events/statistical_models_workshop_2011

⁹<http://www.uncertml.org/>

significant decision at this meeting was to adopt the UncertML resource as a controlled vocabulary that is referenced by the Distributions package.

Much has changed since this meeting, but the output from this meeting was the basis for the first version of this proposal.

2.2.4 HARMONY 2012: Maastricht

Two sessions were dedicated to discussion of Distributions at HARMONY based around the proposals described in version 0.5 of this document. In addition there was discussion about the Arrays and Sets proposal which was very helpful in solving the problem of multivariate distributions in Distributions. The following were the agreed outcomes of the meeting:

- The original proposal included UncertML markup directly in the function definition. This proved unwieldy and confusing and has been replaced by a more elegant solution that eliminates the UncertML markup and integrates well with the fallback function (see details below).
- Multivariate distributions can be supported using the Arrays and Sets package to define a covariance matrix.
- User defined continuous distributions would define a PDF in MathML.
- Usage semantics were clarified so that invocation of a function definition implied a value was sampled from the specified distribution.
- It was agreed from which sections of an SBML model a distribution could be invoked.
- Statistical descriptors of variables (for example mean and standard deviation) would be separated from Distributions and either provided in a new package or in a later version of SBML L3 core.

2.2.5 COMBINE 2012: Toronto

The August proposal was reviewed and an improvement was agreed to the user-defined PMF part of the proposal. In particular it was agreed that the categories should be defined by *distrib* classes rather than by passing in the information as an array. Questions were also raised about whether UncertML was suitably well defined to be used as an external definition for probability distributions. This was resolved subsequent to the meeting with a teleconference to Dan Cornford and colleagues. These changes are incorporated here. Finally, there was considerable debate about whether to keep the dependence of *distrib* on the Arrays package in order to support multi-variate distributions. The outcome was an agreement that we would review this at the end of 2012, based on the results of an investigation into how feasible it would be to implement Arrays and Sets as a package.

2.2.6 2013 Package Working Group discussions

Early 2013 saw a good amount of discussion on the *distrib* Package Working Group mailing list, spurred by proposals by Stuart Moodie¹⁰. While not all of his suggestions ended up being fully accepted by the group, several changes were accepted, including:

- To use UncertML as actual XML, instead of as a set of reference definitions.
- To use UncertML to encode descriptive statistics of SBML elements such as mean, standard deviation, standard error, etc.) bringing this capability back in scope for this package.

¹⁰<http://thestupott.wordpress.com/2013/03/12/an-improved-distrib-proposal/>

3 Proposed syntax and semantics

3.1 Overview

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 never instantiated directly, but rather serve as parents of other 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 tokens *other* than SBML 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.

3.2 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 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 Distributions package for SBML Level 3 Version 1 Core:

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

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model. This is done using the attribute **required** on the `<sbml>` element in the SBML document. For the Distributions package, the value of this attribute must be **"true"** if there are any **DrawFromDistribution** elements in the model, as these elements override the core definition of a

FunctionDefinition. Otherwise, the value of this attribute must be “**false**”, because the other elements of the Distributions package do not change or add to the mathematical meaning of the model.

The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 1 Core and this version of the Distributions package:

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

3.3 Primitive data types

The Distributions package uses the “**string**” primitive data type described in Section 3.1 of the SBML Level 3 Version 1 Core specification, and adds two additional primitive types described below.

3.3.1 Type **UncertId**

The type **UncertId** is derived from **SIId** (SBML Level 3 Version 1 Core specification Section 3.1.7) and has identical syntax. The **UncertId** type is used to create local IDs that can be used in the extended **FunctionDefinition** objects to refer to the arguments of the function, in much the same way that the identities of the **bvar** elements are used in MathML **lambda** elements. Each **UncertId** has a scope local to the **FunctionDefinition** in which it is found. The equality of **UncertId** values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.3.2 Type **UncertIdRef**

Type **UncertIdRef** is used for references to identifiers of type **UncertId**. This type is derived from **UncertId**, but with the restriction that the value of an attribute having type **UncertIdRef** must match the value of a **UncertId** found in the same parent **FunctionDefinition** as the **UncertIdRef**. As with **UncertId**, the equality of **UncertIdRef** values is determined by exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

3.4 Defining Distributions

3.4.1 The approach

The Distributions package has two very simple purposes. First, it provides a mechanism for sampling a random value from a probability distribution. This implies that it must define the probability distribution and then must sample a random value from that distribution.

Secondly, it provides a mechanism for annotating elements with information about their uncertainty. By far the most common use case for this will be to provide the standard deviation for a value, but may also be used to define things such as the exact distribution or the set of samples from which a value was drawn.

There are two ways to define a probability distribution in *distrib*. The first is to use **UncertML** to define a distribution or a probability mass function (PMF). The second way is to define a probability density function (PDF) using MathML.

Strictly speaking, all we need are the explicit PDF and PMF definitions. However, one advantage of using the **UncertML** pre-defined distributions is that software can easily recognise the distribution and use an optimised built-in implementation rather than interpreting the distribution from the PDF definitions. For some applications such optimisations make important performance differences. Another advantage is that some software may only support certain types of distributions, and having them predefined makes it simpler for the software to inform a user that a particular distribution is not supported.

As mentioned above, the distribution must be sampled. In *distrib* the distributions are sampled when they are invoked. To reuse a sampled value the value must be assigned to a parameter first, such as through the use of an **InitialAssignment** or **EventAssignment**.

3.4.2 Design Overview

SBML Core is extended by extending the **FunctionDefinition** class as can be seen in the UML representation in Figure 1. The redefined **FunctionDefinition** optionally contains a single **drawFromDistribution** child.

The **FunctionDefinition** class must still contain the MathML block containing the standard SBML function definition. This is required to comply with the *Validity after Reduction* rule in the package design guidelines [SBML Editorial Board \(2012\)](#) and ensures a degree of backwards compatibility for SBML readers and validators that do not understand the *distrib* package.

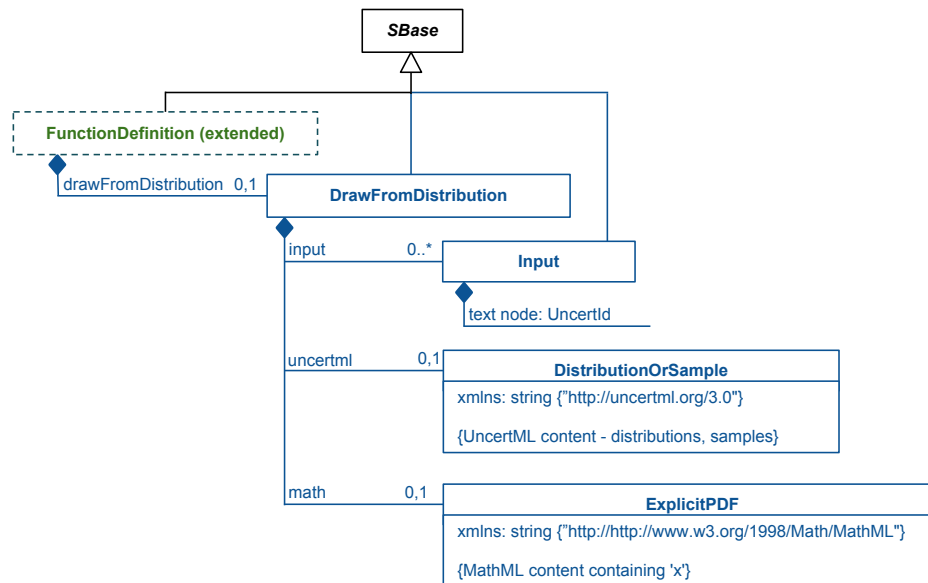


Figure 1: The definition of the extended **FunctionDefinition** class, plus the **DrawFromDistribution**, **Input**, **DistributionOrSample**, and **ExplicitPDF** classes. A **DrawFromDistribution** element must have exactly one **DistributionOrSample** or **ExplicitPDF** child. Together, these classes provide a way to transform a **FunctionDefinition** to sample from a probability density function or a probability mass function.

