Proposal / Contract no.: 248488
Project start: February 1, 2010
Project end: January 31, 2013

UncertWeb

The *Uncert*ainty Enabled Model *Web*

SEVENTH FRAMEWORK PROGRAMME

THEME FP7-ICT-2009-4

ICT for Environmental Services and Climate Change Adaptation

Deliverable 1.2
UncertML best practice proposal

Title of Deliverable	UncertML best practice proposal
Deliverable reference number	D1.2
Related WP and Tasks	WP1, Task 1.2
Type of Document	Public
Authors	Dan Cornford, Matthew Williams
Date	30 March 2011
Version	2.0

Project coordinator

Dr. Dan Cornford

Aston University, United Kingdom

E-mail: d.cornford@aston.ac.uk

http://www.uncertweb.org/

Revision History

Version	Date	Changes	Authors
0.1	09-03-2011	Initial draft	Dan Cornford
0.2	18-03-2011	Added some introductory material	Dan Cornford
0.3	21-03-2011	Added UncertML user guide	Alexis Boukouvalas, Remi Barillec, Matt Williams, Dan Cornford
0.4	23-03-2011	Added UncertML and other standards	Dan Cornford
0.5	24-03-2011	Added UncertML design and overview of user guide	Matt Williams
1.0	25-03-2011	Edits to the document for review	Dan Cornford
1.1	29-03-2011	Review, suggested minor changes	Gerard Heuvelink
2.0	30-03-2011	Final version incorporating changes from review	Dan Cornford

Related task

Task 1.2 UncertML schema

Design the schema and define the semantics for UncertML, attempting to limit dependencies on other schema and maintain modularity as far as possible. UncertML currently offers support for basic uncertainty types based on an extensive and extendable range of sample statistics and probability distributions including mixture models and realisations from random variables or processes. It is clear from preliminary research that it will be necessary to extend UncertML to deal with a wider range of uncertainty representations including support for random processes, the ability to represent conditional distributions, the option to encode Bayes Linear representations (where expectation is the primitive rather than probability density) and possibly fuzzy representations. Consult with interest bodies (e.g. the Data Quality and Sensor Web Enablement groups in OGC, Semantic Web uncertainty community, the statistics community) and W3C to gather the maximum possible consensus at the design stage. Produce a full UncertML schema and supporting documentation.

Active partners: AST, UOM, CNR

Legal Notices

Executive Summary

This report describes a proposal and schema for UncertML 2.0. It forms the basis for a submission as a best practice paper to the Open Geospatial Consortium (OGC). It begins with a review of UncertML 1.0, and highlights the changes that are made to UncertML 1.0 in moving to version 2.0. It then describes the design and usage of UncertML 2.0. The main value of the deliverable is in the 'user guide' section, which we anticipate will be widely used as a reference for using UncertML 2.0.

UncertML was originally created to address the lack of any mechanism for precisely defining values that were not known with certainty. While some schema included some notions of variability or variation, nothing really addressed the issue of uncertainty in a complete and standardised manner. The most relevant XML schema were the data quality elements in the metadata models, for example ISO19115 and extensions, however the XML implementations of these models lacked well defined representations of uncertainties, and those that were there allowed any string to describe the quantitative results, meaning there was really no chance of software being able to interpret and use this information, since one could have anything in the string.

UncertML 2.0 is designed to be agnostic to the domain of application, rather like base types (e.g. float, string, boolean, integer) are used across application domains. We contend that uncertain values are similar in that they are something present in all application domains. Probability theory is the most widely accepted mechanism to represent uncertain information. Thus UncertML 2.0, driven by the requirements from both the UncertWeb project and the broader environmental modelling and statistical modelling communities focusses on probabilistic representations of uncertainty.

UncertML 2.0 takes the idea of domain neutrality further. We have removed dependencies on external XML schema (version 1.0 used SWE Common extensively) to keep the model and schema as simple as possible. We have also restricted the schema to hard types, to allow a complete application programmers interface to be written. This makes it much easier for others to use the schema in practice. It is intended that UncertML 2.0 can be used across a range of application domains, including the geospatial domain, environmental modelling and even systems biology. This will be of benefit to all domains, since it will facilitate the reuse of (typically web based) tools for applications that use UncertML.

The intention to keep UncertML 2.0 simple means we have deliberately constrained its scope. UncertML 2.0 addresses uncertainty in single and multiple variables using probabilistic representations. Within UncertWeb, UncertML 2.0 will be used for all uncertain quantities. It is key to the communication of uncertainty between the data and processing services, and will be used in all the tools being developed in work package 3 of UncertWeb.

