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 J Swat
maciej.swat@certara.com
QSP Simcyp
Certara, Sheffield, UK

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

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

Version 0.19 (Draft)

June XX, 2018

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

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 NlN, MS and SK. Some minor revisions and corrections.
0.5 (Draft)	6 Jan 2012	Stuart Moodie	Incorporated addition comments on aim of package from NlN.
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.
0.12 (Draft)	June 2013	Lucian Smith and Stuart Moodie	Modified based on HARMONY 2013 discussion.
0.13 (Draft)	July 2013	Lucian Smith and Stuart Moodie	Modified based PWG discussion, particularly with respect to UncertML.
0.14 (Draft)	March 2015	Lucian Smith	Modified to match UncertML 3.0.
0.15 (Draft)	March 2015	Lucian Smith and Sarah Keating	Modified to match UncertML 3.0 for real this time.
0.16 (Draft)	March 2015	Lucian Smith	Added information about UncertML 3.0 distributions, and the distributions custom annotations.
0.17 (Draft)	June 2017	Lucian Smith	Extensive update to reflect demise of UncertML 3.0, and appearance of ProbOnto.
0.18 (Draft)	June 2017	Lucian Smith	Fixes to reflect feedback on version 0.17.
0.19 (Draft)	June 2018	Lucian Smith	Resolved id/name issues with SBML core l3v1 vs. l3v2.

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) models[1], which are used to model the action of drugs.

Note that originally the package was called Distributions and Ranges, but Ranges 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 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**:

- Sampling from user-defined probability density function.
- 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 authors it does not soley 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 7 on page 44).

Finally, the authors would encourage the reader to consider them and contribute their ideas or comments — indeed any feedback about this proposal — to the *distrib* discussion list[2].

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 the SBML editors must agree that the implementations and specification are complete. 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.

[1] for more information see: http://www.pharmpk.com/.

[2] sbml-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 and L3 (See Section 4.1 on page 36).

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 others [3] 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 Hackathon[4] and this section is an almost verbatim reproduction of Darren Wilkinson's report on the meeting[5]. 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

[3] http://sbml.org/Community/Wiki/SBML_Leve\T1\l_3_Proposals/Distributions_and_Ranges
[4] http://sbml.org/Events/Hackathons/The_2010_SBML-BioModels.net_Hackathon
[5] http://sbml.org/Forums/index.php?t=tree&goto=6141&rid=0
[6] Slides: http://sbml.org/images/3/3b/Djw-sbml-hackathon-2010-05-04.pdf
[7] Audio: 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 2011[8]. 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 project[9] attended the meeting and described how that resource could be used to describe uncertainty and in particular probability distributions. Perhaps the most

[8] http://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 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 package to define a covariance matrix.
- User defined continuous distributions would define a PDF in MathML.
- Usage semantics were clarified so that invokation 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 is 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 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 Moodie[10]. 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.

2.2.7 HARMONY 2013: Connecticut

At HARMONY at UConn in Connecticut, further discussions revealed the importance of distinguishing the ability to describe an element as a distributed variable vs. a function call within the model performing a draw from a distribution.

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

We also decided to discard the encoding of explicit PDFs for now, as support for it is remarkably complicated, and there no demand for it. The current design could be extended to support this feature so if there is demand for it in the future support for explicit PDFs could be reintroduced.

2.2.8 Early 2017 and HARMONY: Seattle

In early 2017, it became clear that UncertML was no longer being worked on; the web page had lapsed, and its authors had moved on to other things. At the same time, the ProbOnto ontology (Swat et al. 2016; `http://probonto.org/`) was developed that included all the distributions from UncertML as well as a huge number of other distributions. On the mailing list, it was discussed whether to create essentially our own version of UncertML, or to implement a generic 'reference' format that used ProbOnto. The v0.17 draft specification was developed as a compromise 'hybrid' system that did parts of both, so that basic distributions would be hard-coded, but the ability to reference any ProbOnto ontology would also be present. The hope is that with working examples of both approaches, either the hybrid approach will be approved, or if one is preferred, the other approach may be removed. This version of the specification was created for presentation at HARMONY 2017 in Seattle.

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.
- *Red lines*: Classes with red lines in the corner are fully defined in a different figure.

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.

`[elementName]`: In some cases, an element may contain a child of any class inheriting from an abstract base class. In this case, the name of the element is indicated by giving the abstract base class name in brackets, meaning that the actual name of the element depends on whichever subclass is used, with capitalization following the capitalization of the name in brackets.

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. This version of the Distributions package has two URIs, depending on which version of core SBML is being used:

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

"http://www.sbml.org/sbml/level3/version2/distrib/version1"

Note that the Distributions package may be used with both SBML Level 3 Version 1 and SBML Level 3 Version 2 documents, with the only semantic change between the two present in the **DistribBase** class. Note that if used with SBML Level 3 Version 1 the corresponding Distributions namespace for SBML Level 3 Version 1 must be used and similarly if SBML Level 3 Version 2 is being used, the Distributions namespace must be the one for SBML Level 3 Version 2.

In addition, SBML documents using a given package must indicate whether the package may be used to change the mathematical meaning of SBML Level 3 Core elements. 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`", as the **DrawFromDistribution** element overrides the core definition of a **FunctionDefinition**. Note that the value of this attribute must *always* be set to "`true`", even if the particular model does not contain any **DrawFromDistribution** elements.

The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 1 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">
```

The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 2 and this version of the Distributions package:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core" level="3" version="2"
      xmlns:distrib="http://www.sbml.org/sbml/level3/version2/distrib/version1"
      distrib:required="true">
```

3.3 Primitive data types

The Distributions package uses data types described in Section 3.1 of the SBML Level 3 Core specification, and adds the additional primitive types described below.

3.3.1 Type `ExternalRef`

The type `ExternalRef` is derived from `string` with the additional requirement that it be a valid URI. An `ExternalRef` is used in Distributions to point to ontologies such as ProbOnto (Swat et al., 2016) which contain defined distributions and parameters.

3.3.2 Type `UncertId`

The type `UncertId` is derived from `SId` (SBML Level 3 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 **DrawFromDistribution** 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.

3.3.3 Type `UncertIdRef`

Type `UncertIdRef` is used to reference different elements in different contexts. Inside a **FunctionDefinition**, an `UncertIdRef` may only reference an `UncertId` from a **DistribInput** from that same **FunctionDefinition**. Outside a

FunctionDefinition, an `UncertIdRef` may reference any element with an `SId` that has mathematical meaning: even elements from other packages, and not in SBML Level 3 Core. In the context of an SBML Level 3 Version 1 document, this still holds true. Even though SBML Level 3 Version 1 elements with `SIdRef` attributes cannot reference package elements, this does not preclude Distributions elements from doing so.

If a referenced `SId` is from a package that is not understood by the software reading the model, the meaning of the `UncertIdRef` is undefined. 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.

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 describing elements with information about their uncertainty. One common use case for this will be to provide the standard deviation for a value. Another may be describing a parameter's distribution, so that a better search can be performed in a parameter scan experiment.

These purposes are achieved by extending **FunctionDefinition** and ***SBase***, which in turn use the **Distribution** and **UncertStatistics** classes, modeled after UncertML. Several distributions and statistics are defined explicitly in this specification, but more can be defined by referencing the external ontology such as the ProbOnto ontology through the **ExternalDistribution** and **ExternalParameter** classes.

It is hoped that with this approach, modelers may use the **Distribution** classes defined in this specification with a reasonable expectation of various software packages recognizing and replicating those distributions. However, if another distribution is required, those distributions may still be encoded, even if this makes the model less exchangeable.

When a distribution is defined in a **FunctionDefinition**, it is sampled when it is 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**. When a distribution is defined elsewhere, that information may be used outside of the model, using whatever methodology is appropriate to answer the question being pursued.

3.5 The DistribBase class

The **DistribBase** class is an abstract base class which is the parent class for every class in this Distributions package. Its sole purpose is to encapsulate how this package handles the fact that in SBML Level 3 Version 2, the attributes `id` and `name` were added to ***SBase***. It defines attributes 'distrib:id' and 'distrib:name' as optional, so that they can be used in SBML Level 3 Version 1 documents where those attributes would otherwise not exist. However, in SBML Level 3 Version 2 documents, 'distrib:id' and 'distrib:name' may not be used: the core attributes `id` and `name` should be used instead. In this way, every Distributions element will always have exactly one id and exactly one name.

The meaning of these attributes is identical, regardless of the level/version of the document in which they appear. When a subclass makes one of the attributes required (as the `id` of the **DistribInput** class), this means that it must have the 'distrib:id' attribute in a SBML Level 3 Version 1 document, or an `id` attribute in a SBML Level 3 Version 2 document.

The `id` attribute is of type `SId`, and must be unique among other ids in the `SId` namespace in the parent **Model**, and has no mathematical meaning, unless stated otherwise in the definition of that object. The `name` attribute is of type `string`, and is provided to allow the user to define a human-readable label for the object, and has no uniqueness restrictions.

3.6 The extended FunctionDefinition class

To model random processes, this package extends the **FunctionDefinition** class as can be seen in the UML representation in Figure 2. The redefined **FunctionDefinition** optionally contains a single `drawFromDistribution` child.

The **FunctionDefinition** class may or may not still contain the MathML block containing the standard SBML function definition if used in an SBML Level 3 Version 2 document, but it must contain this MathML block if used in an SBML Level 3 Version 1 document, because that element is required in that level and version. If present, the MathML should be ignored, but may ensure a degree of backwards compatibility for SBML readers and validators that do not understand the *distrib* package.

As outlined above, the **FunctionDefinition** class is extended to contain a **DrawFromDistribution** child, which overrides the <lambda> element of the **Math** child present in the **FunctionDefinition**. Software that does not support *distrib* could potentially invoke the original **Math** of the **FunctionDefinition** (see Section 3.22) instead.

In the **Model**, an extended **FunctionDefinition** may be used in any MathML elsewhere in the document 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).

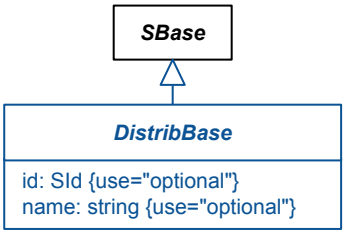

Figure 1: *The definition of the **DistribBase** class. The `id` and `name` attributes defined are optional, but must not be used in SBML Level 3 Version 2 documents.*

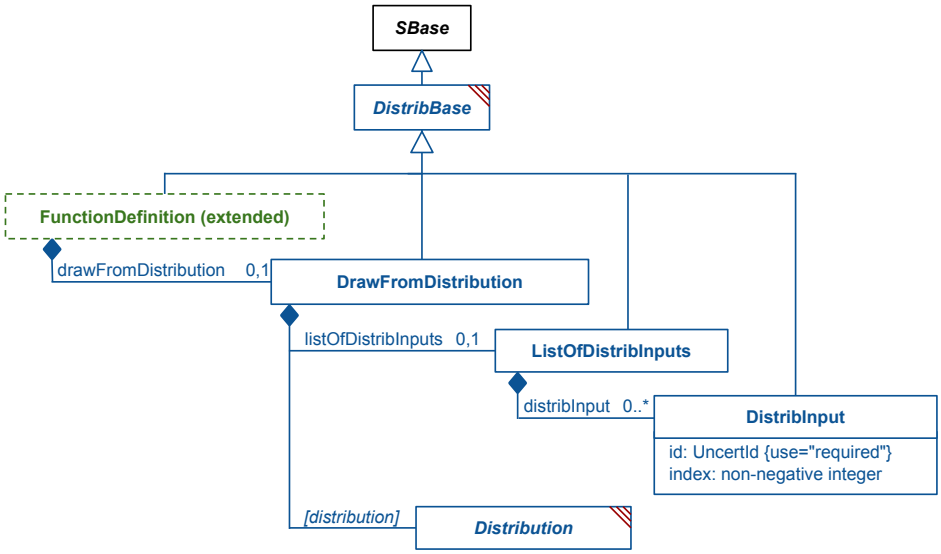

Figure 2: *The definition of the extended **FunctionDefinition** class, plus the **DrawFromDistribution**, **ListOfDistribInputs**, and **DistribInput** classes. The **Distribution** class is used here, but defined later. A **DrawFromDistribution** element must have exactly one **Distribution** child. Together, these classes provide a way to transform a **FunctionDefinition** to sample from a distribution.*

3.7 The DrawFromDistribution class

As illustrated in Figure 2 on the previous page, the **DrawFromDistribution** class may have a **ListOfDistribInputs** child, which may in turn contain any number of **DistribInput** 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**. The order of arguments is determined by the `index` attribute: the first argument (if any) must have an index of "`0`", the second of "`1`", etc. If no **ListOfDistribInputs** is provided, or if one is provided with no **DistribInput** children, the function contains no arguments.

The **DrawFromDistribution** must also have a **Distribution** child, representing the distribution from which to draw a value. Because **Distribution** is an abstract class, the name of the element will be the name of the class of the particular distribution being used, with its first letter made lower case (so, "`normalDistribution`" for **NormalDistribution**, "`studentTDistribution`" for **StudentTDistribution**, etc.). Within this **Distribution**, any `UncertIdRef` must reference the `UncertId` of a **DistribInput** for the **FunctionDefinition** in which it appears.

3.7.1 Attributes inherited from SBase

A **DrawFromDistribution** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of a **DrawFromDistribution** has no mathematical meaning.

3.8 The ListOfDistribInputs class

The **ListOfDistribInputs** class, like other **ListOf**_____ classes in SBML Level 3 Core, is a container for zero or more **DistribInput** objects. If empty, it simply means that no child **DistribInput** objects are defined for its parent, and is equivalent to not including the **ListOfDistribInputs** object at all. This situation defines an extended **FunctionDefinition** with zero arguments, and might be useful if the list is annotated with the reason why it is empty, for example.

3.9 The DistribInput class

The **DistribInput** class mimics the `bvar` elements of MathML lambda functions. It must have an `id` attribute of type `UncertId` and an `index` attribute of type `non-negative integer`.

Each **DistribInput** element represents an argument to the function, and serves as a local identifier, referenced only by the **Distribution** class. See the examples in Section 3.21 for more details.

If the **Lambda** child of the **FunctionDefinition** is present, it must have the same number of `bvar` children as the **DrawFromDistribution** has **DistribInput** 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 **DistribInput** ids are defined as being local to the **DrawFromDistribution** object.

Each `index` attribute on a **DistribInput** within a **ListOfDistribInputs** element must have a unique value, numbered consecutively from "`0`": if one **DistribInput** is present, its `index` value must be "`0`"; if there are two, they must have `index` values of "`0`" and "`1`", etc. Where the **FunctionDefinition** also contains a **Math** child the values of the `index` relates the **DistribInput** to the the <bvar> element in the corresponding position.