3.5 The extended **FunctionDefinition** class

As outlined above, the **FunctionDefinition** class is extended to contain a **DrawFromDistribution** child. Because a **FunctionDefinition** must have a **Lambda** child according to SBML Level 3 Version 1 Core, a valid function will still have one, but the **drawFromDistribution** child, if present, will override any definition found there. However, software that does not support *distrib* could potentially invoke the function found in the **Lambda** element (see [Section 3.13](#)).

In the **Model**, an extended **FunctionDefinition** can have two different uses. First, it may be used in any MathML to perform a draw from a distribution. This draw will be unique for every use of the **FunctionDefinition**, whether or not the draw is performed at the same simulation time as a different draw (for example, if used in two different **InitialAssignment** elements).

Secondly, it may be used in an **Uncertainty** element to denote that a particular element has the uncertainty defined by the extended function. See [Section 3.17 on page 20](#) for more details.

3.6 The `DrawFromDistribution` class

As illustrated in Figure 1 on the previous page, the `DrawFromDistribution` class may have any number of `Input` children, which act as the arguments to the function—they serve the same role as the `bvar` elements of the `Lambda` child of a `FunctionDefinition`. As such, order is important: the first `Input` child represents the first argument to the function, etc.

It must also have one other child: a `DistributionOrSample` if it is intended to represent a probability density function (PDF) or probability mass function (PMF) defined by UncertML, or an `ExplicitPDF` if it is intended to represent an explicit PDF represented in MathML. Each of these situations is described below. In all cases, the IDs defined by the `Input` objects are used as the variables within the distribution.

3.7 The `Input` class

The `Input` class mimics the `bvar` elements of MathML lambda functions. It has a single text node child, which must contain a `string` (optionally surrounded by whitespace) of type `UncertId`. These IDs represent the arguments to the function, and serve as local identifiers, referenced only by the other children of the `DrawFromDistribution` class. See the examples in Section 3.12 for more details.

Because the `Lambda` child of the `FunctionDefinition` is required, it must have the same number of `bvar` children as the `DrawFromDistribution` has `Input` children. They do not, however, have to have the same IDs: the `bvar` ids are defined as being local to the `Lambda` function in much the same way that the `Input` IDs are defined as being local to the `DrawFromDistribution` object.

The value “`x`” may not be used as an `UncertId` in an `Input` element; that symbol is reserved for use by the `ExplicitPDF` element (see Section 3.9).

3.8 The `DistributionOrSample` class

The `DistributionOrSample` class, like the `Math` class of many SBML Level 3 Version 1 Core elements, is a container for UncertML XML. It must contain one of the 28 ‘Distribution’ elements or one of the three ‘Samples’ elements (RandomSample, SystematicSample, or UnknownSample) defined at <http://uncertml.org/dictionary>, and have the namespace “`http://uncertml.org/3.0`”. Note that as of this writing, UncertML is only formally defined through version 2.0, making this use of version 3.0 somewhat speculative. However, the *distrib* Package Working Group has been in communication with the developers of UncertML, and feel certain that the change needed in UncertML to accomodate its use in this package (namely, the possible substitution of IDs for numbers) will be made for UncertML 3.0.

When a `DistributionOrSample` is encountered, its parent `FunctionDefinition` is defined as sampling from the defined distribution, and returning that sample. It may contain any number of `UncertIdRef` strings, each of which must correspond to an `UncertId` defined in an `Input` in the same function.

The full list of the 28 distributions and how they can be used is provided in Section A. Four of these distributions (Dirichlet, Multinomial, Multivariate Normal, and Multivariate Student T) use vectors as both input and output. It is possible, if tedious, to provide vector input to these distributions by simply defining each element of the required vector as a numeric value or as an `UncertId`. However, it is not possible in SBML Level 3 Version 1 Core to take a single vector and simultaneously assign its values to different elements, or even to use a vector within MathML. While it would be theoretically possible to define new elements in this specification to work around this limitation, such capabilities are more obviously the domain of the Arrays package within SBML, or of SED-ML¹¹ (to set a suite of element values). Unfortunately, as of this writing, the Arrays package has not yet been finalized, and many aspects of it have not been set. Therefore, it is left to the future finalized Arrays package to define how to utilize a `FunctionDefinition` that returns a vector, and how to define a `FunctionDefinition` that takes a vector as input. It is also possible for other individuals or groups to come up with custom annotations that define how to do this, and in fact, this is encouraged for any group that requires the use of any of these four distributions for their models. If no such

¹¹<http://sed-ml.org/>

definitions exist, however, any numerical results from the use of these functions remain undefined, and therefore unsimulatable. (Such models may still be useful descriptions of certain situations, however.) A final possibility is that SED-ML could be extended to extract the function definition, perform the sampling itself, and use the resulting vector to assign initial values to certain elements.


All three **Sample** elements in UncertML are logically identical, and only semantically different. The **RandomSample** describes a set of realizations known to come from a randomly-distributed source. The **SystematicSample** describes a set of deliberately chosen realizations, such as those resulting from unscented sampling methods for Gaussian random variables. The **UnknownSample** describes a set of realizations for which the source distribution is unknown. All three describe a set of realizations, each with a weight and one or more values. The weights of all realizations in a single **Sample** must sum to 1.0.

UncertML also allows the definition of realizations with “**categories**” instead of “**values**” (i.e. **strings** instead of **doubles**). This data type is unsupported in SBML Level 3 Version 1 Core, but if a package supports this data type in the future (as the Qualitative Models package might), “**categories**” could be used.

When a **DistributionOrSample** is encountered containing a **Sample**, its parent **FunctionDefinition** is defined as randomly choosing a single realization from the the UncertML according to its **weight**, and, if the realization contains a **values** child, returning that value (or vector of values) when the function is called. If the realization contains a **categories** child, that string (or vector of strings) is returned instead. (UncertML requires that a single realization contain either a **values** or **categories** child, but may not have both.)

As happens with some of the UncertML distributions above, then, if the chosen realization has multiple values or categories, the result of the draw is a vector, which cannot, in SBML Level 3 Version 1 Core, be used either in MathML or as an assignment to an SBML element. The Arrays package or some form of custom annotation must be used to define what happens when a multi-value realization is chosen.

Similarly, there is no **string** data type for SBML elements defined in SBML Level 3 Version 1 Core. Without a package that defines such a data type, then, any call to a **FunctionDefinition** that returns a string or vector of strings from a **categories** UncertML element will be undefined and therefore unsimulatable, but may, again, be a useful description of a particular type of model.

 Note that the **ExplicitPMF** class in previous versions of this specification did not use UncertML, and instead defined its own way of listing samples. The functionality (with the addition of IDs instead of numbers in UncertML 3.0) is identical.

3.9 The **ExplicitPDF** class

The **ExplicitPDF** class is a container for MathML XML. It must define a function, using the same MathML allowed in the rest of the SBML Document, for the value “**x**” that integrates to 1.0 over all **x**, and which never evaluates to a negative value for any **x**. It may also use the **UncertIDRef** identifiers defined by its sibling **Input** elements, and the **SIDs** of any other **FunctionDefinition**, but no other identifier. For this reason, “**x**” is illegal to use as an **UncertId** in any **Input** element in the model: it is reserved for use by the **ExplicitPDF** class. As is the case in SBML Level 3 Version 1 Core, circular function definitions are illegal: the **SId** of a **FunctionDefinition** may not be used in its own child **ExplicitPDF**, nor may it use the **SId** of a **FunctionDefinition** that references itself, etc.