This document describes UncertML 2.0, introduces the online user guide (and presents this in Appendix A), and discusses how UncertML 2.0 should be used within other encodings and standards.

Contents

1 Introduction

This report describes a proposal and schema for UncertML 2.0. It forms the basis for a submission as a best practice paper to the Open Geospatial Consortium (OGC). It begins with a review of UncertML 1.0, and highlights the changes that have been made to UncertML 1.0 in moving to version 2.0.

UncertML was originally created to address the lack of any mechanism for precisely defining values that were not known with certainty. While some schema included some notions of variability or variation, nothing really addressed the issue of uncertainty in a complete and standardised manner. The most relevant XML schema were the data quality elements in the metadata models, for example ISO19115 and extensions, however the XML implementations of these models lacked well defined representations of uncertainties, and those that were there allowed any string to describe the quantitative results, meaning there was really no chance of software being able to interpret and use this information, since one could have anything in the string.

UncertML was designed to be agnostic to the domain of application, rather like base types (e.g. float, string, boolean, integer) are used across application domains. We contend that uncertain values are similar in that they are something present in all application domains. Probability theory is the most widely accepted mechanism to represent uncertain information, indeed all other single valued representations either reduce to probability or are inconsistent with the widely accepted axioms of Cox[1].

UncertML 2.0 takes the idea of domain neutrality further. We have removed dependencies on external XML schema (version 1.0 used SWE Common extensively) to keep the schema as simple as possible. We have also restricted the schema to hard types, to allow a complete application programmers interface to be written. This makes it much easier for others to use the schema in practice. It is intended that UncertML 2.0 can be used across a range of application domains, including the geospatial domain, environmental modelling and even systems biology. This will be of benefit to all domains, since it will facilitate the reuse of web based tools, in much the way that the R[2] environment has brought together a range of statistical packages, for applications that use UncertML.

The intention to keep UncertML 2.0 simple means we have deliberately constrained it's scope. UncertML 2.0 addresses uncertainty in single and multiple variables using probabilistic representations. It does not encode all possible probability distributions, but all widely used distributions are included, it does not support all summary statistics, but all widely used statistics are included and it does not support every type of sampling, but the commonly used methods are included. We imagine that in future versions UncertML will include an even greater range of probabilistic representations of uncertainty, but in 2.0 we focus on the types required directly with the UncertWeb scenarios and also on widely used representations.

Within UncertWeb UncertML 2.0 will be used to represent all uncertain quantities. It is key to the communication of uncertainty between the data and processing services, and will be used in the majority of the tools being developed in work package 3 of UncertWeb. Since

[1] Cox, R. T. (1961). The Algebra of Probable Inference, Johns Hopkins University Press, Baltimore, 127pp

[2] http://www.r-project.org/

UncertML 2.0 plays a central role in UncertWeb, it is important to fix this relatively early in the project.

The deliverable is structured in the following way. The document starts with an introduction to the original UncertML 1.0 in Section 2. Section 3 lists the requirements for UncertML 2.0, and the changes that have been made from version 1.0 to 2.0 are summarised in Section 4. The conceptual model at the heart of UncertML 2.0 is described in Section 5. The main contribution of this deliverable is the UncertML 2.0 User Guide, which is described in Section 6, with the complete version being shown in Appendix A. Section 7 discusses the usage of UncertML 2.0 in the context of other standards, particularly Observations and Measurements 2.0. The document concludes with a summary of the work, and a speculation on the future directions.

2 UncertML version 1.0

The original version of UncertML (version 1.0) took the form of a single XML schema which was submitted to OGC for consideration as a standard[3]. It was designed to allow the interoperable encoding of statistical summaries of uncertainty based on probabilistic approaches.

The design of this schema was weak-typed: that is, it addressed the wide variety of uncertainty representations which might be required by allowing users to define their own statistics and distributions. This approach aimed to allow flexibility by, for example, allowing a user to characterise their uncertainty with a 'Distribution' type, whose specific description and details (including mathematical characteristics) might be defined in their own dictionary. For this reason, the original set of elements defined within UncertML 1.0 was relatively small, and grouped into three major classes: Statistics, Distributions and Realisations. Many commonly-used uncertainty representations could be supported by the use of a Statistic or Distribution element in combination with a reference to the UncertML 1.0 dictionary[4] However, some commonly-used statistics and distributions were defined as explicit elements, and these are described below.