3.9.1 Attributes inherited from SBase

A **DistribInput** always inherits the optional `metaid` and `sboTerm` attributes, and inherits the `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of a **DistribInput** has no external mathematical meaning, is required, and is of type `UncertId`, as described above. Note that in an SBML Level 3 Version 2 document, a **DistribInput** must either have an `id` in the core namespace, or an `id` in the Distributions namespace (but not both).

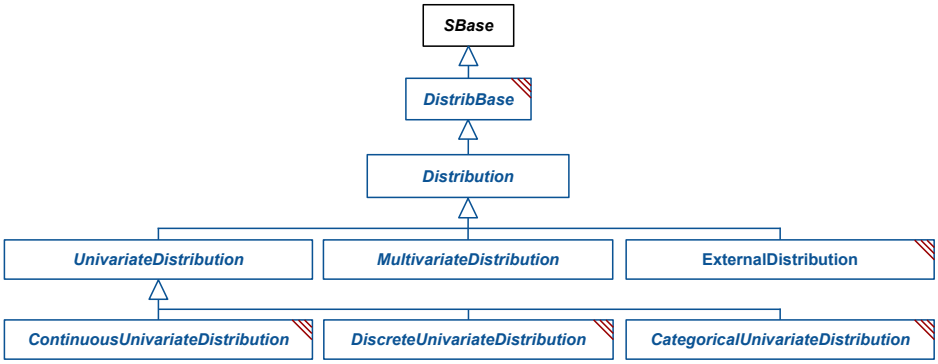

Figure 3: *The definition of the abstract* ***Distribution*** *class, plus the abstract* ***UnivariateDistribution*** *and* ***MultivariateDistribution*** *classes. Also shown are the* ***ContinuousUnivariateDistribution***, ***DiscreteUnivariateDistribution***, ***CategoricalUnivariateDistribution***, *and* ***ExternalDistribution*** *classes, defined later. Note that the* ***MultivariateDistribution*** *class is only defined here, and, as such, has no concrete classes that derive from it at this time in this version of the Distributions specification. If further discussion reveals a need for them, it is shown here as an example of where it would be defined, but may disappear from a future version of the spec if no derived class for it is ever defined.*

3.10 The Distribution class

The **Distribution** class is the abstract class from which all Distributions distributions are derived. They are organized here in much the same way they were in UncertML, by whether they are univariate or multivariate, and whether they are continuous, discrete, or categorical. In addition, the **ExternalDistribution** inherits from **Distribution**, as a 'generic' distribution definition class that allows the user to define any distribution in an external ontology such as ProbOnto.

When a **Distribution** 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 a **DistribInput** in the same function.

? Lucian: NOTE! There are too many distributions defined in this version of the Distributions specification! This is done on purpose. The goal of this draft of the specification is not to be normative, but to lay out the details of everything that might possibly be wanted, with the intent of reducing the number to what is actually going to be used and implemented in a subsequent draft, based on user and developer feedback. As a start, then, every single univariate distribution defined in UncertML, plus the **RayleighDistribution** (defined in Frank's annotation scheme), is presented. The idea is that developers first implement what is useful for them, then we trim the list to only what at least one developer implemented, then we give developers the newly-winnowed list and say 'this is what other people want' and ask them to fill in the gaps. It is predicted that the final number of pre-defined distributions will number about a dozen or so.

In this draft of the Distributions specification, no mixed distributions and no multivariate distributions are presented, as the author has not seen any call for these distributions specifically, and believes that the generic **ExternalDistribution** distribution could cover those cases on an as-needed basis. If this turns out to not be the case, those distributions will be added to a subsequent version of this specification. The use of the Arrays package would be required for any multivariate distribution.

3.10.1 Attributes inherited from SBase

A **Distribution** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of a **Distribution** has no mathematical meaning.

3.11 The UnivariateDistribution class

The **UnivariateDistribution** class is an abstract class that derives from the **Distribution** abstract class, and which has three derived classes itself: **ContinuousUnivariateDistribution**, **DiscreteUnivariateDistribution**, and **CategoricalUnivariateDistribution**. It is provided as a bookkeeping class to distinguish it from other types of distributions.

3.12 The MultivariateDistribution class

The **MultivariateDistribution** class is an abstract class with no derived classes in the current specification, but some could be added in the future. Most likely, it will be removed from the final version of the spec.

3.13 The ContinuousUnivariateDistribution class

The abstract **ContinuousUnivariateDistribution** class is the base class for a wide variety of distributions, all of which describe a potentially-bounded continuous range of probabilities. Many of the most commonly-used distributions such as the **NormalDistribution** and the **UniformDistribution** fall into this category.

All **ContinuousUnivariateDistribution** elements may have two optional children: "`lowerTruncationBound`" and "`upperTruncationBound`", both of the class **UncertBound** (defined below). Either element, if present, limit the range of possible sampled values from the distribution. The "`lowerTruncationBound`" defines the lowest value (inclusive or not, as defined by that element's `inclusive` attribute) that can be sampled, and the "`upperTruncationBound`" defines the highest. If both children are present, the "`lowerTruncationBound`" must either be lower than the "`upperTruncationBound`", or they may be equal, if both bounds are set `inclusive`="`true`". Similarly, some distributions are themselves naturally bound (some may, for example, only return values greater than zero). In those cases, the natural lower bound of the distribution must be either be lower than the "`upperTruncationBound`", or be equal to it if the natural lower bound is inclusive, and if the "`upperTruncationBound`" is set `inclusive`="`true`". Similarly, the natural upper bound of the distribution must either be higher than the "`lowerTruncationBound`", or it may be equal to it if the natural upper bound is inclusive and if the "`lowerTruncationBound`" is set `inclusive`="`true`". It may be impossible to determine this from a static analysis of the model, as either or both bound's values may depend on other dynamic variables. If a simulator encounters this situation, the sampled value and the behavior of the simulator are undefined.

If bounded, the cumulative probability that would have been assigned to the region outside the bound is re-assigned proportionally to the rest of the distribution. It should be noted that while discarding any value obtained from the non-truncated version of the distribution and re-sampling is indeed one method that could be used to accomplish this, the efficiency of that algorithm decreases with the width of the allowed window, and indeed is technically zero (and would take an infinite amount of time to complete) should the bounds be equal to one another. Taking any samples obtained outside the bound window and instead returning the boundary value itself is incorrect, and will not result in a proper draw from the defined distribution.

The distributions of this type allowed in this version of the specification are defined in Figure 4 on the next page and Figure 5 on page 19. A full list of all of the distributions is provided in Appendix A on page 45.

3.14 The DiscreteUnivariateDistribution class

The abstract **DiscreteUnivariateDistribution** class is the base class for a wide variety of distributions, all of which describe a potentially-bounded range of probabilities of discrete values. The most commonly-used distributions in this class is probably the **PoissonDistribution**. Distributions that always return integers fall in this category, which often involve events happening at particular frequencies.

All **DiscreteUnivariateDistribution** elements (like **ContinuousUnivariateDistribution** elements) may have two optional children: "`lowerTruncationBound`" and "`upperTruncationBound`", both of the class **UncertBound** (defined below). Either element, if present, limit the range of possible sampled values from the distribution. The "`lowerTruncationBound`" defines the value below which no sampling may take place (inclusive or not, as defined by that element's `inclusive` attribute), and the "`upperTruncationBound`" defines the value above which no sampling

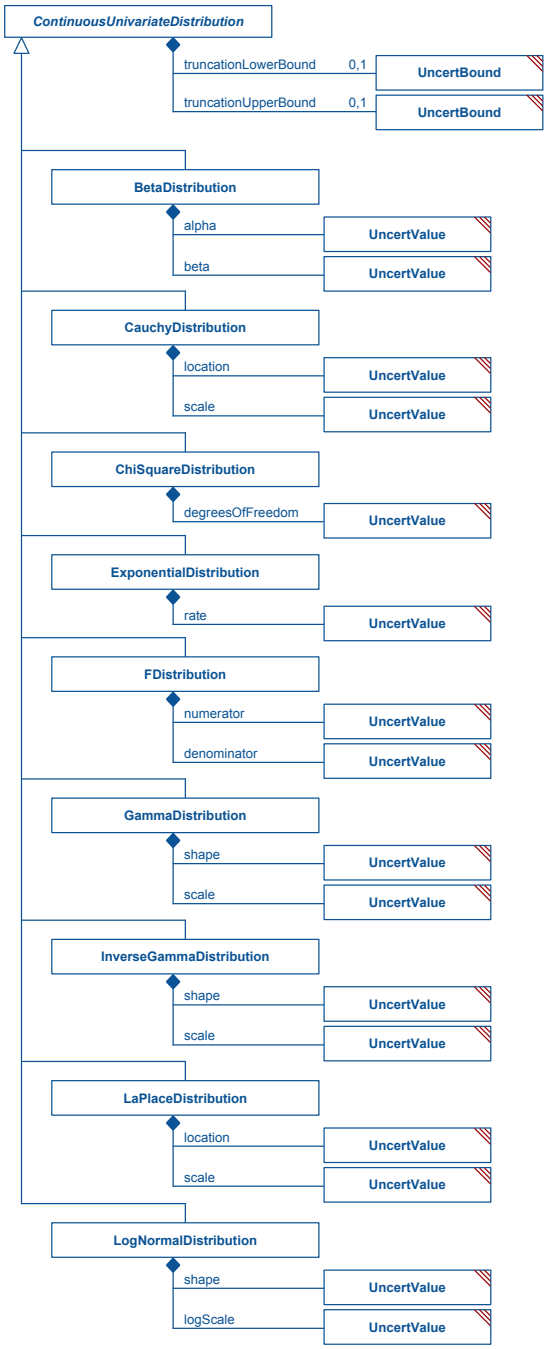

Figure 4: *The definition of the* ***ContinuousUnivariateDistribution*** *abstract class, and its* ***BetaDistribution***, ***CauchyDistribution***, ***ChiSquareDistribution***, ***ExponentialDistribution***, ***FDistribution***, ***GammaDistribution***, ***InverseGammaDistribution***, ***LaPlaceDistribution***, *and* ***LogNormalDistribution*** *children. All may have lower and upper* ***UncertBound*** *children, and each has one or more other parameters encoded as* ***UncertValue*** *children (both defined below).*

may take place. These bounds may fall between the possible discrete values being returned: as an example, for a distribution that returned an integer in the series [0, 1, 2, ...], if it was given a "`lowerTruncationBound`" of 1.5, the lowest value it could return would be 2. In this case, the value of the `inclusive` attribute on the **UncertBound** would be immaterial, as '1.5' could never be returned.

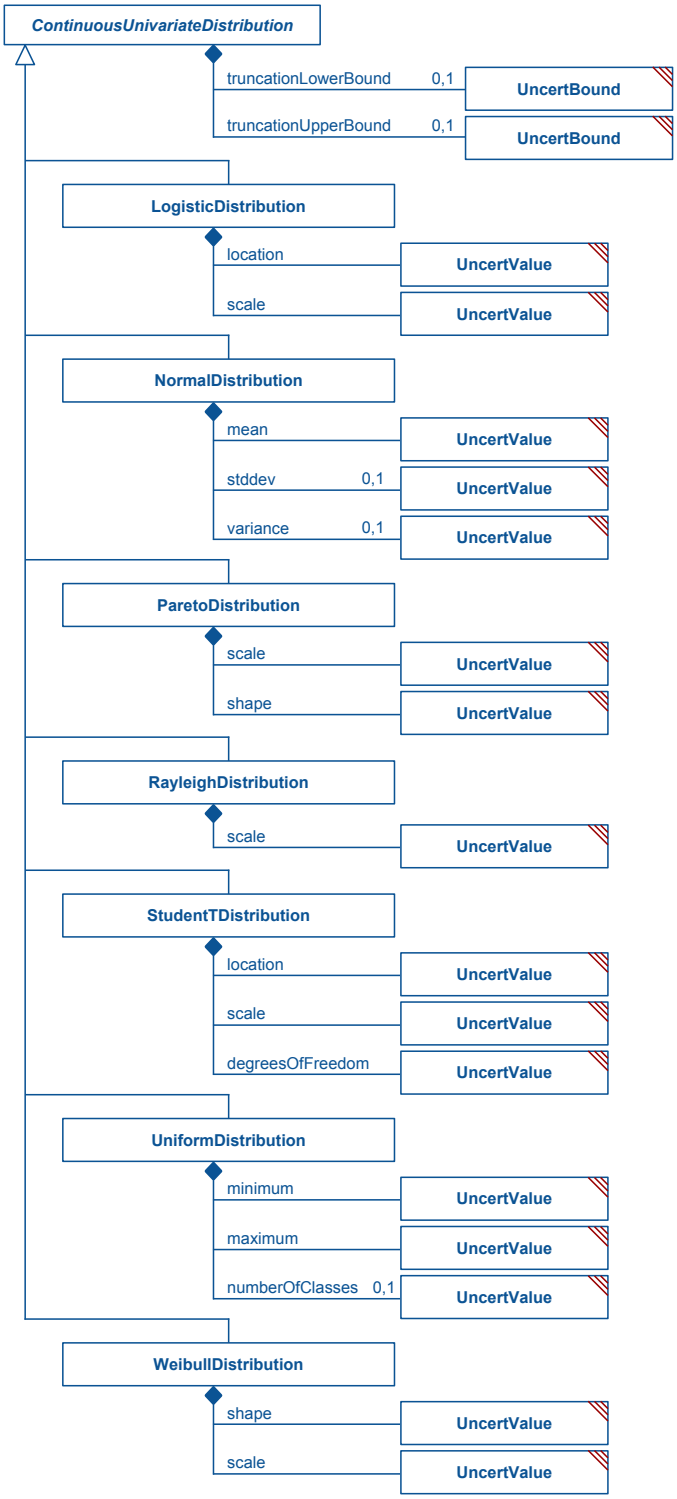

Figure 5: *The definition of the* ***ContinuousUnivariateDistribution*** *abstract class, and its* ***LogisticDistribution***, ***NormalDistribution***, ***ParetoDistribution***, ***RayleighDistribution***, ***StudentTDistribution***, ***UniformDistribution***, *and* ***WeibullDistribution*** *children. All may have lower and upper* ***UncertBound*** *children, and each has one or more other parameters encoded as* ***UncertValue*** *children (both defined below).*