To draw a value from an explicit PDF, choose a value between 0 and 1 (non-inclusive), and integrate the function from $x=-\infty$ until your chosen value is reached, and return the value of **x** at that point. For example, if you chose the value 0.5 when integrating an explicit Gaussian function, you would return the mean of that function.

3.10 Discrete vs. continuous sampling


The **SIDs** of **FunctionDefinition** elements can be used in SBML Level 3 Version 1 Core in both discrete and continuous contexts: **InitialAssignment**, **EventAssignment**, **Priority**, and **Delay** elements are all discrete, while **Rule**, **KineticLaw**, and **Trigger** elements are all continuous in time. For discrete contexts, the behavior of *distrib*-extended **FunctionDefinition** elements is well-defined: one or more random values are sampled from the distribution each time the


function definition is invoked. Each invocation implies one sampling operation. In continuous contexts, however, their behavior is ill-defined. More information than is defined in this package (such as autocorrelation values or full conditional probabilities) would be required to make random sampling tractable in continuous contexts, and is beyond the scope of this version of the package. If some package is defined in the future that adds this information, or if custom annotations are provided that add this information, such models may become simulatable. However, this package does not define how to handle sampling in continuous contexts, and recommends against it: a warning may be produced by any software encountering the use of a *distrib*-extended **FunctionDefinition** in a continuous context. Assuming such models are desirable, and the information is not provided in a separate package, this information may be incorporated into a future version of this specification.

Any other package that defines new contexts for MathML will also either be discrete or continuous. Discrete situations (such as those defined in the Qualitative Models package) are, as above, well-defined. Continuous situations (as might arise within the Spatial Processes package, over space instead of over time) will most likely be ill-defined. Those packages must therefore either define for themselves how to handle *distrib*-extended **FunctionDefinition** elements, or leave it to some other package/annotation scheme to define how to handle the situation.

3.11 Truncation

In order to perform truncation, one or both ends of the distribution are cut off, and the remaining function is re-scaled so the area under the curve is once again 1.0. Earlier versions of this specification allowed one to specify truncation values for UncertML distributions and explicit PDFs: upper and lower limits to the distributions, between which values must be sampled. However, this capability is being introduced in UncertML 3.0, making truncation elements superfluous for **DistributionOrSample** elements, and explicit PDFs can be constructed to be truncated through the use of the **piecewise** MathML construct and a scaling factor, making truncation elements superfluous for **ExplicitPDF** elements as well. Truncation was never considered for inclusion for explicit PMF functions, as truncating a sample is as simple as not including the rejected realizations. Therefore, since its capabilities are duplicated elsewhere, truncation-specific elements have been dropped from this version of the specification.

 *Note: If the truncation elements were desired as a form of annotation, denoting the range between which most, if not all, of a given distribution would fall, that capability is not replicated in the other functions above, and would need to be re-added to this specification.*

 *Note 2: If UncertML 3.0 doesn't actually end up having a truncation element, we can add this back in. Also, if people who would like to use the **ExplicitPDF** element don't want to have to encode truncation in the MathML, it could be added back to there, too.*

3.12 Examples using the extended FunctionDefinition

Several examples are given below that illustrate various uses of an extended **FunctionDefinition**.

3.12.1 Defining and using a normal distribution with UncertML

In the following example, a **FunctionDefinition** is extended to define a draw from an UncertML-defined normal distribution:

```
...
<listOfFunctionDefinitions>
  <functionDefinition id="normal">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <!-- Overridden MathML -->
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> avg </input>
      <input> var </input>
      <uncertml xmlns="http://uncertml.org/3.0">
        <NormalDistribution>
          <mean> avg </mean>
```

```

    <variance> var </variance>
  </NormalDistribution>
</uncertml>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
...

```

Here, the **Input** children of **DrawFromDistribution** define the local **UncertIds** “avg” and “var”, which are then used by the **DistributionOrSample** as the **mean** and **variance** of a normal distribution, as defined by **UncertML**. This function could then be used anywhere the **FunctionDefinition** id “normal” can be used, as for example in an **InitialAssignment**:

```

...
<listOfInitialAssignments>
  <initialAssignment symbol="y">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci> normal </ci>
        <ci> z </ci>
        <cn> 10 </cn>
      </apply>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
...

```

This use would apply a draw from a normal distribution with mean “z” and variance “10” to the SBML element “y”.

3.12.2 Defining a normal distribution with an explicit PDF

In the following example, a **FunctionDefinition** is extended to define a draw from a normal distribution defined in MathML, with “x” as the variable to be sampled. The normal distribution equation is described in [Figure 2](#).

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Figure 2: The equation for a normal distribution.

```

...
<listOfFunctionDefinitions>
  <functionDefinition id="normal">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <!-- Overridden MathML -->
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> mu </input>
      <input> sigma </input>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <apply>
            <divide/>
            <cn> 1 </cn>
          </apply>
          <times/>
          <ci> sigma </ci>
        </apply>
      </math>
    </drawFromDistribution>
  </functionDefinition>
</listOfFunctionDefinitions>

```



```

        <degree>
          <cn> 2 </cn>
        </degree>
      </apply>
    </times/>
    <cn> 2 </cn>
    <pi/>
  </apply>
</apply>
</apply>
</apply>
<exp/>
<apply>
  <times/>
  <apply>
    <divide/>
    <apply>
      <power/>
      <apply>
        <minus/>
        <apply>
          <minus/>
          <ci> x </ci>
          <ci> mu </ci>
        </apply>
      </apply>
    </apply>
    <cn> 2 </cn>
  </apply>
  <cn> 2 </cn>
</apply>
<apply>
  <power/>
  <ci> sigma </ci>
  <cn> 2 </cn>
</apply>
</apply>
</apply>
</math>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
...

```

Again, the **Input** children of **DrawFromDistribution** define the local **UncertId**'s “mu” and “sigma”. Here, they are used by the **ExplicitPDF** as the mean and variance of a normal distribution of “x”. This function could then be used anywhere the **FunctionDefinition** id “normal” can be used, exactly as above.

3.12.3 Defining a 'die roll' PMF with **UncertML**

In the following example, a **FunctionDefinition** is extended to define a draw from an **UncertML**-defined set of explicit PMFs:

```

...
<listOfFunctionDefinitions>
  <functionDefinition id="rolld4">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <!-- Overridden MathML -->
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <uncertml xmlns="http://uncertml.org/3.0">
        <RandomSample>
          <Realisation>

```



```

        <weight> 0.25 </weight>
        <values> 1 </values>
      </Realisation>
    </Realisation>
    <weight> 0.25 </weight>
    <values> 2 </values>
  </Realisation>
  <Realisation>
    <weight> 0.25 </weight>
    <values> 3 </values>
  </Realisation>
  <Realisation>
    <weight> 0.25 </weight>
    <values> 4 </values>
  </Realisation>
</uncertml>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
...

```

No inputs are provided. The four **Realisation** children of **RandomSample** all have equal values for **weight**, and sum to 1.0, as they must. The **values** of each are equally likely to be chosen, therefore, resulting in this function returning “1”, “2”, “3”, or “4”, each with equal probability.

3.12.4 Defining a ‘pick one’ sample with *UncertML*

In the following example, a **FunctionDefinition** is extended to define a draw from an *UncertML*-defined set of samples:

```

...
<listOfFunctionDefinitions>
  <functionDefinition id="pickone">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <!-- Overridden MathML -->
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> A </input>
      <input> B </input>
      <input> C </input>
      <input> D </input>
      <uncertml xmlns="http://uncertml.org/3.0">
        <RandomSample>
          <Realisation>
            <weight> 0.25 </weight>
            <values> A </values>
          </Realisation>
          <Realisation>
            <weight> 0.25 </weight>
            <values> B </values>
          </Realisation>
          <Realisation>
            <weight> 0.25 </weight>
            <values> C </values>
          </Realisation>
          <Realisation>
            <weight> 0.25 </weight>
            <values> D </values>
          </Realisation>
        </uncertml>
      </drawFromDistribution>
    </functionDefinition>
  </listOfFunctionDefinitions>
...