2.1 Explicit types supported in UncertML 1.0

2.1.1 Statistics

Quantile — points taken at regular intervals from the cumulative distribution function (CDF) of a random variable.

Moment — a measure of the shape of the probability distribution of a real-valued random variable, such as skewness or kurtosis.

DiscreteProbability — the probability that a variable has a specific discrete value.

Probability — the probability that a variable exceeds (or does not exceed) a certain threshold.

[3] `http://portal.opengeospatial.org/files/?artifact_id=33234`
[4] `http://dictionary.uncertml.org/dictionary.php`

UncertML 1.0 encoded other common statistics such as mean, median and standard deviation as generic Statistics with appropriate dictionary references. Elements were provided for collecting together statistical summary data, namely the StatisticsSet and the StatisticsArray (the latter used for multiple observations).

2.1.2 Distributions

All distributions in UncertML 1.0 were encoded as a Distribution type with appropriate dictionary references. Elements were provided for collecting together distribution information in useful ways, namely the MixtureModel and the DistributionArray (the latter used for multiple observations).

2.1.3 Realisations

The Realisations type allowed a user to collect a set of samples from a random quantity, allowing uncertainty to be described implicitly. Essentially realisations allowed the user to group together a set of samples that provide a discrete representation of the probability distribution for a given variable. Typically in usage a kernel smoother based density estimate might be used to produce plots of the probability density.

3 Requirements for UncertML 2.0

The requirements for UncertML 2.0 are defined in UncertWeb deliverable D1.1[5]. Essentially these requirements emphasise the need for precise definitions of uncertainty and defined the scope of these definitions in terms of support for probability distributions, statistics and realisations. The requirements emphasised usability and this motivated the change to a strongly typed design. The summary of the requirements, taken from D1.1, is given below.

Functional Type requirements.

- **FTR1: UncertML should provide a mechanism for quantifying statistics using proportions where applicable.**
- **FTR2: UncertML should include explicit types for recording standard deviation and variance.**
- **FTR3: UncertML should retain the Realisations element type included in version 1.0.**
- **FTR4: UncertML should include an explicit GaussianDistribution type.**
- **FTR5: UncertML should retain the explicit DiscreteProbability statistic type.**
- **FTR6: UncertML should allow the communication of standard errors of mean and variance.**

[5] `http://www.uncertweb.org/documents/deliverables`

- **FTR7: UncertML should provide the necessary type to encode a Dirichlet distribution.**
- **FTR8: UncertML should provide a type to represent Ensembles.**
- **FTR9: UncertML should encode common statistics and distributions sufficient to allow the handling of all reasonable sources of uncertainty which will arise during the course of the project - these are listed in FTR11, FTR12, FTR14 and FTR15.**
- **FTR10: The core Statistics in UncertML should be Mean, Mode, Median, Standard Deviation and Variance(FTR2), Quantile (already present in v1.0), Skewness, Probability and Discrete Probability (FTR5)**
- **FTR11: The Statistics included in the extension of UncertML should be Credible Interval, Correlation, Decile, Quartile, Percentile, Inter-quartile Range, Kurtosis, Moment, and Range**
- **FTR12: UncertML should provide a means of grouping statistics into a logical structure.**
- **FTR13: The core Distributions in UncertML should be Gaussian (FTR4), Dirichlet (FTR7), Log-normal, Exponential, Gamma, Multivariate Gaussian, Uniform, Student T, Poisson and Binomial.**
- **FTR14: The Distributions included in the extension of UncertML should be Beta, Laplace, Cauchy, Chi-square, Weibull, Logistic and Geometric.**
- **FTR15: UncertML should provide a mechanism for quantifying conditional distributions, to be used for example in distributions where the parameters are uncertain.**

Non-functional Requirements.