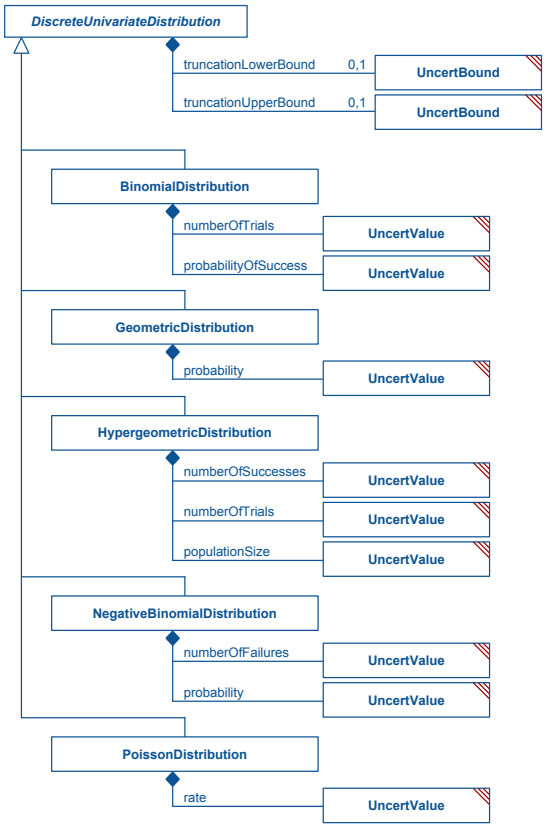

Figure 6: *The definition of the* ***DiscreteUnivariateDistribution*** *abstract class, and its* ***BinomialDistribution***, ***GeometricDistribution***, ***HypergeometricDistribution***, ***NegativeBinomialDistribution***, *and* ***PoissonDistribution*** *children. All may have lower and upper* ***UncertBound*** *children, and each has one or more other parameters encoded as* ***UncertValue*** *children (both defined below).*

As in the case of the **ContinuousUnivariateDistribution** bounds, if both bounds are present, the "`lowerTruncationBound`" must either be lower than the "`upperTruncationBound`", or they may be equal, if both bounds are set `inclusive`="`true`". Similarly, the discrete distributions are themselves often naturally bound (some may, for example, only return values greater than zero). In those cases, the natural lower bound of the distribution must be either be lower than the "`upperTruncationBound`", or it may be equal to it if the natural lower bound is inclusive, and if the "`upperTruncationBound`" is set `inclusive`="`true`". Similarly, the natural upper bound of the distribution must either be higher than the "`lowerTruncationBound`", or it may be equal to it if the natural upper bound is inclusive and if the "`lowerTruncationBound`" is set `inclusive`="`true`". In addition, if both bounds are defined, they must define a span within which at least one possible sampled discrete value may be found. For a distribution that returns integers, for example, one may not define a lower bound of 1.5 and an upper bound of 1.8, as no integer lies within that range. It may be impossible to determine if any of these rules are violated from a static analysis of the model, as either or both bound's values may depend on other dynamic variables. If a simulator encounters this situation, the sampled value and the behavior of the simulator are undefined.

If bounded, the cumulative probability that would have been assigned to the values outside the bound is re-assigned proportionally to the rest of the distribution. It should be noted that while discarding any value obtained from the non-truncated version of the distribution and re-sampling is indeed one method that could be used to accomplish this, the efficiency of that algorithm decreases with the width of the allowed window, and indeed is technically zero (and could take an infinite amount of time to complete) should the bounds allow only a single discrete value. Taking any samples obtained outside the bound window and instead returning the boundary value itself is incorrect, and will not result in a proper draw from the defined distribution.

The distributions of this type allowed in this version of the specification are defined in Figure 6 on the preceding page. A full list of all of the distributions is provided in Appendix A on page 45.

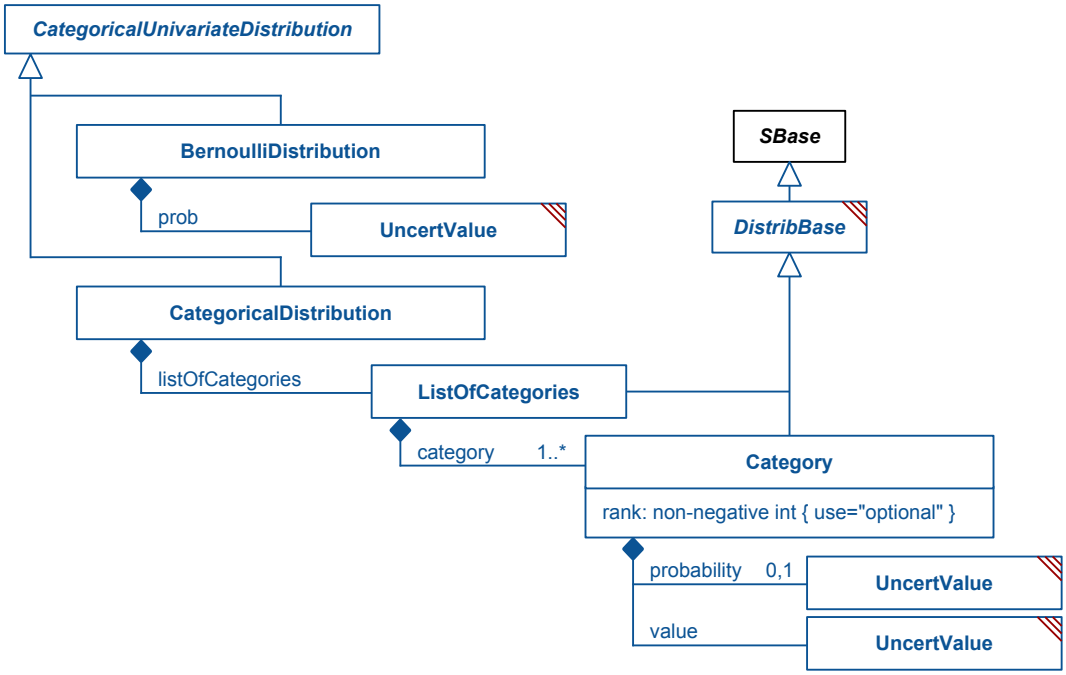

Figure 7: *The definition of the* ***CategoricalUnivariateDistribution*** *abstract class, plus the* ***BernoulliDistribution***, ***CategoricalDistribution***, ***ListOfCategories***, *and* ***Category*** *classes.*

3.15 The CategoricalUnivariateDistribution class

The **CategoricalUnivariateDistribution** abstract class includes distributions where the various possible sampled values are each explicitly listed, along with the probability for that sampled value. The sum of these probabilities must therefore equal 1.0, in order to be valid. This type of distribution class is used for things such as weighted die rolls, or other situations where particular values are obtained at arbitrary probabilities.

Because each possible sampled value is explicitly listed in an **CategoricalUnivariateDistribution**, it does not have the optional **UncertBound** values that the other univariate distributions do: if a particular value is not allowed, it is simply dropped from the list of options, and the probabilities of the other values are scaled accordingly.

3.16 Distribution Elements

Because the list of distributions is extensive, all of them are provided at the end of this document in Appendix A on page 45. The elements they use are defined below.

3.16.1 The UncertValue class

The **UncertValue** class provides two optional attributes, exactly one of which must be defined. The `value` attribute (of type `double`) is used when the **UncertValue** represents a particular number, and the `var` attribute (of type `UncertIdRef`) is used when the **UncertValue** represents a referenced element with mathematical meaning. In the context of a **FunctionDefinition**, this can only reference a **DistribInput**, as no `SId` from the **Model** may be referenced from within a **FunctionDefinition**. In other contexts, it may reference the `SId` of any element with mathematical meaning; see Section 3.3.3 on page 12.

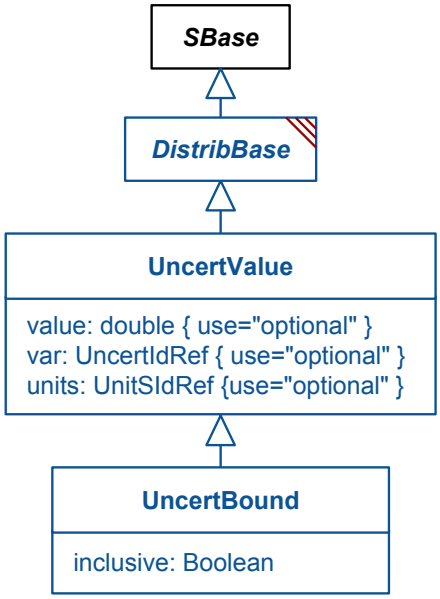

Figure 8: *The definition of the* ***UncertValue*** *and* ***UncertBound*** *classes. These classes define a way to reference either a value or an element with mathematical meaning. The* ***UncertBound*** *class additionally defines whether it is considered to be inclusive or not.*

The optional `units` attribute may be used to indicate the units of the `val` attribute. As such, it may only be defined if the **UncertValue** has a defined `val` attribute, and not if it has a defined `var` attribute. (In the latter case, the units may be derived from the referenced element.)

Any given **UncertValue** in a **Distribution** will have an element name specific to the parameter it represents within that **Distribution**. So, for example, a **NormalDistribution** will have one child **UncertValue** with the name "`<mean>`", and might have another **UncertValue** child with the name "`<stddev>`". All these parameters are defined as the same class for simplicity, since all of them merely need a way to reference a value.

3.16.2 Attributes inherited from SBase

An **UncertValue** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. Outside of a **FunctionDefinition**, the `id` of a **UncertValue** takes the mathematical value of its `value` attribute if that attribute is defined, and the mathematical value of the corresponding `var` if that attribute is defined. This meaning may be used in other contexts, but that meaning may not be set directly by any other SBML element of any level, version, or package. If setting the value is desired, the `var` attribute should be used, and that referenced element set as per normal SBML procedures. The meaning is provided mostly to allow access to the `val` attribute, which otherwise would be undiscoverable to any other SBML element. Do note, however, that the scope of the **UncertValue** `id` is limited to its **FunctionDefinition** parent in that context. In other contexts, its `id` has the scope of its parent **Model**.

3.16.3 The UncertBound class

The **UncertBound** class inherits from **UncertValue** and adds a single required Boolean attribute `inclusive`. This attribute indicates whether the value the bound represents is to be included in that range ("`true`") or not ("`false`"). This allows the creation of either 'open' or 'closed' boundaries of the ranges it is used to define.

3.17 The ExternalDistribution class

The **ExternalDistribution** class is provided to allow a modeler to encode a distribution not otherwise explicitly handled by this specification. Because the range of possibilities is so vast, the modeler should not normally expect

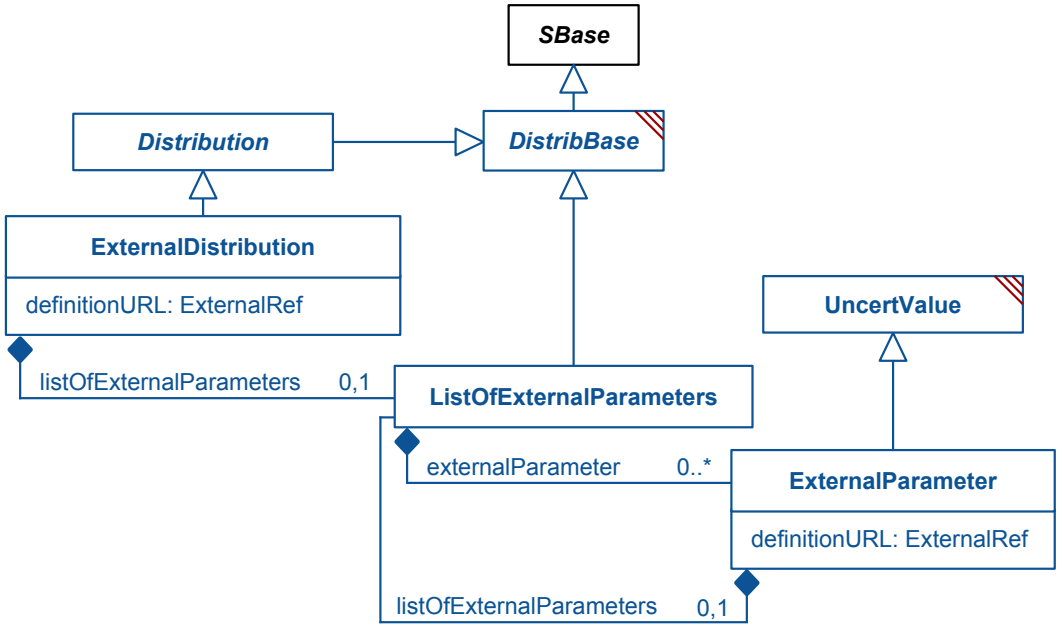

Figure 9: *The definition of the* ***ExternalDistribution****,* ***ExternalParameter****, and* ***ListOfExternalParameters*** *classes. These classes define a way to define a distribution with reference to an external database or ontology of distribution definitions.*

any given SBML simulator or other software to be able to properly manipulate this distribution, but particular software tools may implement support for certain distributions they know their own software's users may require.

The required attribute `definitionURL`, of type `ExternalRef`, must be a URI that defines a valid distribution. It is strongly recommended that modelers use distributions from ProbOnto (`http://probonto.org/`), as consistently referencing a single ontology will improve exchangeability, at least slightly. The referenced distribution is then the distribution defined by this **ExternalDistribution**, along with any parameterization provided by the children **ExternalParameter** elements.

Some referenced distributions are multivarite, meaning they define correlated distributions for two or more parameters. It is impossible with SBML Level 3 Core to define a **FunctionDefinition** that returns a vector, and similarly no `SId` in SBML Level 3 Core can be used to represent a vector. If this is desired, then, the Arrays package must be used in concert with the **ExternalDistribution** to cooperatively set up a model with a **FunctionDefinition** that can use an array as input and/or as output.

The **ExternalDistribution** defines an optional child **ListOfExternalParameters**, which can be used to parameterize the defined distribution.

3.18 The ListOfExternalParameters class

The **ListOfExternalParameters** class, like other **ListOf**____ classes in SBML Level 3 Core, is a container for zero or more **ExternalParameter** objects. If empty, it simply means that no child **ExternalParameter** objects are defined for its parent, and is equivalent to not including the **ListOfExternalParameters** object at all. This situation might be useful if the list is annotated with the reason why it is empty, for example.

3.19 The ExternalParameter class

The **ExternalParameter** class, like the **ExternalDistribution**, is provided to allow a modeler to encode externally-provided parameters not otherwise explicitly handled by this specification. Again, the range of possibilities is vast, so modelers should ensure that the tool they wish to use encodes support for any **ExternalParameter** they define.

The **ExternalParameter** inherits from **UncertValue**, and adds the required attribute `definitionURL`, which is of type `ExternalRef`, and an optional child **ListOfExternalParameters**. The `definitionURL` must be a URI that defines a valid distribution-related parameter. It is again strongly recommended that modelers use distribution parameters from ProbOnto (`http://probonto.org/`), as consistently referencing a single ontology will improve exchangeability.