```

In this example, the function 'pickone' is defined, with four arguments, "A", "B", "C", and "D". When called, each argument has an equal chance of being chosen as the return value.

3.13 Equivalence with Fallback Function

The MathML definition directly contained by the **functionDefinition** is not used and is provided to satisfy the *validity after reduction* rule for packages [SBML Editorial Board \(2012\)](#). This rule states that the SBML document must be syntactically valid if all package specific elements are removed from it. To ensure that this is the case the fallback function used in relation to *distrib* must satisfy the following rules:

- the lambda function should have the same number of arguments as its equivalent distribution (defined by *distrib*).
- Each argument should match the type of the equivalent argument in the external function.
- The lambda function should have the same return type as the *sampled* distribution. For example, if a predefined PDF when sampled returns a scalar value, the dummy function should also do so.

Clearly, these rules can only be enforced by a *distrib*-aware validator.

In the following example, the fallback function is coded to simply return "mean", the first argument of the function. Note that the arguments have been given different local IDs ("mean" and "v" instead of "avg" and "var"); their equivalence is based on order, not string matching.

```
<listOfFunctionDefinitions>
  <functionDefinition id="normal">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
        </bvar>
        <bvar>
          <ci> v </ci>
        </bvar>
        <ci> mean </ci>
      </lambda>
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> avg </input>
      <input> var </input>
      <uncertml xmlns="http://uncertml.org/3.0">
        <NormalDistribution>
          <mean> avg </mean>
          <variance> var </variance>
        </NormalDistribution>
      </uncertml>
    </drawFromDistribution>
  </functionDefinition>
</listOfFunctionDefinitions>
```

3.14 The extended *SBase* class

As can be seen in [Figure 3 on the following page](#), the SBML base class **SBase** is extended to include an optional **Uncertainty** child, which may contain an **UncertML** child and/or a **Math** child, each containing what is essentially annotation about the uncertainty of its parent element. In SBML Level 3 Version 1 Core, one should only extend those **SBase** elements with mathematical meaning (so, **Compartment**, **Parameter**, **Reaction**, **Species**, and **SpeciesReference**), or those **SBase** elements with **Math** children (so, **Constraint**, **Delay**, **EventAssignment**, **FunctionDefinition**, **InitialAssignment**, **KineticLaw**, **Priority**, **Rule**, and **Trigger**). This is added here to **SBase** instead of to each of these the various SBML elements so that other packages inherit the ability to extend their own elements in the same

fashion: the **FluxBound** class from the Flux Balance Constraints package has mathematical meaning, for example, and can be given information about the distribution or set of samples from which it was drawn. Similarly, the **FunctionTerm** class from the Qualitative Models package has a **Math** child, which could be similarly extended.

A few SBML elements can interact in interesting ways that can confuse the semantics here. A **Reaction** element and its **KineticLaw** child, for example, both reference the exact same mathematics, and should therefore have the same **Uncertainty**. Similarly, if an **InitialAssignment** assigns to a constant element (**Parameter**, **Species**, etc.), the uncertainty for both should be the same, or only one should be provided.

Other elements not listed above should probably not be given an **Uncertainty** child, as it would normally not make sense to talk about the uncertainty of something that doesn't have a corresponding mathematical meaning. However, because packages or annotations can theoretically give new meaning (including mathematical meaning) to elements that previously did not have them, this is not a requirement.

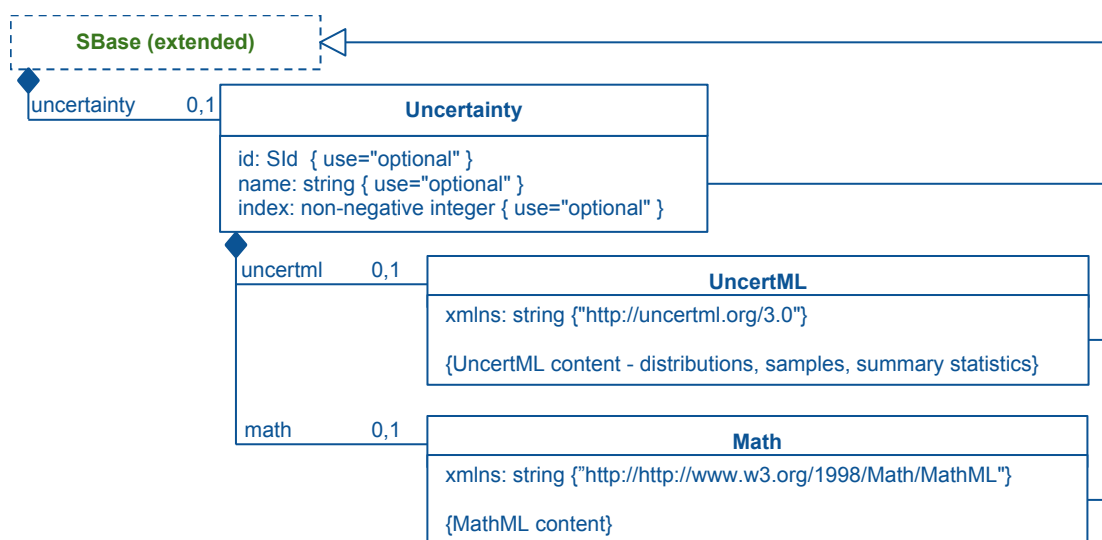


Figure 3: The definition of the extended **SBBase** class to include a new optional **Uncertainty** child, which in turn has optional **UncertML** and **Math** children. Intended for use with any element with mathematical meaning, or with a **Math** child.

3.15 The Uncertainty class

The **Uncertainty** class is a container for **UncertML** and/or **Math** that collectively describes the uncertainty in the parent element's mathematical meaning.

It also contains an optional **index** attribute: this is to assist with function definitions that return a vector, and indicate the position in the vector that corresponds to the parent element's value. For example, a **FunctionDefinition** might define an explicit PMF where each realization contains three values, one for each of three different parameters in the model (see example [Section 3.18.4 on page 23](#)). Each of the three parameters could be given an **Uncertainty** child containing **Math** referencing that **FunctionDefinition**, each with the index of the vector corresponding to its value. Note that the index is zero-indexed: the first position in the vector will have an **index** of "0", the second of "1", etc.

With the release of the Arrays package, presumably the need for this attribute will go away, but it seems a simple enough addition in the absence of that package, and would provide a tangible benefit to many. Even after the release of the Arrays package, some software might not support arrays, but will still want to annotate this information: this attribute will allow them to do so. See [Section 3.18 on the following page](#) for examples of how this might work.

3.16 The **UncertML** class

The **UncertML** class is defined as describing the uncertainty in its parent element's mathematics. It may contain arbitrary **UncertML**, which means it has the capability to provide anything from a full distribution to a **Sample** element to summary statistics such as **StandardDeviation** and **Mean**. There is no requirement that this **UncertML** be wrapped in a single element.

For convenience, **UncertML** provides a **StatisticsCollection**, which may be used to collect multiple other elements. For clarity, if more than one **UncertML** statistics element or **StatisticsCollection** are present, they should not conflict with each other.

The namespace for IDs used in the **UncertML** is the **SI** namespace of elements with mathematical meaning in the model, including from other packages. If that other package is not understood by an interpreter, the **UncertML** element may be ignored. If an interpreter does not understand an ID and cannot tell whether that ID came from a not-understood package, it may issue a warning.