- **NFR1: UncertML will permit the communication of uncertainty information through a standardised and universally-accessible conceptual model.**
- **NFR2: UncertML should be split into a 'core' and 'extension' model with the common statistics and distributions implemented in the core and the less common implemented as an extension.**
- **NFR3: UncertML should use a strong-typed approach, where selected entities are explicitly modelled by the definition of explicit types as appropriate.**
- **NFR4: UncertML should remain domain-agnostic and *not* include specific metadata for concepts such as space, time or phenomena.**
- **NFR5: UncertML should be structured as a vocabulary and conceptual model which will support multiple implementations in different languages.**
- **NFR6: UncertML should provide multiple encodings (e.g., XML and binary) to allow for efficient transport of large raster datasets.**

- **NFR7: UncertML should provide a means to encode all statistics and distributions for categorical, discrete and continuous data, where applicable.**

4 Changes from version 1.0 to version 2.0

The main change from version 1.0 to version 2.0 is the move to a strongly typed design. This allows us to create a complete Application Programmer Interface (API) for UncertML 2.0, allowing users to confidently state that their applications are interoperable with UncertML 2.0, something not possible with weak typed designs. It also allows the development of a complete dictionary.

The implementation of a strong-typed design also allows constraints to be placed on each UncertML 2.0 type. For example, it is possible to stipulate that every value in the `Variance` type must be positive. These constraints are enforced at the schema level and can therefore be validated. This functionality is not possible in a weak-typed design without resorting to third-party solutions such as Schematron.

5 UncertML 2.0 conceptual model

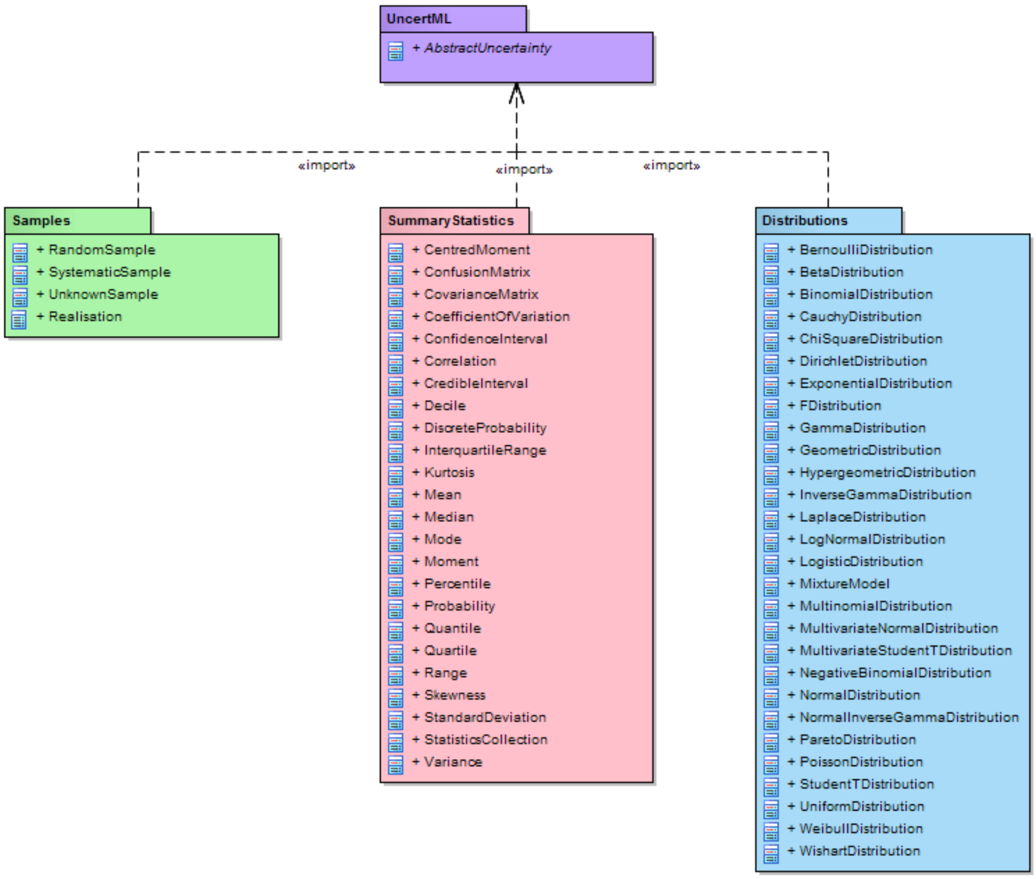

Figure 1: UML diagram outlining the hierarchy in UncertML 2.0.