The child **ListOfExternalParameters** is provided because some parameters may themselves need further parameterization. For example, a mixture distribution defined as an **ExternalDistribution** would contain as child **ExternalParameter** objects those other base distributions that were mixed in the overall distribution. Those base distributions would need to define their own parameterization, which could be accomplished here with child **ExternalParameter** objects. Similarly, ranges or categories might also need to be further defined with reference to child **ExternalParameter** objects that would be considered to 'belong' to the parent **ExternalParameter**.

The referenced parameter is then the parameter defined by this **ExternalParameter**, along with any further parameterization provided by its own children **ExternalParameter** elements.

Some referenced distributions are multivarite, meaning they define correlated distributions for two or more parameters. If an input **ExternalParameter** is required for a distribution, one must use the inherited `var` attribute to define the parameter's value rather than the `value` attribute, and SBML must contain some way to define that referenced `var` as an array (such as the Arrays package).

3.20 Discrete vs. continuous sampling

The `SId`s of **FunctionDefinition** elements can be used in SBML Level 3 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.

3.21 Examples using the extended FunctionDefinition

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

3.21.1 Defining and using a NormalDistribution

In the following example, a **FunctionDefinition** is extended to define a draw from a **NormalDistribution**:

```
...
      <functionDefinition id="normal">
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="mu" distrib:index="0"/>
```

```
            <distrib:distribInput distrib:id="sdev" distrib:index="1"/>
          </distrib:listOfDistribInputs>
          <distrib:normalDistribution>
            <distrib:mean distrib:var="mu"/>
            <distrib:stddev distrib:var="sdev"/>
          </distrib:normalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
...
```

Here, the **DistribInput** children of **DrawFromDistribution** define the local `UncertId`s "`mu`" and "`sdev`", which are then used by the **Distribution** as the `mean` and `stddev` of a normal distribution, as defined by UncertML. (The example shown is from an SBML Level 3 Version 2 document, where the `<math>` child of a **FunctionDefinition** is optional.) This function could then be used anywhere the **FunctionDefinition** id "`normal`" can be used, as for example in an **InitialAssignment**:

```
...
    <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>
...
```

This use would apply a draw from a normal distribution with mean "`z`" and standard deviation "`10`" to the SBML element "`y`".

3.21.2 Defining a truncated NormalDistribution

In the following example, a **FunctionDefinition** is extended to define a draw from a truncated **NormalDistribution**:

```
...
      <functionDefinition id="normal_trunc">
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="mean" distrib:index="0"/>
            <distrib:distribInput distrib:id="stddev" distrib:index="1"/>
            <distrib:distribInput distrib:id="lowerBound" distrib:index="2"/>
            <distrib:distribInput distrib:id="upperBound" distrib:index="3"/>
          </distrib:listOfDistribInputs>
          <distrib:normalDistribution>
            <distrib:truncationLowerBound distrib:var="lowerBound" distrib:inclusive="true"/>
            <distrib:truncationUpperBound distrib:var="upperBound" distrib:inclusive="true"/>
            <distrib:mean distrib:var="mean"/>
            <distrib:stddev distrib:var="stddev"/>
          </distrib:normalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
...
```

Here, the **DistribInput** children of **DrawFromDistribution** define four arguments: `mean`, `stddev`, `lowerBound`, and `upperBound`. These arguments are then used as the values of the `var` attributes of the corresponding **UncertValue** children of the **NormalDistribution**. An example use in an **InitialAssignment** might look like:

```
...
      <initialAssignment symbol="y">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
          <apply>
            <ci> normal_trunc </ci>
            <ci> z </ci>
            <cn type="integer"> 10 </cn>
            <apply>
              <minus/>
              <ci> z </ci>
              <cn type="integer"> 2 </cn>
            </apply>
            <apply>
              <plus/>
              <ci> z </ci>
              <cn type="integer"> 2 </cn>
            </apply>
          </apply>
        </math>
      </initialAssignment>
...
```

This use would apply a draw from a normal distribution with mean z, standard deviation 10, lower bound $z-2$ and upper bound $z+2$ (inclusive) to the SBML element "y".

3.21.3 Defining a 'die roll' PMF with a CategoricalDistribution

In the following example, a **FunctionDefinition** is extended to define a draw from a **CategoricalDistribution** PMF:

```
...
      <functionDefinition id="die_roll">
        <distrib:drawFromDistribution>
          <distrib:categoricalDistribution>
            <distrib:listOfCategories>
              <distrib:category>
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="1"/>
              </distrib:category>
              <distrib:category>
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="2"/>
              </distrib:category>
              <distrib:category>
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="3"/>
              </distrib:category>
              <distrib:category>
                <distrib:value distrib:value="4"/>
              </distrib:category>
            </distrib:listOfCategories>
          </distrib:categoricalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
...
```

No inputs are provided. Three of the four **Category** children of the **CategoricalDistribution** all have equal values for their `probability` children, with the fourth not having its probability set, meaning that it has a probability such that the sum of all the probabilities among all categories is 1.0. Each child **Category** is therefore equally likely to be chosen, resulting in this function returning "1", "2", "3", or "4", each with equal probability.

In a modeling context where it is not a die being rolled, but different patients which we are sampling, the following function definition uses the same structure:

```
...
      <functionDefinition id="pick_patient">
```

```
        <distrib:drawFromDistribution>
          <distrib:categoricalDistribution>
            <distrib:listOfCategories>
              <distrib:category id="patient1">
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="1.21"/>
              </distrib:category>
              <distrib:category id="patient2">
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="2.24"/>
              </distrib:category>
              <distrib:category id="patient3">
                <distrib:probability distrib:value="0.25"/>
                <distrib:value distrib:value="-0.6"/>
              </distrib:category>
              <distrib:category id="patient4">
                <distrib:value distrib:value="1.82"/>
              </distrib:category>
            </distrib:listOfCategories>
          </distrib:categoricalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
...
```

Here, we have four patients, each of which had a different value. When sampled (such as in an initial assignment), one would be chosen at random to populate a value for the simulation. In a more complicated context where several different values were sampled from a single patient, the 'Arrays' package would need to be used, so that all the values from a single patient could be returned as a group, and then assigned to their appropriate targets. In that case, various arrays could be set up with the first value in all arrays corresponding to the first patient, the second value with the second patient, etc. A **DrawFromDistribution** could then be created to return a patient index at random, and that index used to assign values in the simulation.

3.21.4 Defining an external distribution

In the following example, a **FunctionDefinition** is extended to define a draw from an externally-defined distribution (in this case, an alternative parameterization of the **ExponentialDistribution**):

```
...
      <functionDefinition id="Exponential2">
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput id="beta" distrib:index="0"/>
          </distrib:listOfDistribInputs>
          <distrib:externalDistribution name="Exponential␣2"
              distrib:definitionURL="http://www.probonto.org/ontology#PROB_k0000353">
            <distrib:listOfExternalParameters>
              <distrib:externalParameter name="Beta" distrib:var="beta"
                distrib:definitionURL="http://www.probonto.org/ontology#PROB_k0000362"/>
            </distrib:listOfExternalParameters>
          </distrib:externalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
...
```

As defined in the ProbOnto ontology, this parameterization of the exponential distribution takes the form $1/\beta e^{-x/\beta}$ instead of the form $\lambda e^{-x\lambda}$, used by the **ExponentialDistribution**.

3.22 Equivalence with Fallback Function

The MathML definition directly contained by the **functionDefinition** is not used, but is required by SBML Level 3 Version 1. (It is optional in SBML Level 3 Version 2.) To ensure the continued validity of the model, the following

rules must be followed:

- 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 "`s`" instead of "`avg`" and "`sdev`"); their equivalence is based on order, not string matching.

```
<functionDefinition id="normal">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <lambda>
      <bvar>
        <ci> mean </ci>
      </bvar>
      <bvar>
        <ci> sdev </ci>
      </bvar>
      <ci> mean </ci>
    </lambda>
  </math>
  <distrib:drawFromDistribution>
    <distrib:listOfDistribInputs>
      <distrib:distribInput distrib:id="mean" distrib:index="0"/>
      <distrib:distribInput distrib:id="s" distrib:index="1"/>
    </distrib:listOfDistribInputs>
    <distrib:normalDistribution>
      <distrib:mean distrib:var="mean"/>
      <distrib:stddev distrib:var="s"/>
    </distrib:normalDistribution>
  </distrib:drawFromDistribution>
</functionDefinition>
```

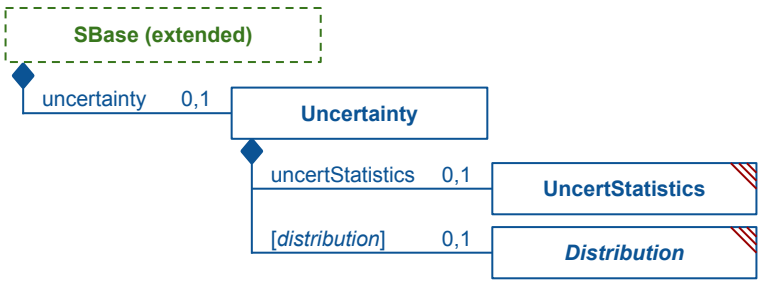

Figure 10: *The definition of the extended* ***SBase*** *class to include a new optional* ***Uncertainty*** *child, which in turn has optional* ***UncertStatistics*** *and* ***Distribution*** *children. Intended for use with any element with mathematical meaning, or with a* ***Math*** *child.*

3.23 The extended *SBase* class

As can be seen in Figure 10, the SBML base class ***SBase*** is extended to include an optional **Uncertainty** child, which in turn contains an optional **Distribution** child, and an optional **UncertStatistics** child, either or both of which may

be used to include information about the uncertainty of the parent element. In SBML Level 3 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**). These are 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: for example, the **FluxBound** class from the Flux Balance Constraints package has mathematical meaning, and could 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, and 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, so only one should be extended with a child **Distribution** and/or **UncertStatistics**. 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.

It is important to note that the uncertainty described either by the **Distribution** or the **UncertStatistics** elements are defined as being the uncertainty at the moment the element's mathematical meaning is calculated, and does not describe the uncertainty of how that element changes over time. For a **Species**, **Parameter**, **Compartment**, and **SpeciesReference**, this means that it is the uncertainty of their initial values, and does not describe the uncertainty in how those values evolve in time. The reason for this is that other SBML constructs all describe how (or if) the values change in time, and it is those other constructs that should be used to describe a symbol's time-based uncertainty. For example, a **Species** whose initial value had uncertainty due to instrument precision could have an **Uncertainty** child describing this. A **Species** whose value was known to change over time due to unknown processes, but which had a known average and standard deviation could be given an **AssignmentRule** that set that **Species** amount to the known average, and the **AssignmentRule** itself could be given an **UncertStatistics** child describing the standard deviation of the variability.

3.24 The Uncertainty class

The **Uncertainty** class has two optional children: an **UncertStatistics** child and a **Distribution** child. Either or both may be used, depending on the information about the parent that the modeler wishes to store in this object. If neither is present, this means that no information about the uncertainty of the object is provided by this package. The **Uncertainty** may be annotated to provide more information about why this is.

Note that the described uncertainty for elements that change in value over time apply only to the element's uncertainty at a snapshot in time, and not the uncertainty in how it changes in time. For typical simulations, this means the element's initial condition. Note too that the description of the uncertainty of a **Species** should describe the uncertainty of its `amount`, not the uncertainty of its `concentration`. The 'primary' mathematical meaning of a **Species** in SBML is always the amount; the concentration may be used, but is considered to be derived.

3.24.1 Attributes inherited from SBase

An **Uncertainty** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of an **Uncertainty** has no mathematical meaning.

3.25 The UncertStatistics class

The **UncertStatistics** class is a collection of zero or more statistical measures related to the uncertainty of the parent SBML element. It contains three types of children: **UncertValue** children and **UncertStatisticSpan** children, distinguished from each other by the element name of that child (listed below), and a **ListOfExternalParameters**

child, which contains zero or more **ExternalParameter** objects.

The possible **UncertValue** children are listed below. Each is defined by its element name; the mean would be defined as <mean>, the standard deviation as <standardDeviation>, etc.

All the definitions below are from an archived copy of the definitions at http://uncertml.org/.

- coefficientOfVariation: For a random variable with mean μ and strictly positive standard deviation σ, the coefficient of variation is defined as the ratio $\frac{\sigma}{|\mu|}$. One benefit of using the coefficient of variation rather than the standard deviation is that it is unitless.
- kurtosis: The kurtosis of a distribution is a measure of how peaked the distribution is. The kurtosis is defined as $\frac{\mu_4}{\sigma^4}$ where μ_4 is the fourth centred moment of the distribution and σ is its standard deviation.
- mean: The arithmetic mean (typically just the mean) is what is commonly called the average. It is defined as $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ where x_i represents with ith observation of the quantity x in the sample set of size n. It is related to the expected value of a random variable, $\mu = E[X]$ in that the population mean, μ, which is the average of all quantities in the population and is typically not known, is replaced by its estimator, the sample mean $\bar{x}$. Note that this statistic does not deal with issues of sample size, rather the mean is taken to refer to the population mean.
- median: The median is described as the numeric value separating the higher half of a sample (or population) from the lower half. The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one. If there is an even number of observations, then there is no single middle value, then the average of the two middle values is used. The median is also the 0.5 quantile, or 50th percentile.