3.17 The **Math** class

The **Math** class child of **Uncertainty**, like the **UncertML** class, describes the uncertainty in its parent element's mathematics. It is provided as a way to reference distributions defined in function definitions elsewhere in the model which have been extended by *distrib*. The most straightforward way to use the **Math** class is to simply include a single call to a **FunctionDefinition**. Other uses, such as adding a constant to a function definition ("**myfunc**(3.1) + 2") are also possible, but discouraged.

The namespace for IDs used in the **MathML** is the **SI** namespace of elements with mathematical meaning in the model, including from other packages. If that other package is not understood by an interpreter, the **Math** element may be ignored. If an interpreter does not understand an ID and cannot tell whether that ID came from a not-understood package, it may issue a warning.

It should be emphasized that this **Math** class is *not* intended to be used like the **ExplicitPDF** class, i.e. as a way to explicitly define a probability density function. Instead, it is a way to provide a *reference* to a probability density function or probability mass function, as defined within an extended **FunctionDefinition**.

3.18 Examples using extended **SBase**

Several examples are given to illustrate the use of the **UncertML** and **Math** classes, both individually and in tandem:

3.18.1 Basic **UncertML** example

In this example, a species is given a **UncertML** child to describe its uncertainty in two ways:

```
...
<species id="x" compartment="C" boundaryCondition="false" initialConcentration="3.22"
  hasOnlySubstanceUnits="false" constant="false">
  <distrib:uncertainty>
    <uncertml xmlns="http://uncertml.org/3.0">
      <NormalDistribution>
        <mean> 3.2 </mean>
        <variance> 0.09 </variance>
      </NormalDistribution>
      <StandardDeviation>
        <values> 0.3 </values>
      </StandardDeviation>
    </uncertml>
  </distrib:uncertainty>
</species>
...
```

In the above example, a species whose initial concentration is 3.22 is noted as coming from a normal distribution with a mean of 3.2 and a variance of 0.09. Additionally, its standard deviation is given as 0.3. In this case, the standard deviation could have been calculated from the variance directly; the modeler has chosen to include both elements for the benefit of any other software that might understand standard deviation, but not the 'distributions' branch of UncertML.

Note also that 3.22 (the `initialConcentration`) is different from 3.2 (the `mean`): evidently, this model was constructed as a realization of the underlying uncertainty, instead of trying to capture the single most likely model of the underlying process.

3.18.2 Combining *UncertML* and *Math*

This example is the same as the previous one, but uses *Math* to reference a **FunctionDefinition** defined as a normal distribution, and *UncertML* to note the standard deviation:

```
...
<listOfFunctionDefinitions>
  <functionDefinition id="normal">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
        </bvar>
        <bvar>
          <ci> variance </ci>
        </bvar>
        <ci> mean </ci>
      </lambda>
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> avg </input>
      <input> var </input>
      <uncertml xmlns="http://uncertml.org/3.0">
        <NormalDistribution>
          <mean> avg </mean>
          <variance> var </variance>
        </NormalDistribution>
      </uncertml>
    </drawFromDistribution>
  </functionDefinition>
</listOfFunctionDefinitions>
<listOfSpecies>
  <species id="x" compartment="C" boundaryCondition="false" initialConcentration="3.22"
    hasOnlySubstanceUnits="false" constant="false">
    <distrib:uncertainty>
      <uncertml xmlns="http://uncertml.org/3.0">
        <StandardDeviation>
          <values> 0.3 </values>
        </StandardDeviation>
      </uncertml>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> normal </ci>
          <cn> 3.2 </cn>
          <cn> 0.09 </cn>
        </apply>
      </math>
    </uncertainty>
  </species>
</listOfSpecies>
...
```

Again, the **UncertML** notes that the standard deviation is 0.3, and the **Math** notes that the mean is 3.2 and the variance is 0.09.

3.18.3 Basic **Math** example

In the following example, a function definition is created with an **ExplicitPDF** child, then used to annotate two parameters.

```
...
<listOfFunctionDefinitions>
  <functionDefinition id="dist1">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <bvar>
          <ci> mean </ci>
        </bvar>
        <ci> mean </ci>
      </lambda>
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <input> mean </input>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <apply>
            <divide/>
            <cn> 1 </cn>
            <apply>
              <times/>
              <cn> 0.011 </cn>
              <apply>
                <root/>
                <degree>
                  <cn> 2 </cn>
                </degree>
                <apply>
                  <times/>
                  <cn> 2 </cn>
                  <pi/>
                </apply>
              </apply>
            </apply>
          </apply>
          <exp/>
          <apply>
            <times/>
            <apply>
              <divide/>
              <apply>
                <power/>
                <apply>
                  <minus/>
                  <apply>
                    <minus/>
                    <ci> x </ci>
                    <ci> mean </ci>
                  </apply>
                </apply>
              </apply>
            </apply>
            <cn> 2 </cn>
          </apply>
          <cn> 2 </cn>
        </apply>
      </math>
    </drawFromDistribution>
  </functionDefinition>
</listOfFunctionDefinitions>
```

```

        <cn> 0.011 </cn>
        <cn> 2 </cn>
      </apply>
    </apply>
  </apply>
</math>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
<listOfParameters>
  <parameter id="y" value="1.1" constant="true"/>
  <distrib:uncertainty>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci> dist1 </ci>
        <cn> 1.1 </cn>
      </apply>
    </math>
  </parameter id="z" value="3.2" constant="true"/>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
      <ci> dist1 </ci>
      <cn> 3.2 </cn>
    </apply>
  </math>
</distrib:uncertainty>
</listOfParameters>
...

```

In the above example, a function definition is provided with an **ExplicitPDF** that defines a distribution with a mean of “mean” and a variance of “0.011”. The parameters “y” and “z” are each annotated with an **Math** child that calls that function, the first with the argument “1.1” (its value), and the second with the argument “3.2” (its value). The effect of adding the **Uncertainty** children, then, is to denote that both values come from a gaussian distribution with a mean of the parameter value, and a variance of 0.011. Note that this use of the *distrib* package does not use UncertML at all.

3.18.4 An **Math** example using indices

In the following example, a function definition is created with a **DistributionOrSample** child that returns a vector, which is used to annotate two parameters and a species.

```

...
<listOfFunctionDefinitions>
  <functionDefinition id="samples">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <lambda>
        <cn> 0.0 </cn>
      </lambda>
    </math>
    <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
      <uncertml xmlns="http://uncertml.org/3.0">
        <UnknownSample>
          <Realisation id="patient1">
            <weight> 0.5 </weight>
            <values> 1.01 42.3 0.004 </values>
          </Realisation>
          <Realisation id="patient2">
            <weight> 0.25 </weight>
            <values> 2.24 45.9 0.002 </values>
          </Realisation>
          <Realisation id="patient3">
            <weight> 0.25 </weight>

```

```

        <values> 1.72 50.0 0.0033 </values>
      </Realisation>
    </UnknownSample>
  </uncertml>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
<listOfSpecies>
  <species id="S1" compartment="C" boundaryCondition="false" initialConcentration="2.24"
    hasOnlySubstanceUnits="false" constant="false">
    <distrib:uncertainty index="0">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> samples </ci>
        </apply>
      </math>
    </distrib:uncertainty>
  </species>
</listOfSpecies>
<listOfParameters>
  <parameter id="y" constant="true" value="45.9">
    <distrib:uncertainty index="1">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> samples </ci>
        </apply>
      </math>
    </distrib:uncertainty>
  </parameter>
  <parameter id="z" constant="true" value="0.002">
    <distrib:uncertainty index="2">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> samples </ci>
        </apply>
      </math>
    </distrib:uncertainty>
  </parameter>
</listOfParameters>
...