UncertML 2.0 maintains a similar design to version 1.0. Three distinct packages form the core: statistics, distributions and samples (Figure 1). However, in version 1.0 each package only included a few basic types. For instance, the statistics package included a `Statistic` type and several specialisations including `Quantile` and `Probability`. Each package within version 2.0 contains a richer set of types as a consequence of the hard-typed design. Each package contains an abstract super-type that every hard-type extends, in turn these abstract super-types extend the UncertML root element: `AbstractUncertainty`. This substitutability chain (`AbstractUncertainty` — `AbstractStatisitic` — `Mean`, for example) provides a logical grouping of elements that can be used in aggregate types such as the `MixtureModel` and `StatisticsCollection` types. These abstract types provide an extensibility point for users to extend the default functionality of UncertML 2.0 with additional elements to meet their requirements, although we anticipate the core UncertML 2.0 types will be sufficient for all UncertWeb applications and most applications beyond UncertWeb.

6 UncertML 2.0 user guide

Appendix A provides a thorough description of every type in UncertML 2.0. This 'user guide' is split into three categories: statistics, distributions & samples. Each element within these categories has an information box comprising of the following details:

URI
: A string that uniquely identifies that element. This is primarily used to refer to UncertML 'concepts' from outside an UncertML 2.0 encoding, for instance within the SWE Common `Quantity` type.

UncertML name
: The unique name used within UncertML 2.0 encodings to represent that element.

Alternative names
: A list of names by which that element is also commonly referred to. These names are *never* used within UncertML 2.0 encodings — only the UncertML name may be used.

Definition
: A mathematical definition that describes the concepts encapsulated by that element. This is not a complete formal definition but is sufficient to uniquely identify the element.

Parameters
: Any parameters that are required in order to encode that element — both the mathematical and UncertML 2.0 notation is provided, so there is a clear linking between the definition and schema. Any constraints on the parameters are also noted.

Source
: The reference material used to write the definition. If blank it is assumed the element is sufficiently well-understood that no formal linkage to a source is needed.

Categories
: Tags that are relevant to that element, which in the future will allow users to query for elements in UncertML 2.0 suitable for a given task.

Further information

References to additional information on that element, should it be required. These are meant to be informative not normative, so will typically link to external web sites, or other resources.

Schema

The complete UncertML 2.0 XML schema fragment for that element.

XML

XML examples for that element. Multiple examples are provided where appropriate. These examples are not meant to show realistic values, but how the XML instance document might look.

JSON

JSON examples for that element. Multiple examples are provided where appropriate.

Java API

Java code demonstrating how to instantiate, parse and encode that element using the prototype UncertML 2.0 Java API.

Value constraints

Any constraints on the values and parameters of that element. These are enforced at the schema and API level.

Further to these details, all distribution types have the following additional details:

Support

The range of values over which the distribution is valid, for example the positive real numbers.

PDF

The mathematical representation of that distribution's probability density function for continuous variables, and probability mass function for discrete (ordinal and categorical) variables.

Each UncertML type allows the encoding of both single and multiple values. For instance, every statistic type in UncertML 2.0 has a `values` element that is an XML list type that allows multiple space-separated values to be encoded. This flexibility allows the user to encode a single value (e.g., the variance of a variable at a single location), or multiple values (e.g., the variance of a variable over a spatial field) in the same data structure. Examples of both use cases are provided in the user guide where applicable.

The exception to this rule is when using multivariate distributions. Every multivariate type in UncertML encodes only a single instance. For example, it is possible to encode a multivariate normal distribution of a variable over a spatial field with a single encoding. However, if the user wishes to encode two multivariate variables over the same field, two separate UncertML elements must be used. This restriction is in place to ensure the data contained within the values element can be reliably unpacked within the API.

This user guide provides the complete reference to UncertML 2.0, and is available and maintained on the web, currently at: `http://wiki.aston.ac.uk/UncertWeb/UncertMLDictionary`. In the future the user guide will be migrated to the UncertML web site[6].

7 UncertML 2.0 in wider usage

UncertML 2.0 is not typically intended to be used in isolation, but in conjunction with other information models. In particular UncertML 2.0 is designed to replace some of the base numeric data types (float, integer, arrays of those), with their equivalent uncertain types. This section gives a brief overview of recommendations for using UncertML 2.0 within the OGC information models being used in UncertWeb.