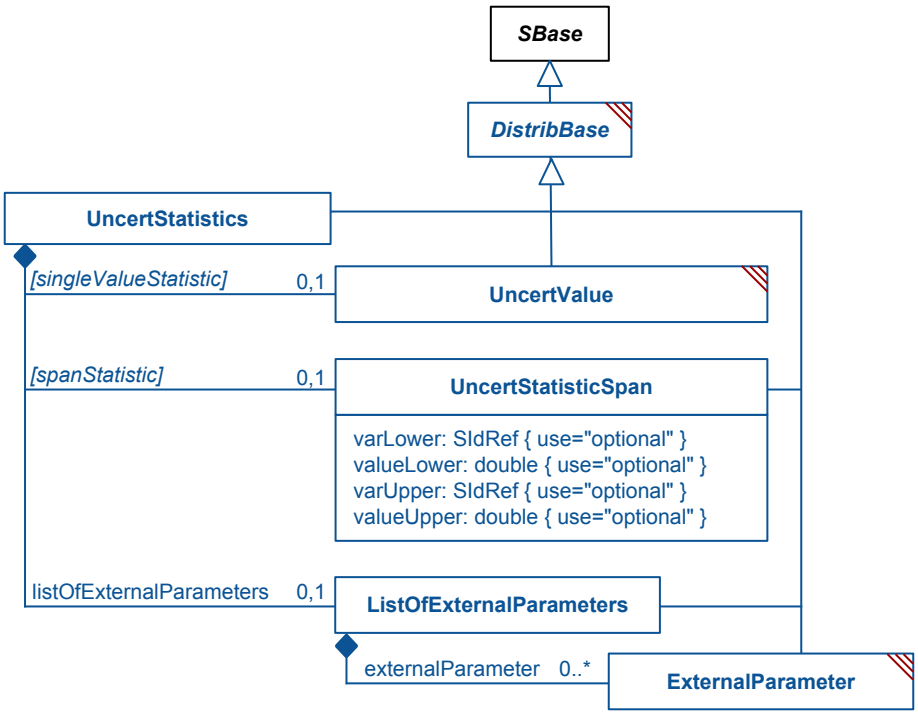

Figure 11: *The definition of the **UncertStatistics** and **UncertStatisticSpan** classes. (The **UncertValue** and **ExternalParameter** classes are defined elsewhere and re-used here.) The **UncertStatistics** class actually has a number of optional children, in three groups: those that can be classified as 'single value' statistics, those that can be classified as a 'span', and those not defined in this specification, but by an external ontology such as ProbOnto. The possible 'single value' statistics are listed in Section 3.25 on the preceding page*

- `mode`: The mode is the value that occurs the most frequently in a data set (or a probability distribution). It need not be unique (e.g. two or more quantities occur equally often) and is typically defined for continuous valued quantities by first defining the histogram, and then giving the central value of the bin containing the most counts.

- `skewness`: The skewness of a random variable is a measure of how asymmetric the corresponding probability distribution is. The skewness is defined as $\frac{\mu_3}{\sigma^3}$ where μ_3 is the 3rd centred moment of the distribution and σ is its standard deviation.

- `standardDeviation`: The standard deviation of a distribution or population is the square root of its variance and is given by $\sigma = \sqrt{E[(X-\mu)^2]}$ where $\mu = E[X]$. The population standard deviation is given by $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$ where $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$, x_i represents with ith observation of the quantity x in the population of size n. The standard deviation is a widely used measure of the variability or dispersion since it is reported in the natural units of the quantity being considered. Note that if a finite sample of a population has been used then the sample standard deviation is the appropriate unbiased estimator to use.

- `variance`: The variance of a random quantity (or distribution) is the average value of the square of the deviation of that variable from its mean, given by $\sigma^2 = Var[X] = E[(X-\mu)^2]$ where $\mu = E[X]$. The complete population variance is given by $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2$ where $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$, x_i represents with ith observation of the quantity x in the population of size n. This is the estimator of the population variance and should be replaced by the sample variance when using samples of finite size.

The possible **UncertStatisticSpan** children are similar, defining a bounded span of values instead of a single value:

- `confidenceInterval`: For a univariate random variable x, a confidence interval is a range $[a, b]$, $a < b$, so that x lies between a and b with given probability. For example, a 95% confidence interval is a range in which x falls 95% of the time (or with probability 0.95). Confidence intervals provide intuitive summaries of the statistics of the variable x.

  If x has a continuous probability distribution P, then $[a, b]$ is a 95% confidence interval if $\int_a^b P(x) = 0.95$.

  Unless specified otherwise, the confidence interval is usually chosen so that the remaining probability is split equally, that is $P(x < a) = P(x > b)$. If x has a symmetric distribution, then the confidence intervals are usually centred around the mean. However, non-centred confidence intervals are possible and are better described by their lower and upper quantiles or levels. For example, a 50% confidence interval would usually lie between the 25% and 75% quantiles, but could in theory also lie between the 10% and 60% quantiles, although this would be rare in practice. The `confidenceInterval` allows you the flexibility to specify non-symmetric confidence intervals however in practice we would expect the main usage to be for symmetric intervals.

  The `confidenceInterval` child of a **UncertStatistics** is always the 95% confidence interval. For other confidence intervals, use an **ExternalParameter** instead.

- `credibleInterval`: In Bayesian statistics, a credible interval is similar to a confidence interval determined from the posterior distribution of a random variable x. That is, given a prior distribution $p(x)$ and some observations D, the posterior probability $p(x \mid D)$ can be computed using Bayes theorem. A 95% credible interval is then any interval $[a, b]$ so that $\int_a^b p(x \mid D) = 0.95$, that is the variable x lies in the interval $[a, b]$ with posterior probability 0.95. Note that the interpretation of a credible interval is not the same as a (frequentist) confidence interval.

  The `credibleInterval` child of a **UncertStatistics** is always the 95% credible interval. For other credibility intervals, use an **ExternalParameter** instead.

- `interquartileRange`: The interquartile range is the range between the 1st and 3rd quartiles. It contains the middle 50% of the sample realisations (or of the sample probability). It is typically used and shown in box plots.

- `range`: The range is the interval $[a, b]$ so that $a < b$ and contains all possible values of x. This is also often called the statistical range, which is the distance from the smallest value to the largest value in a sample dataset. For a sample dataset $X = (x_1, ..., x_N)$, the range is the distance from the smallest x_i to the largest. It is often used as a first estimate of the sample dispersion.

Any number of **ExternalParameter** children may be included as well, each defined by its `definitionURL`.

The following statistics could be added to **UncertStatistics** fairly straightforwardly, but are not in the current version, since each is defined not by a single value, but by at least two, and are not defined by a range. If any of the following are desired, they can be added to a subsequent version. Note that is is possible to encode them in the current version though the use of the **ExternalParameter** construct, though no ontology is known currently that defines these elements (ProbOnto is solely for distributions, and does not encompass statistical measures. UncertML did include them, and though it is now defunct, the old URLs could still be used if desired.)

- `centredMoment`: For a given positive natural number k, the k^{th} central moment of a random variable x is defined as $\mu_k = E[(x - E[x])^k]$. That is, it is the expected value of the deviation from the mean to the power k. In particular, $\mu_0 = 1$, $\mu_1 = 0$ and μ_2 is the variance of x.

- `correlation`: The correlation between two random variables x_1 and x_2 is the extent to which these variable vary together in a linear fashion. It is characterised by the coefficient $\rho_{1,2} = \frac{E[(x_1 - \mu_1)(x_2 - \mu_2)]}{\sigma_1 \sigma_2}$ where μ_1 and μ_2 are the means of x_1 and x_2 respectively, and σ_1 and σ_2 are their respective standard deviations. Note this is strictly not a description of uncertainty, but it can be useful to represent the correlation between two variables. Generally a covariance specification would be preferred since this describes the uncertainty.

- `decile`: A decile, d, is any of the nine values that divide the sorted quantities into ten equal parts, so that each part represents $1/10$ of the sample, population or distribution. The first decile is equivalent to the 10th percentile.

- `moment`: For a given positive natural number k, the k^{th} moment of a random variable x is defined as $\mu_k = E[x^k]$. In particular, $\mu_0 = 1$ and μ_1 is the mean of x. The moments can be defined with respect to some point a, that is $\mu_k(a) = E[(x - a)^k]$. Moments defined about the mean are called centred moments.

- `percentile`: A percentile is the value of a quantity below which a certain percent of values fall. This can be defined for samples, populations and distributions. For finite samples there is no widely accepted method, but all methods essentially rank the quantities and then use some interpolation to compute the percentile, unless the sample size n is a multiple of 100. For probability distributions the inverse cumulative density function can be used. The most widely used method is as follows: to estimate the value, x_p, of the pth percentile of an ascending ordered dataset containing n elements with values $x_1, x_2, ..., x_n$ first compute $\rho = \frac{p}{100}(n - 1) + 1$. Now ρ is split into its integer component, k, and decimal component, d, such that $\rho = k + d$. x_p is then calculated as $x_p = x_k + d(x_{k+1} - x_k)$ where $1 < \rho < n$ with special cases $x_p = x_1$ $[\rho = 1]$; x_n $[\rho = n]$.

- `probability`: Given a random variable x with probability density function $f(x)$, the probability that x lies in some part of its domain $\mathscr{X}$ is defined as $P(x \in \mathscr{X}) = \int_{x \in \mathscr{X}} f(x)$. $\mathscr{X}$ can be defined as a lower- or upper-bounded range, e.g. $P(x < 3.2)$ or as the intersection of several such ranges, e.g. $P(x \geq 1.7 \cap x < 3.2)$.

- `quantile`: Given a random variable x, the n-quantiles are the values of x which split the domain into n regions of equal probability. For instance, the k^{th} n-quantile is the value q_k for which $P(x < q_k) = \frac{k}{n}$. For some common values of n, the n-quantiles have additional names, namely quartiles for $n = 4$, deciles for $n = 10$ and percentiles for $n = 100$. More generally, a quantile can be associated to any probability p, so that q is the value of x below which a proportion p of the probability lies, i.e. $P(x < q) = p$. The plot on the right shows the 1st to 9th 10-quantiles (or deciles) for a normal distribution ($\mu = 4$, $\sigma = 1$) as orange dots. The blue curve is the cumulative density function of x. Note how the quantiles split the probability (y-axis) into 10 equal regions.

- `quartile`: The quartiles are the 4-quantiles, that is the 4 values of x below which lies a proportion 0.25, 0.50, 0.75 and 1 of the probability. One can also think of them as the 4 values of x which split the domain into 4 regions of equal probability.

3.25.1 Attributes inherited from SBase

An **UncertStatistics** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of an **UncertStatistics** has no mathematical meaning.

3.26 The UncertStatisticSpan class

The **UncertStatisticSpan** class defines a span of values that define an uncertainty statistic such as confidence interval or range. It has four optional attributes, `varLower` and `varUpper`, of type `SIdRef`, and `valueLower` and `valueUpper`, of type double. Exactly one of the attributes `varLower` and `vaueLower` may be defined, and exactly one of the attributes `varUpper` and `valueUpper` may be defined. If no attributes are defined, the parameters of the span are undefined. If only one attribute is defined (one of the upper or lower attributes), that aspect of the span is defined, and the other end is undefined. The span is fully defined if two attributes (one lower and one upper) is defined.

The value of the lower attribute (whichever is defined) must be lesser or equal to the value of the upper attribute (whichever is defined), at the initial conditions of the model. The **UncertStatistics** element cannot affect the core mathematics of an SBML model, but if it is used in a mathematical context during simulation of the model, this restriction on the attribute values must be maintained, or the **UncertStatisticSpan** object as a whole will be undefined.

3.26.1 Attributes inherited from SBase

An **UncertStatisticSpan** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of an **UncertStatisticSpan** has no mathematical meaning.

3.27 Examples using extended *SBase*

Several examples are given to illustrate the use of the **Uncertainty** class:

3.27.1 Basic *Uncertainty* example

In this examples, a species is given an **Uncertainty** child to describe its standard deviation:

```
...
      <species id="s1" compartment="C" initialAmount="3.22" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:standardDeviation distrib:value="0.3"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </species>
...
```

Here, the species with an initial amount of 3.22 is described as having a standard deviation of 0.3, a value that might be written as "`3.22 ± 0.3`". This is probably the simplest way to use the package to introduce facts about the uncertainty of the measurements of the values present in the model.

It is also possible to include additional information about the species, should more be known:

```
...
      <species id="s1" compartment="C" initialAmount="3.22" hasOnlySubstanceUnits="true"
               boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:mean distrib:value="3.2"/>
```

```
            <distrib:standardDeviation distrib:value="0.3"/>
            <distrib:variance distrib:value="0.09"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </species>
...
```

In this example, the initial amount of 3.22 is noted as having a mean of 3.2, a standard deviation of 0.3, and a variance of 0.09. Note that the standard deviation can be calculated from the variance (or visa versa), but the modeler has chosen to include both here for convenience. Note too that this use of the **Uncertainty** element does not imply that the species amount comes from a normal distribution with a mean of 3.2 and standard deviation of 0.3, but rather that the species amount comes from an unknown distribution with those qualities. If it is known that the value was drawn from a particular distribution, that distribution should be used, rather than the `Mean` and `StandardDeviation` statistical values.

Note also that 3.22 (the `initialAmount`) 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.

3.27.2 Defining a Random Variable

In addition to describing the uncertainty about an experimental observation one can also use this mechanism to describe a parameter as a random variable. In the example below the parameter, Z, is defined as following a normal distribution, with a given mean and variance. No value is given for the parameter so it is then up the modeler to decide how to use this random variable. For example they may choose to simulate the model in which case they may provide values for mu_Z and var_Z and then sample a random value from the simulation. Alternatively they may choose to carry out a parameter estimation and use experimental observations to estimate mu_Z and var_Z.

```
    <listOfParameters>
      <parameter id="mu_Z" value="10" constant="true"/>
      <parameter id="var_Z" value="0.1" constant="true"/>
      <parameter id="Z" constant="true">
        <distrib:uncertainty>
          <distrib:normalDistribution>
            <distrib:mean distrib:var="mu_Z"/>
            <distrib:variance distrib:var="var_Z"/>
          </distrib:normalDistribution>
        </distrib:uncertainty>
      </parameter>
    </listOfParameters>
```

One could also similarly define a parameter that represented gender through two values:

```
...
      <parameter id="gender" constant="false">
        <distrib:uncertainty>
          <distrib:categoricalDistribution>
            <distrib:listOfCategories>
              <distrib:category id="male">
                <distrib:probability distrib:value="0.5"/>
                <distrib:value distrib:value="0"/>
              </distrib:category>
              <distrib:category id="female">
                <distrib:probability distrib:value="0.5"/>
                <distrib:value distrib:value="1"/>
              </distrib:category>
            </distrib:listOfCategories>
          </distrib:categoricalDistribution>
        </distrib:uncertainty>
      </parameter>
```

```
...
```

4 Interaction with other packages

4.1 Custom annotations for function definitions

Before this package was available, a collection of SBML simulator authors developed an *ad-hoc* convention for exchanging annotated **FunctionDefinition** objects that represented draws from distributions. This convention is described by Frank T. Bergmann at https://docs.google.com/file/d/0B_wMqVOQLkZ3TVZHblNNRWgzNTg/, and represents a basic starting point for any modeler interested in exchanging SBML models containing draws from distributions.

When implementing Distributions support, it would be possible to include 'backwards' support for this annotation convention by annotating any extended **FunctionDefinition** that happens to match the following distribution to also include these annotations, where appropriate.

The following table is taken from the above document by Frank Bergmann, and can be used to annotate **FunctionDefinition** elements that have been extended by Distributions to perform the same functions, providing the arguments are presented in the same order. The suggested fallback function returns the mean of the distribution.