```

In this example, the “**samples**” function is never used in any math in the model, and is instead used just for the purposes of annotation. The three realizations it contains (patients 1-3) each have three values associated with them. The second realization (from patient 2) has been assigned to the species “S1” and the parameters “y” and “z”, respectively. Each element was then annotated with **Math** that it came from the “**samples**” distribution, each with an **index** value corresponding to its position in the **values** vector.

4 Interaction with other packages

This package is dependent on no other package, but relies on the Arrays and Sets package to provide vector and matrix structures if those are desired/used.

If the Required Elements package is used, any **FunctionDefinition** that has been given an **DrawFromDistribution** child must be given a **ChangedMath** child referencing this package's namespace. If the fallback function provides a complete **Lambda** function, its **viableWithoutChange** attribute *may* be set “**true**” if the modeler considers that function an acceptable alternative to the draw from the distribution, otherwise it must be set “**false**”.

5 Use-cases and examples

The following examples are more fleshed out than the ones in the main text, and/or illustrate features of this package that were not previously illustrated.

5.1 Sampling from a distribution: PK/PD Model

This is a very straightforward use of an UncertML-defined distribution. The key point to note is that a value is sampled from the distribution and assigned to a variable when it is invoked in the initialAssignments element in this example. Later use of the variable does not result in re-sampling from the distribution. This is consistent with current SBML semantics.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  distrib:required="false">
  <model id="PKModel" name="PKModel">
    <listOfFunctionDefinitions>
      <functionDefinition id="logNormal">
        <!-- below is a dummy function provided to ensure valid SBML Core
            for software that do not understand distrib. -->
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar><ci>scale</ci></bvar>
            <bvar><ci>shape</ci></bvar>
            <cn>0</cn>
          </lambda>
        </math>
        <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
          <input> scale </input>
          <input> shape </input>
          <uncertml xmlns="http://uncertml.org/3.0">
            <LogNormalDistribution xmlns:un="http://www.uncertml.org/2.0">
              <logScale> scale </logScale>
              <shape> shape </shape>
            </LogNormalDistribution>
          </uncertml>
        </drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfCompartments>
      <compartment id="central" name="central" size="0" constant="true"/>
      <compartment id="gut" name="gut" size="0" constant="true"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="Qc" compartment="central" initialAmount="1"
        hasOnlySubstanceUnits="true" boundaryCondition="false" constant="false"/>
      <species id="Qg" compartment="gut" initialAmount="1"
        hasOnlySubstanceUnits="true" boundaryCondition="false" constant="false"/>
    </listOfSpecies>
    <listOfParameters>
      <parameter id="ka" value="1" constant="true"/>
      <parameter id="ke" value="1" constant="true"/>
      <parameter id="Cc" value="1" constant="false"/>
      <parameter id="Cc_obs" value="1" constant="false"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="central">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci>logNormal</ci>
            <cn>0.5</cn>
          </math>

```

```

        <cn>0.1</cn>
      </apply>
    </math>
  </initialAssignment>
  <initialAssignment symbol="ka">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci>logNormal</ci>
        <cn>0.5</cn>
        <cn>0.1</cn>
      </apply>
    </math>
  </initialAssignment>
  <initialAssignment symbol="ke">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci>logNormal</ci>
        <cn>0.5</cn>
        <cn>0.1</cn>
      </apply>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
<listOfRules>
  <assignmentRule variable="Cc">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <divide/>
        <ci> Qc </ci>
        <ci> central </ci>
      </apply>
    </math>
  </assignmentRule>
  <assignmentRule variable="Cc_obs">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <plus/>
        <ci> Cc </ci>
        <cn type="integer"> 1 </cn>
      </apply>
    </math>
  </assignmentRule>
</listOfRules>
<listOfReactions>
  <reaction id="absorption" reversible="false" fast="false">
    <listOfReactants>
      <speciesReference species="Qg" stoichiometry="1" constant="false"/>
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="Qc" stoichiometry="1" constant="false"/>
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci> ka </ci>
          <ci> Qg </ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>
  <reaction id="excretion" reversible="false" fast="false">
    <listOfReactants>
      <speciesReference species="Qc" stoichiometry="1" constant="false"/>
    </listOfReactants>
    <kineticLaw>

```

```

        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <divide/>
            <apply>
              <times/>
              <ci> ke </ci>
              <ci> Qc </ci>
            </apply>
            <ci> central </ci>
          </apply>
        </math>
      </kineticLaw>
    </reaction>
  </listOfReactions>
</model>
</sbml>

```

5.2 Truncated distribution

To encode a truncated distribution we rely on an as-of-this-writing hypothetical extension to UncertML (the **lower** and **upper** child elements of **NormalDistribution**). The final version of truncation in UncertML may not look exactly like this example, but should follow the basics. Note that the values for truncation are here provided as arguments to the function definition, but could instead be hard-coded.

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  distrib:required="true">
  <model id="truncated_example">
    <listOfFunctionDefinitions>
      <functionDefinition id="normal">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
          xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <lambda>
            <bvar><ci>mu</ci></bvar>
            <bvar><ci>sigma</ci></bvar>
            <bvar><ci>lower</ci></bvar>
            <bvar><ci>upper</ci></bvar>
            <ci>mu</ci>
          </lambda>
          </math>
          <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
            <input> mu </input>
            <input> sigma </input>
            <input> lower </input>
            <input> upper </input>
            <uncertml xmlns="http://uncertml.org/3.0">
              <NormalDistribution>
                <mean> mu </mean>
                <variance> sigma </variance>
                <lower> lower </lower>
                <upper> upper </upper>
                <!--Note: 'lower' and 'upper' are proposed children of NormalDistribution
                  in UncertML 3.0, but have not yet been officially endorsed. -->
              </NormalDistribution>
            </uncertml>
          </drawFromDistribution>
        </functionDefinition>
      </listOfFunctionDefinitions>
      <listOfParameters>
        <parameter id="V" constant="true"/>
        <parameter id="V_pop" value="105" constant="true"/>
        <parameter id="V_omega" value="0.70" constant="true"/>
      </listOfParameters>
    </model>
  </sbml>

```

```

<parameter id="V_upper" value="150" constant="true"/>
<parameter id="V_lower" value="15" constant="true"/>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="V">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
      <apply>
        <ci>normal</ci>
        <ci>V_pop</ci>
        <ci>V_omega</ci>
        <ci>V_upper</ci>
        <ci>V_lower</ci>
      </apply>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
</model>
</sbml>

```

5.3 Multivariate distribution

In this example two correlated parameters are sampled from a multivariate distribution. The correlation is defined using a covariance matrix and the sampled values are returned as a vector of 2 values, and assigned to the variable “correlated_params”. This vector is then used to assign values to “V” and “C1”, thereby associating those two values with the same draw from the multivariate normal distribution. The use of various array and matrix MathML here is speculative: in the absence of a finalized Arrays package, it is impossible to tell exactly what form that will take. However, all of the functionality expressed here will need to be incorporated in some form into the Arrays package, and much if not all of it may take the form illustrated here.