7.1 UncertML 2.0 and data quality

ISO19115 and the extensions to imagery and gridded data are the most relevant to issues of the representation of quality information. ISO 19115 is comprehensive and provides clear guidance on the metadata elements that constitute useful descriptors for data quality. This analysis will only consider those elements associated with data quality. These include information on:

- Lineage - the source of the data (LI_Source), and processing applied to the data (LI_ProcessingStep)
- Completeness - how complete the data set is (DQ_Completeness)
- Logical consistency - this is particularly relevant to geospatial data, where things like topological consistency are considered important (DQ_LogicalConsistency)
- Positional accuracy - is it located correctly in space (DQ_PositionalAccuracy)
- Thematic accuracy - is the thematic information correct (DQ_ThematicAccuracy)
- Temporal accuracy - how accurate is the time information (DQ_TemporalAccuracy)

A scope element allows this metadata to apply only to sub-elements of the data set, down to object level it seems. However in practice metadata typically describes data set level scope, that is it summarises properties of the whole dataset which is assumed to be homogeneous, or at least the properties are assumed to represent average properties at the data set level. The only data quality element that is mandatory in the core metadata profile in ISO 19115 is the lineage information.

The `DQ_Element` types (all except lineage) have a common format, containing:

- nameOfMeasure [0..*] : CharacterString
- measureIdentification [0..1] : MD_Identifier - this identifies the thing that is being measured

[6] `http://www.uncertml.org`

- measureDescription [0..1] : CharacterString
- evaluationMethodType [0..1] : DQ_EvaluationMethodTypeCode - this can be: directInternal, directExternal or indirect
- evaluationMethodDescription [0..1] : CharacterString
- evaluationProcedure [0..1] : CI_Citation - this links to documentation that defines / legitimises the method used to compute the evaluation measure.
- dateTime [0..*] : DateTime - at which the metadata was calculated
- result [1..2] : DQ_Result - this can be a quantitative result (essentially a measure of uncertainty), or a conformance result (pass / fail as specific test / threshold).

UncertML 2.0 could be used within the the `DQ_QuantitativeResult` type which can be substituted for the `result` and contains:

- valueType [0..1] : RecordType
- valueUnit : UnitOfMeasure
- errorStatistic [0..1] : CharacterString
- value [1..*] : Record

The main issue with the ISO 19115 implementation is its flexibility - which is mirrored in the XML implementation (ISO 19139). It essentially allows the user to specify pretty much any measure in the `DQ_Element`, and then since the `errorStatistic` in the `DQ_QuantitativeResult` is also a string this could be anything. Such flexibility can be seen as assisting the deployment of this standard - basically users can all keep using whatever language they like to describe the `errorStatistics` they are using to characterise the quality of the data, which does not allow machine level interoperability. A minimal integration of UncertML 2.0 into the ISO standards could be achieved by simply using an UncertML 2.0 type in the `errorStatistic` string using the UncertML 2.0 identifier. This only exploits the controlled vocabulary (dictionary) associated with UncertML 2.0, which brings the benefits of using a complete and supported set of error statistics. A more complete integration is discussed next, because the record type used for the actual `value` is not as flexible as the UncertML 2.0 encodings which are more explicit and supported by the API.

7.2 UncertML 2.0, O&M 2.0 and NetCDF

In O&M 2.0, the result quality element of an O&M 2.0 observation is of type `DataQuality` as defined in ISO 19139. In order to integrate uncertainty information encoded as UncertML 2.0 in the result quality of an observation, we defined a `DQ_UncertaintyResult_Type` as a subtype of the `Abstract_DQ_Result_Type`. This enables the integration uncertainty information in the ISO `DataQuality` model as a quantitative data quality result. Longer term we believe it makes sense to integrate the UncertML 2.0 dictionary with the ISO19157 (Geographic Information - Data quality) definitions in the data quality measures section, however this is on a longer time scale.

In many of the UncertWeb applications it is not that there are single results and associated quality information, but the results themselves are uncertain. Thus we have also created an `OM_UncertaintyObservation` subtype which allows the users of O&M 2.0 to specify a result that is an UncertML 2.0 type. In this way it is possible to use UncertML 2.0 very easily in the UncertWeb O&M 2.0 profiles, examples of which can be found on `http://wiki.aston.ac.uk/UncertWeb/OandMProfiles`.

Within UncertWeb, UncertML 2.0 will also be used in NetCDF encodings. The approach is a very simple one of essentially using the UncertML 2.0 dictionary to define the meaning of layers within a NetCDF file. The development of the NetCDF encodings for uncertainty forms part of the UncertWeb API that is currently under development and will be reported on in UncertWeb deliverable D8.2.