Id	Name	URL	Fallback
uniform	Uniform distribution	http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)	$lambda(a, b, \frac{a+b}{2})$
normal	Normal distribution	http://en.wikipedia.org/wiki/Normal_distribution	$lambda(m, s, m)$
exponential	Exponential distribution	http://en.wikipedia.org/wiki/Exponential_distribution	$lambda(l, \frac{1}{l})$
gamma	Gamma distribution	http://en.wikipedia.org/wiki/Gamma_distribution	$lambda(a, b, a * b)$
poisson	Poisson distribution	http://en.wikipedia.org/wiki/Poisson_distribution	$lambda(mu, mu)$
lognormal	Lognormal distribution	http://en.wikipedia.org/wiki/Log-normal_distribution	$lambda(z, s, e^{z+\frac{s^2}{2}})$
chisq	Chi-squared distribution	http://en.wikipedia.org/wiki/Chi-squared_distribution	$lambda(nu, nu)$
laplace	Laplace distribution	http://en.wikipedia.org/wiki/Laplace_distribution	$lambda(a, 0)$
cauchy	Cauchy distribution	http://en.wikipedia.org/wiki/Cauchy_distribution	$lambda(a, a)$
rayleigh	Rayleigh distribution	http://en.wikipedia.org/wiki/Rayleigh_distribution	$lambda(s, s * \sqrt{\pi/2})$
binomial	Binomial distribution	http://en.wikipedia.org/wiki/Binomial_distribution	$lambda(p, n, p * n)$
bernoulli	Bernoulli distribution	http://en.wikipedia.org/wiki/Bernoulli_distribution	$lambda(p, p)$

As an example, here is a complete (if small) model that uses both the above 'custom annotation' scheme and the Distributions extensions of a **FunctionDefinition**:

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

```
  <model id="__main" name="__main">
    <listOfFunctionDefinitions>
      <functionDefinition id="normal">
        <annotation>
          <distribution xmlns="http://sbml.org/annotations/distribution"
                    definition="http://en.wikipedia.org/wiki/Normal_distribution"/>
        </annotation>
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> mean </ci>
            </bvar>
            <bvar>
              <ci> stddev </ci>
            </bvar>
            <ci> mean </ci>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:listOfDistribInputs>
            <distrib:distribInput distrib:id="mean" distrib:index="0"/>
            <distrib:distribInput distrib:id="stddev" distrib:index="1"/>
          </distrib:listOfDistribInputs>
          <distrib:normalDistribution>
            <distrib:mean distrib:var="mean"/>
            <distrib:stddev distrib:var="stddev"/>
          </distrib:normalDistribution>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>
      <parameter id="x" constant="true"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="x">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <apply>
            <ci> normal </ci>
            <cn> 3 </cn>
            <cn> 0.2 </cn>
          </apply>
        </math>
      </initialAssignment>
    </listOfInitialAssignments>
  </model>
</sbml>
```

4.2 The Arrays package

This package is dependent on no other package, but might rely on the Arrays package to provide vector and matrix structures if those are desired/used. Note that currently, the only way to need arrays is if an **ExternalDistribution** is defined that requires array input or output.

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 a **LogNormalDistribution**. 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"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  level="3" version="1" distrib:required="true">
  <model>
    <listOfFunctionDefinitions>
      <functionDefinition id="logNormal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> scale </ci>
            </bvar>
            <bvar>
              <ci> shape </ci>
            </bvar>
            <notanumber/>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:logNormalDistribution>
            <distrib:shape distrib:var="shape"/>
            <distrib:logScale distrib:var="scale"/>
          </distrib:logNormalDistribution>
          <distrib:listOfInputs>
            <distrib:input distrib:id="scale" distrib:index="0"/>
            <distrib:input distrib:id="shape" distrib:index="1"/>
          </distrib:listOfInputs>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfCompartments>
      <compartment id="central" size="0" constant="true"/>
      <compartment id="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" constant="true"/>
      <parameter id="ke" constant="true"/>
      <parameter id="Cc" constant="false"/>
      <parameter id="Cc_obs" constant="false"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="central">
        <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="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="true"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="Qc" stoichiometry="1" constant="true"/>
        </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="true"/>
```

```
        </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 Multiple uses of distributions

In this example, a **NormalDistribution** is 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 through the use of an extended function definition. 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 assigment; 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"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version1/distrib/version1"
  level="3" version="1" distrib:required="true">
  <model>
    <listOfFunctionDefinitions>
      <functionDefinition id="normal">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <lambda>
            <bvar>
              <ci> x </ci>
            </bvar>
            <bvar>
              <ci> y </ci>
            </bvar>
            <notanumber/>
          </lambda>
        </math>
        <distrib:drawFromDistribution>
          <distrib:normalDistribution>
            <distrib:mean distrib:var="mean"/>
            <distrib:stddev distrib:var="stddev"/>
          </distrib:normalDistribution>
          <distrib:listOfInputs>
            <distrib:input distrib:id="mean" distrib:index="0"/>
            <distrib:input distrib:id="stddev" distrib:index="1"/>
          </distrib:listOfInputs>
        </distrib:drawFromDistribution>
      </functionDefinition>
    </listOfFunctionDefinitions>
    <listOfParameters>
      <parameter id="V" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:mean distrib:var="V_pop"/>
            <distrib:standardDeviation distrib:var="V_omega"/>
```

```
          </distrib:uncertStatistics>
        </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> normal </ci>
            <ci> V_pop </ci>
            <ci> V_omega </ci>
          </apply>
        </math>
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:mean distrib:var="V_pop"/>
            <distrib:standardDeviation distrib:var="V_omega"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </initialAssignment>
    </listOfInitialAssignments>
  </model>
</sbml>
```

5.3 Defining confidence intervals

In this example, several **Parameter** elements are given confidence intervals, and several species are given standard deviations. Each indicates the modeler's assessment of the precision of the estimated given values for those elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version2/core"
  xmlns:distrib="http://www.sbml.org/sbml/level3/version2/distrib/version1"
  level="3" version="2" distrib:required="true">
  <model>
    <listOfCompartments>
      <compartment id="C" spatialDimensions="3" size="1" constant="true"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S1" compartment="C" initialAmount="5.2"
        hasOnlySubstanceUnits="false" boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:standardDeviation distrib:value="0.3"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </species>
      <species id="S2" compartment="C" initialAmount="8.7"
        hasOnlySubstanceUnits="false" boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:standardDeviation distrib:value="0.01"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </species>
      <species id="S3" compartment="C" initialAmount="1102"
        hasOnlySubstanceUnits="false" boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:standardDeviation distrib:value="53"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
```

```
      </species>
      <species id="S4" compartment="C" initialAmount="0.026"
        hasOnlySubstanceUnits="false" boundaryCondition="false" constant="false">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:standardDeviation distrib:value="0.004"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </species>
    </listOfSpecies>
    <listOfParameters>
      <parameter id="P1" value="5.13" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="5" distrib:valueUpper="5.32"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
      <parameter id="P2" value="15" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="10.22" distrib:valueUpper="15.02"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
      <parameter id="P3" value="0.003" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="-0.001" distrib:valueUpper="0.0041"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
      <parameter id="P4" value="0.34" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="0.22" distrib:valueUpper="0.51"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
      <parameter id="P5" value="92" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="90" distrib:valueUpper="99"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
      <parameter id="P6" value="40.002" constant="true">
        <distrib:uncertainty>
          <distrib:uncertStatistics>
            <distrib:confidenceInterval distrib:valueLower="40.0018" distrib:valueUpper="40.0021"/>
          </distrib:uncertStatistics>
        </distrib:uncertainty>
      </parameter>
    </listOfParameters>
  </model>
</sbml>
```

6 Prototype implementations

As of this writing (May 2017), libsbml has full support for elements defined in previous iterations of Distributions, and work is ongoing to develop support for the new distributions and statistics introduced in this version of the specification. Antimony (http://antimony.sf.net/) has support for a limited number of UncertML function (normal, uniform, exponential, gamma, poisson, and truncated versions of these) for model creation only (no simulation). LibRoadRunner (http://libroadrunner.org) also supports the normal and uniform functions (though not their truncated forms), and is a full simulator. Neither Antimony nor LibRoadRunner support the `uncertainty` child of ***SBase***, and support the extended **FunctionDefinition** only.

7 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 authors 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

In this table, all distributions are listed, along with their types (Continuous, Categorical, or Discrete), whether they're univariate or multivariate, and a brief description. The element name is the name of the distribution with spaces removed, the initial letter lower-cased, and "`Distribution`" appended, so, for example, the "`Exponential`" distribution becomes "`<exponentialDistribution>`", and the "`Student T`" distribution becomes "`<studentTDistribution>`".

All of these distributions inherit from the abstract **Distribution** class. Additionally, the appropriate distributions inherit from the **UnivariateDistribution** or **MultivariateDistribution** abstract classes, and further from the **ContinuousUnivariateDistribution**, **DiscreteUnivariateDistribution**, or **CategoricalUnivariateDistribution** classes, which are related to one another as one would expect.

All descriptions are based on the information from `http://www.uncertml.org/`, which is now defunct, but which can still be accessed at `http://web.archive.org/web/20160313012501/uncertml.org`.

Distributions are listed grouped by category (type and univarite/multivariate), and alphabetical within those categories.

A.1 The BetaDistribution class

The **BetaDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `alpha` (α) and `beta` (β). Both `alpha` and `beta` must be positive.

A random variable x is Beta distributed if the probability density function (pdf) is of the form:

$$\frac{1}{B(\alpha,\beta)} x^{\alpha-1} (1-x)^{\beta-1}, \text{ where } B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

The distribution is usually denoted as $x \sim Be(\alpha,\beta)$ with parameters α and β, both positive real values. As the domain of the random variable is defined to be $[0,1]$ the Beta distribution is normally used to describe the distribution of a probability value.

A.2 The CauchyDistribution class

The **CauchyDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `location` (θ) and `scale` (γ). The `scale` value must be positive.

A random variable x follows a Cauchy distribution if the probability density function (pdf) is of the form:

$$\frac{1}{\pi\gamma}\left[1+\left(\frac{x-\theta}{\gamma}\right)^2\right]^{-1}$$

The Cauchy distribution is equivalent to a Student-T distribution with 1 degree of freedom. It is widely used in physics, optics and astronomy. It is also known as the Lorenz or the Breit-Wigner distribution.

A.3 The ChiSquareDistribution class

The **ChiSquareDistribution** is a **ContinuousUnivariateDistribution** defining a **UncertValue** child `degreesOfFreedom` (ν). The `degreesOfFreedom` must be a positive integer.

A random variable x is Chi-square distributed if the probability density function (pdf) is of the form:

$$\frac{1}{\Gamma(\nu/2)2^{\nu/2}} x^{\nu/2-1} exp(-x/2)$$

The distribution is usually denoted as $x \sim \chi_\nu$ where ν is known as the degrees of freedom parameter. ν has to be positive and x has to be non-negative for the density to be defined. The Chi-square distribution is a special case of the Gamma distribution where $\chi \sim \Gamma(k=\nu/2, \theta=2)$.

A.4 The ExponentialDistribution class

The **ExponentialDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** child `rate` (λ). The `rate` value must be positive.

A random variable x follows an exponential distribution if the probability density function (pdf) is of the form:

$$\lambda e^{-\lambda x}$$

It is often represented as $x \sim \text{Exp}(\lambda)$. It is used to model the time between events for a Poisson process and is used in simulation of stochastic systems.

A.5 The FDistribution class

The **FDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `numerator` (ν_1) and `denominator` (ν_2). Both `numerator` and `denominator` must be positive integers.

A random variable x follows an F distribution if the probability density function (pdf) is of the form:

$$\frac{1}{B(\nu_1/2,\nu_2/2)}\left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} x^{\nu_1/2-1}\left(1+\frac{\nu_1}{\nu_2}x\right)^{-\frac{\nu_1+\nu_2}{2}}$$

where $B(.)$ is the Beta function. It often arises as the ratio of two random variables that are identically Chi-Square distributed.

A.6 The GammaDistribution class

The **GammaDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `shape` (k) and `scale` (θ). Both `shape` and `scale` must be positive.

A random variable x is Gamma distributed if the probability density function (pdf) is of the form:

$\frac{1}{\Gamma(k)\theta^k} x^{k-1} exp(-x/\theta)$ with $\Gamma(\cdot)$ the Gamma function.

The distribution is usually denoted as $x \sim Gamma(k,\theta)$ where k is known as the shape parameter and θ the scale parameter. Both parameters have be positive and x has to be non-negative for the density to be defined. In practice the Gamma distribution is often use to model the distribution of non-negative quantities such as variances.

A.7 The InverseGammaDistribution class

The **InverseGammaDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `shape` (α) and `scale` (β). Both `alpha` and `beta` must be positive.

A random variable x is Inverse Gamma distributed if the probability density function (pdf) is of the form:

$$\frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} exp(-\beta/x)$$

If variable x is Inverse Gamma distributed, $1/x$ is gamma distributed. The Inverse Gamma distribution function can be obtained from the Gamma distribution by a transformation of variables.

A.8 The LaPlaceDistribution class

The **LaPlaceDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `location` (μ) and `scale` (b). The `scale` value must be positive.

A random variable x is Laplace distributed if the probability density function (pdf) is of the form:

$$\frac{1}{2b} exp(-\frac{abs(x-\mu)}{b})$$

where abs denotes the absolute value. It can be thought of as a combination of two exponential distributions.

A.9 The LogNormalDistribution class

The **LogNormalDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `shape` (σ^2) and `logScale` (μ). The `shape` value must be positive.

A random variable x is Log Normal distributed if the probability density function (pdf) is of the form:

$$\frac{1}{x\sqrt{2\pi\sigma^2}} exp(-\frac{(\ln(x)-\mu)^2}{2\sigma^2})$$

If variable x is normally distributed, exp(x) is Log Normal distributed. The Log Normal distribution function can be obtained from the normal distribution by a transformation of variables. It is often used for variables that must be positive.

A.10 The LogisticDistribution class

The **LogisticDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `location` (μ) and `scale` (s). The `scale` value must be positive.

A random variable x is Logistic distributed if the probability density function (pdf) is of the form:

$$\frac{exp(-(x-\mu)/s)}{s(1+exp(-(x-\mu)/s))^2}$$

A.11 The NormalDistribution class

The **NormalDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `mean` (μ), `stddev` (σ), and `variance` (σ^2). The distribution must either define a `stddev` or a `variance`, but not both. The `variance`, if defined, must be positive.