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  distrib:required="true"
  xmlns:arrays="http://www.sbml.org/sbml/level3/version1/arrays/version1"
  arrays:required="true">
  <!-- NOTE: This requires the arrays package! -->
  <model id="MultivariateExample" name="Multivariate_Example">
    <listOfFunctionDefinitions>
      <functionDefinition id="multivariateNormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML"
          xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
          <lambda>
            <bvar><ci>meanVector</ci></bvar>
            <bvar><ci>covarianceMatrix</ci></bvar>
            <ci>meanVector</ci>
          </lambda>
        </math>
        <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
          <input> meanVector </input>
          <input> covarianceMatrix </input>
          <uncertml xmlns="http://uncertml.org/3.0">
            <MultivariateNormalDistribution xmlns:un="http://www.uncertml.org/2.0">
              <mean> meanVector </mean>
              <covarianceMatrix dimension="2">
                <values> covarianceMatrix </values>
              </covarianceMatrix>
            </MultivariateNormalDistribution>
          </uncertml>
        </drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>

```

```

<parameter id="V" constant="false"/>
<parameter id="V_pop" value="105" constant="true"/>
<parameter id="V_omega" value="0.70" constant="true"/>
<parameter id="C1" constant="true"/>
<parameter id="C1_pop" value="73" constant="true"/>
<parameter id="C1_omega" value="0.70" constant="true"/>
<parameter id="covariance" constant="true">
  <arrays:listOfDimensions>
    <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
    <arrays:dimension id="j" lowerLimit="1" upperLimit="2" />
  </arrays:listOfDimensions>
</parameter>
<parameter id="correlated_means" constant="true">
  <arrays:listOfDimensions>
    <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
  </arrays:listOfDimensions>
</parameter>
<parameter id="correlated_params" constant="true">
  <arrays:listOfDimensions>
    <arrays:dimension id="i" lowerLimit="1" upperLimit="2" />
  </arrays:listOfDimensions>
</parameter>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="covariance">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:sbml="http://www.sbml.org/sbml/level3/version1/arraymaths/version1">
      <!-- This is an unresolved issue. The l3 V1 Core mathml subset does not support
        matrices. One solution - as above is to use a different MathML subset definition.
      -->
      <matrix>
        <matrixrow>
          <apply><times/><ci>V_omega</ci><ci>V_omega</ci></apply>
          <apply><times/><ci>V_omega</ci><ci>C_omega</ci><ci>V_C_rho</ci></apply>
        </matrixrow>
        <matrixrow>
          <ci>0</ci>
          <apply><times/><ci>C_omega</ci><ci>C_omega</ci></apply>
        </matrixrow>
      </matrix>
    </math>
  </initialAssignment>
  <initialAssignment symbol="correlated_means">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
      <vector>
        <ci>V_pop</ci>
        <ci>C_pop</ci>
      </vector>
    </math>
  </initialAssignment>
  <initialAssignment symbol="correlated_params">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
      <apply>
        <ci>multivariateNormal</ci>
        <ci>correlated_means</ci>
        <ci>covariance</ci>
      </apply>
    </math>
  </initialAssignment>
  <initialAssignment symbol="V">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
      <apply>
        <selector/>

```

```

        <ci>correlated_params</ci>
        <cn type="integer">1</cn>
      </apply>
    </math>
  </initialAssignment>
  <initialAssignment symbol="C1">
    <math xmlns="http://www.w3.org/1998/Math/MathML"
          xmlns:sbml="http://www.sbml.org/sbml/level3/version1/core">
      <apply>
        <selector/>
        <ci>correlated_params</ci>
        <cn type="integer">2</cn>
      </apply>
    </math>
  </initialAssignment>
</listOfInitialAssignments>
<!-- This is an incomplete model snippet, sufficient to illustrate the use of
a multivariate distribution. -->
</model>
</sbml>

```

5.4 User-defined continuous distribution

In this example, an [ExplicitPDF](#) is constructed, and used in three places: to denote the uncertainty in the parameter “V”, the uncertainty in the initial assignment to “V”, and to construct the initial assignment itself. Note that strictly speaking, since “V” is constant, one could assume that the uncertainty in the parameter itself was identical to the uncertainty in its initial assignment; both are given here by way of illustration.

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
      distrib:required="true">
  <model id="UserDefined" name="User_Defined_Example">
    <listOfFunctionDefinitions>
      <functionDefinition id="mynormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar><ci>param1</ci></bvar>
            <bvar><ci>param2</ci></bvar>
            <cn>0</cn>
          </lambda>
        </math>
        <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
          <input> mu </input>
          <input> sigma </input>
          <math>
            <apply>
              <times/>
              <apply>
                <divide/>
                <cn> 1 </cn>
              </apply>
              <times/>
              <ci> sigma </ci>
            </math>
            <apply>
              <root/>
              <degree>
                <cn> 2 </cn>
              </degree>
            </apply>
            <times/>
            <cn> 2 </cn>
            <pi/>
          </math>
        </drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
  </model>
</sbml>

```

```

        </apply>
      </apply>
    </apply>
  </apply>
  <apply>
    <exp/>
    <apply>
      <times/>
      <apply>
        <divide/>
        <apply>
          <power/>
          <apply>
            <minus/>
            <apply>
              <minus/>
              <ci> x </ci>
              <ci> mu </ci>
            </apply>
          </apply>
        </apply>
      <cn> 2 </cn>
    </apply>
    <cn> 2 </cn>
  </apply>
  <apply>
    <power/>
    <ci> sigma </ci>
    <cn> 2 </cn>
  </apply>
</apply>
</apply>
</math>
</drawFromDistribution>
</functionDefinition>
</listOfFunctionDefinitions>
<listOfParameters>
  <parameter id="V" constant="true">
    <distrib:uncertainty>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <ci> mynormal </ci>
          <ci> V_pop </ci>
          <ci> V_omega </ci>
        </apply>
      </math>
    </distrib:uncertainty>
  </parameter>
  <parameter id="V_pop" value="100" constant="true"/>
  <parameter id="V_omega" value="0.25" constant="true"/>
</listOfParameters>
<listOfInitialAssignments>
  <initialAssignment symbol="V">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci> mynormal </ci>
        <ci> V_pop </ci>
        <ci> V_omega </ci>
      </apply>
    </math>
  <distrib:uncertainty>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <ci> mynormal </ci>
        <ci> V_pop </ci>
        <ci> V_omega </ci>

```



```

        </apply>
      </math>
    </distrib:uncertainty>
  </initialAssignment>
</listOfInitialAssignments>
</model>
</sbml>

```

5.5 User-defined discrete distribution

In this example, a [DistributionOrSample](#) is used where the weights themselves are the arguments to the function instead of the values.

```

<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  distrib:required="true">
  <model id="UserDefined" name="User_Defined_Example">
    <listOfFunctionDefinitions>
      <functionDefinition id="myDistribution">
        <drawFromDistribution xmlns="http://www.sbml.org/sbml/level3/version1/distrib/version1">
          <input> wt1 </input>
          <input> wt2 </input>
          <input> wt3 </input>
          <uncertml xmlns="http://uncertml.org/3.0">
            <RandomSample>
              <Realisation>
                <weight> wt1 </weight>
                <values> 2 </values>
              </Realisation>
              <Realisation>
                <weight> wt2 </weight>
                <values> 3 </values>
              </Realisation>
              <Realisation>
                <weight> wt3 </weight>
                <values> 4 </values>
              </Realisation>
            </RandomSample>
          </uncertml>
        </drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>
      <parameter id="W" constant="true"/>
      <parameter id="catProb" constant="true"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="catProb">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply><divide><cn>33</cn><cn>36</cn></apply>
        </math>
      </initialAssignment>
      <initialAssignment symbol="W">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>