8 Summary

UncertML 2.0 is designed to be simple and easy to use. It is domain agnostic and can be readily used within any other schema with minimal complications. The hard typed design makes it possible to build a complete API that supports all UncertML 2.0 types. The hard typed nature also means that it is possible for applications to interoperate by fully supporting UncertML 2.0, something that can never be possible with weak typed designs, at least in any meaningful (beyond parsing) manner. The range of statistics and distributions supported has been increased in UncertML 2.0, to include all types required within the UncertWeb project, and a range of widely used types from the field of statistics and probabilistic modelling.

Some things are missing from UncertML 2.0. Specifically support for random functions, and more complex conditional probability distributions is something that we believe could be of value to add, possibly in other schema associated with UncertML, but not integrated with UncertML 2.0, to allow UncertML 2.0 to maintain its simplicity. Another open issue remains the exact mechanism for exposing the UncertML 2.0 dictionary. The definitions are complete and in existence and the trend in the web community seems to be toward using SKOS[7] (Simple Knowledge Organization System) as a method of exposing controlled vocabularies. We are currently investigating the options for deploying a SKOS solution, and exploring the appropriate governance model for UncertML 2.0.

[7] `http://www.w3.org/2004/02/skos/`

Appendix A: The complete UncertML 2.0 dictionary

The following pages are taken from `http://wiki.aston.ac.uk/UncertWeb/UncertMLDictionary` and present the on-line dictionary and user guide for UncertML 2.0. The structure of the document is explained in Section 6.

Statistics

Centred moment

URI	http://www.uncertml.org/statistics/centred-moment
UncertML name	CentredMoment
Alternative names	N/A
Definition	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.
Parameters	k (`order`), the order of the moment
Source	N/A
Categories	summary statistic, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/CentralMoment.html

Schema

```
<!-- Element -->
<xs:element name="CentredMoment"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:CentredMomentType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="CentredMomentType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="order" type="un:naturalNumber"
use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:CentredMoment order="2" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:CentredMoment>

<!-- Multiple values -->
<un:CentredMoment order="2" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:CentredMoment>
```

JSON

```
// Single value
{"CentredMoment": {"order":2, "values": [3.14]}}

// Multiple values
{"CentredMoment": {"order":2, "values": [3.14, 6.28, 9.42]}}
```

Java API	<pre>// Single value declaration CentredMoment cm = new CentredMoment(2, 3.14); // Multiple value declaration CentredMoment cm = new CentredMoment(2, new double[] {3.14, 6.28, 9.42}); // Parsing from an XML file XMLParser xml = new XMLParser(); CentredMoment cm = (CentredMoment)xml.parse(new File("centred- moment.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); CentredMoment cm = (CentredMoment)json.parse(new File("centred- moment.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(cm, new File("centred-moment.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(cm, new File("centred-moment.json"));</pre>
Value constraints	**order**: *any natural number* **values**: *any real number*

Coefficient of variation

URI	http://www.uncertml.org/statistics/coefficient-of-variation
UncertML name	[CoefficientOfVariation]()
Alternative names	N/A
Definition	For a random variable with mean μ and strictly positive standard deviation σ, the coefficient of variation is defined as the ratio $\frac{\sigma}{\lvert\mu\rvert}$. One benefit of using the coefficient of variation rather than the standard deviation is that it is unitless.
Parameters	N/A
Source	N/A
Categories	summary statistic, dispersion, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/VariationCoefficient.html
Schema	<pre><!-- Element --> <xs:element name="CoefficientOfVariation" substitutionGroup="un:AbstractSummaryStatistic"> <xs:complexType> <xs:complexContent> <xs:extension base="un:CoefficientOfVariationType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="CoefficientOfVariationType"> <xs:complexContent> <xs:extension base="un:AbstractSummaryStatisticType"> <xs:sequence> <xs:element name="values" type="un:ContinuousValuesType"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType></pre>
XML	

	`` <!-- Single value --> <un:CoefficientOfVariation xmlns:un="http://www.uncertml.org/2.0"> <un:values>3.14</un:values> </un:CoefficientOfVariation> <!-- Multiple values --> <un:CoefficientOfVariation xmlns:un="http://www.uncertml.org/2.0"> <un:values>3.14 6.28 9.42</un:values> </un:CoefficientOfVariation> ```
JSON	``` // Single value {"CoefficientOfVariation": {"values": [3.14]}} // Multiple values {"CoefficientOfVariation": {"values": [3.14, 6.28, 9.42]}} ```
Java API	``` // Single value declaration CoefficientOfVariation cov = new CoefficientOfVariation(3.14); // Multiple value declaration CoefficientOfVariation cov = new CoefficientOfVariation(new double[] {3.14, 6.28, 9.42}); // Parsing from an XML file XMLParser xml = new XMLParser(); CoefficientOfVariation cov = (CoefficientOfVariation)xml.parse(new File("coefficient-of-variation.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); CoefficientOfVariation cov = (CoefficientOfVariation)json.parse(new File("coefficient-of-variation.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(cov, new File("coefficient-of-variation.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(cov, new File("coefficient-of-variation.json")); ```
Value constraints	**values**: *any real number*