A random variable x is normally distributed if the probability density function (pdf) is of the form:

$$\frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

The distribution is usually denoted as $x \sim \mathcal{N}(\mu, \sigma^2)$ where μ is known as the mean parameter and σ^2 the variance parameter. If the random variable x is a vector of length greater than one, the normal distribution can be generalised to the Multivariate normal. A reason for the widespread usage of the normal distribution is the Central limit theorem which states that the distribution of the mean of a large number of independent identically distributed random variables tends to a normal distributions as the number of random variables increases.

A.12 The ParetoDistribution class

The **ParetoDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `scale` (x_m) and `shape` (α). Both `shape` and `scale` must be positive.

A random variable x follows a Pareto distribution if the probability density function is of the form:

$$\frac{\alpha x_m^\alpha}{x^{\alpha+1}}$$

The distribution allows for the specification of a minimum value below which the density is 0. It is a skewed heavy-tailed distribution.

A.13 The RayleighDistribution class

The **RayleighDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `scale`.

[From Wikipedia:] A Rayleigh distribution is often observed when the overall magnitude of a vector is related to its directional components. One example where the Rayleigh distribution naturally arises is when wind velocity is analyzed into its orthogonal 2-dimensional vector components. Assuming that each component is uncorrelated, normally distributed with equal variance, and zero mean, then the overall wind speed (vector magnitude) will be characterized by a Rayleigh distribution. A second example of the distribution arises in the case of random complex numbers whose real and imaginary components are independently and identically distributed Gaussian with equal

variance and zero mean. In that case, the absolute value of the complex number is Rayleigh-distributed.

A.14 The StudentTDistribution class

The **StudentTDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `location` (μ), `scale` (σ^2) and `degreesOfFreedom` (ν).

A random variable x follows a Student-t distribution if the probability density function (pdf) is of the form:

$\frac{\Gamma(\nu/2+1/2)}{\Gamma(\nu/2)(\pi\nu\sigma^2)^{1/2}}[1+\frac{(x-\mu)^2}{\nu\sigma^2}]^{-\nu/2-1/2}$. The distribution is usually denoted as $x \sim St(\mu,\lambda,\nu)$

This distribution corresponds to integrating out the variance of a normal distribution using a inverse Gamma prior. It can therefore be interpreted as an infinite mixture of normal distributions having the same mean but different variances. The three parameters are the mean (μ), degrees of freedom (ν) and variance (σ^2). Setting the variance to 1 and the mean to 0 we obtain the Student-t form found in standard statistics references such as Wikipedia. Setting the d.f. to 1 the Cauchy distribution is obtained. Setting the d.f. to infinity the normal distribution is obtained. The student-t distribution is commonly used in likelihood inference as the maximum likelihood parameter estimates are more robust to outlier observations compared to the normal distribution.

A.15 The UniformDistribution class

The **UniformDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `minimum` (a), `maximum` (b) and the optional `numberOfClasses`. The `minimum` value must be less than the `maximum` value. If `numberOfClasses` is defined, its value must be an integer greater than or equal to two.

A random variable x follows a uniform distribution if the probability density function (pdf) is of the form:

$$\frac{1}{b-a}$$

The distribution assigns equal probability to all events within the chosen domain between (and including) the minimum (a) and the maximum (b).

If `numberOfClasses` is included, the uniform range is divided into $numberOfClasses-1$ sections, and each of the borders of those sections are equally likely to be returned. If `numberOfClasses` is 2 (the minimum), the range just has $2-1=1$ section, and the borders of that section (the `minimum` and `maximum`) are the two possible return values. If `numberofClasses` is 3, the range is broken into $3-1=2$ sections, leaving the `minimum`, `maximum`, and mean as the three possible return values, etc.

A.16 The WeibullDistribution class

The **WeibullDistribution** is a **ContinuousUnivariateDistribution** that defines the **UncertValue** children `shape` (k) and `scale` (λ). Both `shape` and `scale` must be positive.

A random variable x follows an Weibull distribution if the probability density function (pdf) is of the form:

$$\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} exp(-x/\lambda)^k$$

It includes the exponential distribution as a special case. It is often used in engineering and finance.

A.17 The BinomialDistribution class

The **BinomialDistribution** is a **DiscreteUnivariateDistribution** that defines the **UncertValue** children `numberOfTrials` (n) and `probabilityOfSuccess` (θ). The `numberOfTrials` must be a positive integer, and `probabilityOfSuccess` must be a value between zero and one, inclusive.

A random variable x follows a Binomial distribution if the probability mass function (pmf) is of the form:

$$\binom{n}{x}\theta^x(1-\theta)^{n-x}$$

where $\binom{n}{x}$ denotes n choose x. The distribution is usually denoted as $x \sim b(n,\theta)$. The distribution describes the probability of getting x successes in n trials of independent experiments that have the same probability of success.

A.18 The GeometricDistribution class

The **GeometricDistribution** is a **DiscreteUnivariateDistribution** that defines the **UncertValue** child `probability` (p). The `probability` must have a value must be between zero and one, inclusive.

A random variable x follows a geometric distribution if the probability mass function (pmf) is of the form:

$$(1-p)^{x-1}p$$

It is often represented as $x \sim \text{Geom}(p)$. It is the discrete analogue of the exponential distribution. It is used to model distribution of the number of binary (Bernoulli) trials needed to get one success, with parameter, probability p.

A.19 The HypergeometricDistribution class

The **HypergeometricDistribution** is a **DiscreteUnivariateDistribution** that defines the **UncertValue** children `numberOfSuccesses` (m). `numberOfTrials` (n), and `populationSize` (N). All three values must be positive integers, chosen such that `numberOfTrials` is less than or equal to `populationSize`.

A random variable x follows a hypergeometric distribution if the probability mass function (pmf) is of the form:

$$\frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$$

probability of getting x successes. It describes the number of successes in a sequence of draws without replacement.

A.20 The NegativeBinomialDistribution class

The **NegativeBinomialDistribution** is a **DiscreteUnivariateDistribution**. It has two defined **UncertValue** children `numberOfFailures` (r) and `probability` (p). The `numberOfFailures` must be a positive integer, and `probability` must have a value between zero and one, inclusive.

A random variable x follows a Negative Binomial distribution if the probability mass function (pmf) is of the form:

$$\binom{x+r-1}{x}p^x(1-p)^r$$

The distribution describes the probability of getting x successes in trials of independent experiments that have the same probability of success, and are run until we observe r failures. Note that some systems formulate this distribution differently: observing k failures before obtaining the r^{th} success. The formulation above follows the English version of Wikipedia; the alternate formulation is used on other language Wikipedia definitions of the distribution, as well as various software packages like Matlab and R.

? Lucian: NOTE! The above formulation was used by UncertML and Wikipedia, the sort-of-default distribution definition source for the annotation scheme. However, once people actually start implementing it, they may find that their software package uses the alternative. The ProbOnto 2.5 specification (`https://sites.google.com/site/probonto/download`) goes into great detail on this issue in Appendix A.3, for anyone who wants to know more. I would be happy to change the definition to match people's software, if need be.

A.21 The PoissonDistribution class

The **PoissonDistribution** is a **DiscreteUnivariateDistribution** that defines the **UncertValue** child `rate` (λ). The `rate` value must be positive.

A random variable x follows a Poisson distribution if the probability mass function (pmf) is of the form:

$$\frac{\lambda^x}{x!}\exp(-\lambda)$$

The Poisson distribution can be used to model the number of events occurring within fixed time period of time.

A.22 The BernoulliDistribution class

The **BernoulliDistribution** is a **CategoricalUnivariateDistribution** that defines the **UncertValue** child `prob` (μ). The `prob` must have a value between zero and one, inclusive. It defines the probability that $x = 1$.

A random variable x follows a Bernoulli distribution if the probability mass function (pmf) is of the form:

$$\mu^x(1-\mu)^{1-x}$$

It describes the distribution of a single binary variable x.

A.23 The CategoricalDistribution class

The **CategoricalDistribution** is a **CategoricalUnivariateDistribution** that contains one or more **Category** elements, each of which defines **UncertValue** `value` and `probability` children associated with that category. In order to be valid, the sum of the probabilities over all categories must either equal 1.0, or there must be exactly one **Category** without a child **UncertValue** `probability`, which is then set to $1.0 - sum(otherprobabilities)$. (In this case, that sum must be between 0.0 and 1.0, inclusive.)

A Categorical distribution is a generalisation of the Bernoulli distribution to K discrete outcomes, giving the K probabilities p_i, $i = 1, .., K$ for each outcome. There is no ordering in the K outcomes.

The optional `rank` attribute, if present, is provided as a way to differentiate between an ordered vs. unordered categorical distribution. It does not affect the sampling of the distribution in any way, and is provided for reference only. The `rank` attributes, if present, must be unique among the **Category** elements of a single **CategoricalDistribution**, and must begin with "`0`". Thus, if one **Category** with a `rank` is present, the value of its `rank` must be "`0`"; if there are two, they must be "`0`" and "`1`", etc.

A.24 The ListOfCategories class

The **ListOfCategories** class, like other **ListOf**_____ classes in SBML Level 3 Core, is a container for one or more **Category** objects. Unlike many of **ListOf**_____ classes in SBML Level 3 Core, at least one child **Category** is required, because the behavior of the parent distribution would be undefined if it had no child **Category** objects from which to choose.

A.25 The Category class

The **Category** class has a required **UncertValue** child `value`, and an optional **UncertValue** child `probability`. In any **CategoricalDistribution**, only one child **Category** may have an undefined `probability`; the rest must be defined and their totals add up to less than one. If all **Category** children have defined `probability` children, the total of all of those probabilities must add up to exactly one.

Each **Category** defines a `value`, and that value's `probability` of being sampled from that distribution. If the `probability` is not explicitly defined, it is implicitly defined as one minus the sum of the probabilities of all the other **Category** objects in the same **CategoricalDistribution**.

A.25.1 Attributes inherited from SBase

A **Category** always inherits the optional `metaid` and `sboTerm` attributes, and inherits optional `id` and `name` attributes as described in Section 3.5 on page 13. The `id` of a **Category** has no mathematical meaning.

B Validation of SBML documents

B.1 Validation and consistency rules

This section summarizes all the conditions that must (or in some cases, at least *should*) be true of an SBML Level 3 Version 2 model that uses the Distributions package. We use the same conventions as are used in the SBML Level 3 Version 2 Core specification document. In particular, there are different degrees of rule strictness. Formally, the differences are expressed in the statement of a rule: either a rule states that a condition *must* be true, or a rule states that it *should* be true. Rules of the former kind are strict SBML validation rules—a model encoded in SBML must conform to all of them in order to be considered valid. Rules of the latter kind are consistency rules. To help highlight these differences, we use the following three symbols next to the rule numbers:

☑ A checked box indicates a *requirement* for SBML conformance. If a model does not follow this rule, it does not conform to the Distributions package specification. (Mnemonic intention behind the choice of symbol: "This must be checked.")

▲ A triangle indicates a *recommendation* for model consistency. If a model does not follow this rule, it is not considered strictly invalid as far as the Distributions package specification is concerned; however, it indicates that the model contains a physical or conceptual inconsistency. (Mnemonic intention behind the choice of symbol: "This is a cause for warning.")

★ A star indicates a strong recommendation for good modeling practice. This rule is not strictly a matter of SBML encoding, but the recommendation comes from logical reasoning. As in the previous case, if a model does not follow this rule, it is not strictly considered an invalid SBML encoding. (Mnemonic intention behind the choice of symbol: "You're a star if you heed this.")

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

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

General rules about this package

distrib-10101 ☑ To conform to the Distributions package specification for SBML Level 3 Version 2, an SBML document must declare "`http://www.sbml.org/sbml/level3/version2/distrib/version1`" as the XMLNamespace to use for elements of this package. (Reference: SBML Level 3 Package specification for Distributions, Version 1 Section 3.2 on page 11.)

distrib-10102 ☑ Wherever they appear in an SBML document, elements and attributes from the Distributions package must use the "`http://www.sbml.org/sbml/level3/version2/distrib/version1`" namespace, declaring so either explicitly or implicitly. (Reference: SBML Level 3 Package specification for Distributions, Version 1 Section 3.2 on page 11.)

General rules about identifiers

distrib-10301 ☑ (Extends validation rule #10301 in the SBML Level 3 Core specification. TO DO list scope of ids) (Reference: SBML Level 3 Version 1 Core, Section 3.1.7.)

distrib-10302 ☑ The value of a `distrib:id` must conform to the syntax of the **SBML** data type `SId` (Reference: SBML Level 3 Version 1 Core, Section 3.1.7.)

TODO: ANY LIST OF ELEMENTS THAT HAVE ATTRIBUTES

Rules for the extended SBML class

distrib-20101 ☑ In all SBML documents using the Distributions package, the **SBML** object must have the `distrib:required` attribute. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)

distrib-20102 ☑ The value of attribute `distrib:required` on the **SBML** object must be of data type `boolean`. (Reference: SBML Level 3 Version 1 Core, Section 4.1.2.)

distrib-20103 ☑ The value of attribute `distrib:required` on the **SBML** object must be set to "`true`". (Reference: SBML Level 3 Package specification for Distributions, Version 1 Section 3.2 on page 11.)

Rules for extended FunctionDefinition object

distrib-20201 ☑ A **FunctionDefinition** object may contain one and only one instance of the **DrawFromDistribution** element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **FunctionDefinition** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.6 on page 14.)

Rules for extended SBase object

distrib-20301 ☑ A ***SBase*** object may have the optional attributes `distrib:id` and `distrib:name`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on a ***SBase*** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.23 on page 28.)

distrib-20302 ☑ A ***SBase*** object may contain one and only one instance of the **Uncertainty** element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a ***SBase*** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.23 on page 28.)

distrib-20303 ☑ The attribute `distrib:name` on a ***SBase*** must have a value of data type `string`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.23 on page 28.)

Rules for DrawFromDistribution object

distrib-20401 ☑ A **DrawFromDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **DrawFromDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20402 ☑ A **DrawFromDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **DrawFromDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20403 ☑ A **DrawFromDistribution** object may contain one and only one instance of each of the **ListOfDistribInputs** and Distribution elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **DrawFromDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.7 on page 15.)

distrib-20404 ☑ Apart from the general notes and annotations subobjects permitted on all SBML objects, a **ListOfDistribInputs** container object may only contain **DistribInput** objects. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.8 on page 15.)

distrib-20405 ☑ A **ListOfDistribInputs** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ListOfDistribInputs** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.8 on page 15.)

Rules for DistribInput object

distrib-20501 ☑ A **DistribInput** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **DistribInput**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20502 ☑ A **DistribInput** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **DistribInput**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20503 ☑ A **DistribInput** object may have the optional attributes `distrib:id`, `distrib:name` and `distrib:index`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on a **DistribInput** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.9 on page 15.)

distrib-20504 ☑ The attribute `distrib:name` on a **DistribInput** must have a value of data type `string`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.9 on page 15.)

distrib-20505 ☑ The attribute `distrib:index` on a **DistribInput** must have a value of data type `integer`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.9 on page 15.)