```

```

    <ci>myDistribution</ci>
    <!-- Defined mathematically as a fraction -->
    <apply><divide/><cn>1</cn><cn>36</cn></apply>
    <!-- Defined as decimal -->
    <cn>0.05555556</cn>
    <!-- Defined by a reference to a parameter or another variable -->
    <ci>catProb</ci>
  </apply>
</math>
</initialAssignment>
</listOfInitialAssignments>
</model>
</sbml>

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14

6 Prototype implementations

1

None as yet.

2

7 Unresolved issues

1. Should the **ExplicitPDF** class have a truncation attribute? This would allow one to define a truncated distribution without defining the truncated PDF in the MathML definition. Is this desirable or not? *Lucian: I think not. Since explicit PDFs can already be truncated through the use of the 'piecewise' function, let's see how this works out for people first before putting in new functionality that has to be implemented explicitly. (On the other hand, it may turn out that if people have to implement it for UncertML, they may not mind implementing it for explicit PDFs either. In that case if it was much easier to implement truncated explicit PDFs as models, it might be worth adding in some constructs. Either way, I vote for waiting for implementations to clear things up here.)*
2. Currently the proposal provides an appendix that clarifies the parameters that should be specified for each UncertML distribution and their type. It also defines whether one or more values are sampled from a given distribution. There was some argument on the list about whether this appendix is required. An alternative would be to use UncertML definitions as-is, and live with any inherent ambiguity. We should make a decision about the approach we want to take for the final version of this proposal. *Lucian: As is obvious, we did indeed go with using the uncertML as-is. I think it works out a lot better this way, though I would still keep the appendix to ensure people understand the different distributions.*
3. The use of arrays. The package and examples have been written that some distribution definitions will require the use of vector or matrix types that can be provided by the Arrays package. While relying on such types enables us to describe a wide range of probability distributions this will come at the cost of delaying this package's approval until Arrays has been developed and approved. There are two alternative strategies. One is to only support distributions that do not require array types (see appendix [Section A](#)). The other would be to define workarounds within *distrib* to provide array structures without using the arrays package. The decision at COMBINE was to wait. Do we want to revisit this decision? *Lucian: I tried to write this specification in such a way that when Arrays is available, it will work seamlessly with distrib, and until that time, everything that does **not** require Arrays will still work. This will also allow people to implement support for each separately.*
4. Currently sampling of distributions is permitted in the event related elements of SBML: EventAssignments, Delays and Priorities. The reason for prohibiting sampling in a rate law (or other circumstances where the model is being continuously evaluated) was because this can make the simulation unstable¹². It is unclear whether a similar situation could arise while sampling from events. The author feels this need to be looked into further by 'who know'. *Lucian: I believe this has been resolved.*
5. During the review of this proposal it was questioned whether the use of the term PMF was appropriate for the eponymous *distrib* class: **userDefinedPMF**. The issue is that a PMF is defined for (numerical) values which a random variable can assume and not general 'categories'. Since we permit non-numerical values to define categories such as eye colour (red, green, blue etc.) or sex (M, F) it is argued that the current name is mis-leading. UncertML uses the term discrete probability for a similar construct that describes this, so perhaps a better class name would be **UserDefinedDiscreteProbability** or just **DiscreteProbability**. *Lucian: There was a bit of discussion on the list in favor of 'PMF'; I have retained that wording in this version.*
6. Limiting the use of **FunctionDefinition**. In review it was pointed out that the restriction on sampling to InitialAssignment, EventAssignment etc. affects MathML validity rules in SBML. Currently a function definition can be 'called' from any MathML block in an SBML document. However, *distrib* imposes an additional special case. *A function definition that defines a distribution can only be called from MathML, where sampling is permitted.* It is not absolutely clear what the technical implications of this are, but it would appear that to implement *distrib* support any implementation would be required to modify the MathML validation rules in libSBML. Given, that these rules are not designed to interact with packages this may be technically challenging to implement. The author feels that it would be desirable to find an alternative design that avoids the need to modify the MathML validation rules. *Lucian: This was the main topic of discussion on the list, and the result of*

¹²See slide entitled Extrinsic noise: <http://sbml.org/images/3/3b/Djw-sbml-hackathon-2010-05-04.pdf>.

*the survey that we put out at the end was that we should indeed continue to extend FunctionDefinitions, but should *not* limit their use to discrete contexts. Rather, to allow 'continuous' contexts to work in the future, they should be allowed but their behavior now should be undefined, and their use discouraged. That is the tack I have tried to take in this version of the document.*

1
2
3
4

8 Acknowledgements

Much of the initial concrete work leading to this proposal document was carried out at the Statistical Models Workshop in Hinxton in 2011, which was organised by Nicolas le Novère. A list of participants and recordings of the discussion is available from http://sbml.org/Events/Other_Events/statistical_models_workshop_2011. Before that a lot of the ground work was carried out by Darren Wilkinson who led the discussion on *distrib* at the Seattle SBML Hackathon and before that Colin Gillespie who wrote an initial proposal back in 2005. The author would also like to thank the participants of the *distrib* sessions during HARMONY 2012 and COMBINE 2012 for their excellent contributions in helping revising this proposal; Sarah Keating, Maciej Swat and Nicolas le Novère for useful discussions, corrections and review comments; and Mike Hucka for \LaTeX advice and the beautiful template upon which this document is based.

A Distributions incorporated from UncertML

Note: It would be nice if this table was extended to include basic definitions of the various distributions.

The distributions described by UncertML¹³ will provide the standard external definition of distributions used by Distributions. The definitions are comprehensive, but have multiple implicit definitions in some cases. For example the mean and variance parameters of the Normal distribution can be each be described as a vector of k values, where each mean, variance pair describes a different probability distribution. However, in *distrib* we want to define one probability density and so it is necessary to disambiguate definitions where more than one is possible.

The table below describes, for each probability distribution defined in UncertML: the arguments (referred to as parameters in UncertML) it takes, the type of each argument and type of the result obtained when a random value (or values) is sampled from the distribution. The type can be either a scalar (in effect a double because this is the only scalar type supported by SBML), or an array (again of doubles), which can have one or more dimensions.

UncertML Name	Argument		Sampled Result
	Name	Type	
BernouliDistribution	probability	scalar	scalar
BetaDistribution	alpha	scalar	scalar
	beta	scalar	
BinomialDistribution	numberOfTrials	scalar	scalar
	probabilityOfSuccess	scalar	
CauchyDistribution	location	scalar	scalar
	scale	scalar	
ChiSquaredDistribution	degreesOfFreedom	scalar	scalar
DirichletDistribution	concentration	array[k], $k \geq 2$	array[k]
ExponentialDistribution	rate	scalar	scalar
FDistribution	numerator	scalar	scalar
	denominator	scalar	
GammaDistribution	shape	scalar	scalar
	scale	scalar	
GeometricDistribution	probability	scalar	scalar
HypergeometricDistribution	populationSize	scalar	scalar
	numberOfTrials	scalar	
	numberOfSuccesses	scalar	
InverseGammaDistribution	shape	scalar	scalar
	scale	scalar	
LaplaceDistribution	location	scalar	scalar
	scale	scalar	
LogNormalDistribution	logScale	scalar	scalar
	shape	scalar	
LogisticDistribution	location	scalar	scalar
	scale	scalar	

Continued on next page...

¹³These distributions are defined here: <http://www.uncertml.org/distributions>

UncertML Name	Argument		Sampled Result
	Name	Type	
MixtureModel	weight	?	?
MultinomialDistribution	numberOfTrials probabilityOfSuccess	scalar array[k]	array[k]
MultivariateNormalDistribution	mean covarianceMatrix	array[k] array[k][k]	array[k]
MultivariateStudentTDistribution	mean covarianceMatrix degreesOfFreedom	array[k] array[k][k] scalar	array[k]
NegativeBinomialDistribution	numberOfFailures probability	scalar scalar	scalar
NormalDistribution	mean variance	scalar scalar	scalar
NormalInverseGammaDistribution	mean varianceScaling shape scale	scalar	scalar
ParetoDistribution	scale shape	scalar scalar	scalar
PoissonDistribution	rate	scalar	scalar
StudentTDistribution	location scale degreesOfFreedom	scalar scalar scalar	scalar
WeibullDistribution	scale shape	scalar scalar	scalar
WishartDistribution	scaleMatrix degreesOfFreedom	scalar scalar	scalar

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*. Addison-Wesley.

SBML Editorial Board (2012). SBML development process for SBML level 3. Available via the World Wide Web at http://sbml.org/Documents/SBML_Development_Process/SBML_Development_Process_for_SBML_Level_3.

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/>.