Rules for Distribution object

distrib-20601 ☑ A **Distribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **Distribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20602 ☑ A **Distribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **Distribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

Rules for UnivariateDistribution object

distrib-20701 ☑ An **UnivariateDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20702 ☑ An **UnivariateDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

Rules for MultivariateDistribution object

distrib-20801 ☑ A **MultivariateDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **MultivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20802 ☑ A **MultivariateDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **MultivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

Rules for ContinuousUnivariateDistribution object

distrib-20901 ☑ A **ContinuousUnivariateDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ContinuousUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20902 ☑ A **ContinuousUnivariateDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **ContinuousUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-20903 ☑ A **ContinuousUnivariateDistribution** object may contain one and only one instance of each of the UncertBound and UncertBound elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **ContinuousUnivariateDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.13 on page 17.)

Rules for DiscreteUnivariateDistribution object

distrib-21001 ☑ A **DiscreteUnivariateDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **DiscreteUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21002 ☑ A **DiscreteUnivariateDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **DiscreteUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21003 ☑ A **DiscreteUnivariateDistribution** object may contain one and only one instance of each of the UncertBound and UncertBound elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **DiscreteUnivariateDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.14 on page 17.)

Rules for CategoricalUnivariateDistribution object

distrib-21101 ☑ A **CategoricalUnivariateDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **CategoricalUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21102 ☑ A **CategoricalUnivariateDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **CategoricalUnivariateDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

Rules for UncertValue object

distrib-21201 ☑ An **UncertValue** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UncertValue**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21202 ☑ An **UncertValue** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UncertValue**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21203 ☑ An **UncertValue** object may have the optional attributes `distrib:value`, `distrib:var` and `distrib:units`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on an **UncertValue** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.1 on page 21.)

distrib-21204 ☑ The attribute `distrib:value` on an **UncertValue** must have a value of data type `double`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.1 on page 21.)

distrib-21205 ☑ The value of the attribute `distrib:var` of an **UncertValue** object must be the identifier of an existing ***SBase*** object defined in the enclosing **Model** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.1 on page 21.)

distrib-21206 ☑ The value of the attribute `distrib:units` on an **UncertValue** must have a taken from the following: the identifier of a **UnitDefinition** object in the enclosing **Model**, or one of the base units in SBML. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.1 on page 21.)

Rules for UncertBound object

distrib-21301 ☑ An **UncertBound** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UncertBound**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21302 ☑ An **UncertBound** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UncertBound**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21303 ☑ An **UncertBound** object must have the required attribute `distrib:inclusive`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on an **UncertBound** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.3 on page 22.)

distrib-21304 ☑ The attribute `distrib:inclusive` on an **UncertBound** must have a value of data type `boolean`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.16.3 on page 22.)

Rules for ExternalDistribution object

distrib-21401 ☑ An **ExternalDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **ExternalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21402 ☑ An **ExternalDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **ExternalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21403 ☑ An **ExternalDistribution** object must have the required attribute `distrib:definitionURL`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on an **ExternalDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.17 on page 22.)

distrib-21404 ☑ An **ExternalDistribution** object may contain one and only one instance of the **ListOfExternalParameters** element. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **ExternalDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.17 on page 22.)

distrib-21405 ☑ The attribute `distrib:definitionURL` on an **ExternalDistribution** must have a value of data type `string`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.17 on page 22.)

distrib-21406 ☑ Apart from the general notes and annotations subobjects permitted on all SBML objects, a **ListOfExternalParameters** container object may only contain **ExternalParameter** objects. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

distrib-21407 ☑ A **ListOfExternalParameters** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ListOfExternalParameters** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

Rules for ExternalParameter object

distrib-21501 ☑ An **ExternalParameter** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **ExternalParameter**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21502 ☑ An **ExternalParameter** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **ExternalParameter**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21503 ☑ An **ExternalParameter** object must have the required attribute `distrib:definitionURL`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on an **ExternalParameter** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.19 on page 23.)

distrib-21504 ☑ An **ExternalParameter** object may contain one and only one instance of the **ListOfExternalParameters** element. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **ExternalParameter** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.19 on page 23.)

distrib-21505 ☑ The attribute `distrib:definitionURL` on an **ExternalParameter** must have a value of data type `string`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.19 on page 23.)

distrib-21506 ☑ Apart from the general notes and annotations subobjects permitted on all SBML objects, a **ListOfExternalParameters** container object may only contain **ExternalParameter** objects. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

distrib-21507 ☑ A **ListOfExternalParameters** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ListOfExternalParameters** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

Rules for NormalDistribution object

distrib-21601 ☑ A **NormalDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **NormalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21602 ☑ A **NormalDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **NormalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21603 ☑ A **NormalDistribution** object must contain one and only one instance of the UncertValue element, and may contain one and only one instance of each of the UncertValue and UncertValue elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **NormalDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.11 on page 47.)

Rules for UniformDistribution object

distrib-21701 ☑ An **UniformDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UniformDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21702 ☑ An **UniformDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UniformDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21703 ☑ An **UniformDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements, and may contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **UniformDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.15 on page 48.)

Rules for CategoricalDistribution object

distrib-21801 ☑ A **CategoricalDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **CategoricalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21802 ☑ A **CategoricalDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **CategoricalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21803 ☑ A **CategoricalDistribution** object must contain one and only one instance of the **ListOfCategories** element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **CategoricalDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.23 on page 50.)

distrib-21804 ☑ The **ListOfCategories** subobject on a **CategoricalDistribution** object must not be empty. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.23 on page 50.)

distrib-21805 ☑ Apart from the general notes and annotations subobjects permitted on all SBML objects, a **ListOfCategories** container object may only contain **Category** objects. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.24 on page 50.)

distrib-21806 ☑ A **ListOfCategories** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ListOfCategories** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.24 on page 50.)

Rules for Category object

distrib-21901 ☑ A **Category** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **Category**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21902 ☑ A **Category** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **Category**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-21903 ☑ A **Category** object may have the optional attribute `distrib:rank`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on a **Category** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.25 on page 50.)

distrib-21904 ☑ A **Category** object must contain one and only one instance of the UncertValue element, and may contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **Category** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.25 on page 50.)

distrib-21905 ☑ The attribute `distrib:rank` on a **Category** must have a value of data type `integer`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.25 on page 50.)

Rules for BernoulliDistribution object

distrib-22001 ☑ A **BernoulliDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **BernoulliDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22002 ☑ A **BernoulliDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **BernoulliDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22003 ☑ A **BernoulliDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **BernoulliDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.22 on page 49.)

Rules for BetaDistribution object

distrib-22101 ☑ A **BetaDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **BetaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22102 ☑ A **BetaDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **BetaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22103 ☑ A **BetaDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **BetaDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.1 on page 45.)

Rules for BinomialDistribution object

distrib-22201 ☑ A **BinomialDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **BinomialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22202 ☑ A **BinomialDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **BinomialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22203 ☑ A **BinomialDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **BinomialDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.17 on page 48.)

Rules for CauchyDistribution object

distrib-22301 ☑ A **CauchyDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a

CauchyDistribution. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22302 ☑ A **CauchyDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **CauchyDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22303 ☑ A **CauchyDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **CauchyDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.2 on page 45.)

Rules for ChiSquareDistribution object

distrib-22401 ☑ A **ChiSquareDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ChiSquareDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22402 ☑ A **ChiSquareDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **ChiSquareDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22403 ☑ A **ChiSquareDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **ChiSquareDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.3 on page 45.)

Rules for ExponentialDistribution object

distrib-22501 ☑ An **ExponentialDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **ExponentialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22502 ☑ An **ExponentialDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **ExponentialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22503 ☑ An **ExponentialDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **ExponentialDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.4 on page 46.)

Rules for FDistribution object

distrib-22601 ☑ A **FDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **FDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22602 ☑ A **FDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **FDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22603 ☑ A **FDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **FDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.5 on page 46.)

Rules for GammaDistribution object

distrib-22701 ☑ A **GammaDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **GammaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22702 ☑ A **GammaDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **GammaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22703 ☑ A **GammaDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **GammaDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.6 on page 46.)

Rules for GeometricDistribution object

distrib-22801 ☑ A **GeometricDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **GeometricDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22802 ☑ A **GeometricDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **GeometricDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22803 ☑ A **GeometricDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **GeometricDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.18 on page 49.)

Rules for HypergeometricDistribution object

distrib-22901 ☑ A **HypergeometricDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **HypergeometricDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22902 ☑ A **HypergeometricDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **HypergeometricDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-22903 ☑ A **HypergeometricDistribution** object must contain one and only one instance of each of the UncertValue, UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **HypergeometricDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.19 on page 49.)

Rules for InverseGammaDistribution object

distrib-23001 ☑ An **InverseGammaDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **InverseGammaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23002 ☑ An **InverseGammaDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **InverseGammaDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23003 ☑ An **InverseGammaDistribution** object must contain one and only one instance of each of the

UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **InverseGammaDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.7 on page 46.)

Rules for LaPlaceDistribution object

distrib-23101 ☑ A **LaPlaceDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **LaPlaceDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23102 ☑ A **LaPlaceDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **LaPlaceDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23103 ☑ A **LaPlaceDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **LaPlaceDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.8 on page 46.)

Rules for LogNormalDistribution object

distrib-23201 ☑ A **LogNormalDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **LogNormalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23202 ☑ A **LogNormalDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **LogNormalDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23203 ☑ A **LogNormalDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **LogNormalDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.9 on page 47.)

Rules for LogisticDistribution object

distrib-23301 ☑ A **LogisticDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **LogisticDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23302 ☑ A **LogisticDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **LogisticDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23303 ☑ A **LogisticDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **LogisticDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.10 on page 47.)

Rules for NegativeBinomialDistribution object

distrib-23401 ☑ A **NegativeBinomialDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **NegativeBinomialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23402 ☑ A **NegativeBinomialDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces

are permitted on a **NegativeBinomialDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23403 ☑ A **NegativeBinomialDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **NegativeBinomialDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.20 on page 49.)

Rules for ParetoDistribution object

distrib-23501 ☑ A **ParetoDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ParetoDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23502 ☑ A **ParetoDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **ParetoDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23503 ☑ A **ParetoDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **ParetoDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.12 on page 47.)

Rules for PoissonDistribution object

distrib-23601 ☑ A **PoissonDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **PoissonDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23602 ☑ A **PoissonDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **PoissonDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23603 ☑ A **PoissonDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **PoissonDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.21 on page 49.)

Rules for RayleighDistribution object

distrib-23701 ☑ A **RayleighDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **RayleighDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23702 ☑ A **RayleighDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **RayleighDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23703 ☑ A **RayleighDistribution** object must contain one and only one instance of the UncertValue element. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **RayleighDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.13 on page 47.)

Rules for StudentTDistribution object

distrib-23801 ☑ A **StudentTDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **StudentTDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23802 ☑ A **StudentTDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **StudentTDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23803 ☑ A **StudentTDistribution** object must contain one and only one instance of each of the UncertValue, UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **StudentTDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.14 on page 48.)

Rules for WeibullDistribution object

distrib-23901 ☑ A **WeibullDistribution** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **WeibullDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23902 ☑ A **WeibullDistribution** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on a **WeibullDistribution**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-23903 ☑ A **WeibullDistribution** object must contain one and only one instance of each of the UncertValue and **UncertValue** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on a **WeibullDistribution** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Appendix A.16 on page 48.)

Rules for Uncertainty object

distrib-24001 ☑ An **Uncertainty** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **Uncertainty**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24002 ☑ An **Uncertainty** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **Uncertainty**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24003 ☑ An **Uncertainty** object may contain one and only one instance of each of the UncertStatistics and Distribution elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **Uncertainty** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.24 on page 29.)

Rules for UncertStatistics object

distrib-24101 ☑ An **UncertStatistics** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UncertStatistics**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24102 ☑ An **UncertStatistics** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UncertStatistics**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24103 ☑ An **UncertStatistics** object may contain one and only one instance of each of the UncertValue, UncertValue, UncertValue, UncertValue, UncertValue, UncertValue, UncertValue, UncertStatisticSpan, UncertStatisticSpan, UncertStatisticSpan, UncertStatisticSpan and **ListOfExternalParameters** elements. No other elements from the SBML Level 3 Distributions namespaces are permitted on an **UncertStatistics** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.25 on page 29.)

distrib-24104 ☑ Apart from the general notes and annotations subobjects permitted on all SBML objects, a **ListOfExternalParameters** container object may only contain **ExternalParameter** objects.

(Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

distrib-24105 ☑ A **ListOfExternalParameters** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on a **ListOfExternalParameters** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.18 on page 23.)

Rules for UncertStatisticSpan object

distrib-24201 ☑ An **UncertStatisticSpan** object may have the optional SBML Level 3 Core attributes `metaid` and `sboTerm`. No other attributes from the SBML Level 3 Core namespaces are permitted on an **UncertStatisticSpan**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24202 ☑ An **UncertStatisticSpan** object may have the optional SBML Level 3 Core subobjects for notes and annotations. No other elements from the SBML Level 3 Core namespaces are permitted on an **UncertStatisticSpan**. (Reference: SBML Level 3 Version 1 Core, Section 3.2.)

distrib-24203 ☑ An **UncertStatisticSpan** object may have the optional attributes `distrib:varLower`, `distrib:valueLower`, `distrib:varUpper` and `distrib:valueUpper`. No other attributes from the SBML Level 3 Distributions namespaces are permitted on an **UncertStatisticSpan** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.26 on page 33.)

distrib-24204 ☑ The value of the attribute `distrib:varLower` of an **UncertStatisticSpan** object must be the identifier of an existing ***SBase*** object defined in the enclosing **Model** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.26 on page 33.)

distrib-24205 ☑ The attribute `distrib:valueLower` on an **UncertStatisticSpan** must have a value of data type `double`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.26 on page 33.)

distrib-24206 ☑ The value of the attribute `distrib:varUpper` of an **UncertStatisticSpan** object must be the identifier of an existing ***SBase*** object defined in the enclosing **Model** object. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.26 on page 33.)

distrib-24207 ☑ The attribute `distrib:valueUpper` on an **UncertStatisticSpan** must have a value of data type `double`. (Reference: SBML Level 3 Package specification for Distributions, Version 1, Section 3.26 on page 33.)

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 Publishing Company.

Swat, M., Grenon, P., and S.M.Wimalaratne (2016). Probonto - ontology and knowledge base of probability distributions. *Bioinformatics*, 17(32):2719–2721.

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/.