Confidence interval

URI	http://www.uncertml.org/statistics/confidence-interval
UncertML name	[ConfidenceInterval]()
Alternative names	N/A
Definition	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. UncertML

	allows you the flexibility to specify non-symmetric confidence intervals however in practice we would expect the main usage to be for symmetric intervals and this will be addressed in the API.
Parameters	a (`lower level`) - the lower quantile in the range [0, 1] b (`upper level`) - the upper quantile in the range [0, 1]
Source	N/A
Categories	summary statistic, dispersion, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/ConfidenceInterval.html

Schema

```
<!-- Element -->
<xs:element name="ConfidenceInterval"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ConfidenceIntervalType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ConfidenceIntervalType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="lower" type="un:QuantileType"/>
        <xs:element name="upper" type="un:QuantileType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:ConfidenceInterval xmlns:un="http://www.uncertml.org/2.0">
  <un:lower level="0.05">
    <un:values>3.14</un:values>
  </un:lower>
  <un:upper level="0.95">
    <un:values>6.28</un:values>
  </un:upper>
</un:ConfidenceInterval>

<!-- Multiple values -->
<un:ConfidenceInterval xmlns:un="http://www.uncertml.org/2.0">
  <un:lower level="0.05">
    <un:values>3.14 6.28 9.42</un:values>
  </un:lower>
  <un:upper level="0.95">
    <un:values>6.28 12.57 18.85</un:values>
  </un:upper>
</un:ConfidenceInterval>
```

JSON

```
// Single value
{"ConfidenceInterval": {"lower": {"level":0.05, "values": [3.14]},
"upper": {"level":0.95, "values": [6.28]}}}

// Multiple values
{"ConfidenceInterval": {"lower": {"level":0.05, "values": [3.14,
6.28, 9.42]}, "upper": {"level":0.95, "values": [6.28, 12.57,
18.85]}}}
```

Java API

```
// Single value declaration
ConfidenceInterval ci = new ConfidenceInterval(new Quantile(0.05,
3.14),  new Quantile(0.95,  6.28));

// Multiple value declaration
ConfidenceInterval ci = new ConfidenceInterval(new Quantile(0.05,
new double[] {3.14,  6.28,  9.42}),  new Quantile(0.95,  new
double[] {6.28,  12.57,  18.85}));

// Parsing from an XML file
XMLParser xml = new XMLParser();
```

```
ConfidenceInterval ci = (ConfidenceInterval)xml.parse(new
File("confidence-interval.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
ConfidenceInterval ci = (ConfidenceInterval)json.parse(new
File("confidence-interval.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(ci,  new File("confidence-interval.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(ci,  new File("confidence-interval.json"));
```

Value constraints	**lower level**: *any real number in the range [0 - 1] (inclusive)* **upper level**: *any real number in the range [0 - 1] (inclusive)* **values** *any real number*

Confusion matrix

URI	http://www.uncertml.org/statistics/confusion-matrix
UncertML name	ConfusionMatrix
Alternative names	Matching matrix
Definition	Consider a classification model which assigns one category out of n to its input, for instance a model which assigns one of the 26 alphabetical characters ('a', 'b', 'c'...) to an image of a handwritten character. The classification matrix is an $n \times n$ matrix which matches the true category (rows) against the prediction (columns). The matrix element (i, j) indicates the number of occurrences where the model assigned the category j to an element actually belonging to category i. Correct predictions thus appear on the diagonal of the matrix (where $i = j$). The confusion matrix is typically used as a diagnostic which shows whether and how misclassification (i.e. incorrect predictions) happen.
Parameters	N/A
Source	N/A
Categories	summary statistics, classification, categorical variables, ordinal variables
Further information	http://en.wikipedia.org/wiki/Confusion_matrix
Schema	(see below)

```
<!-- Element -->
<xs:element name="ConfusionMatrix"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ConfusionMatrixType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ConfusionMatrixType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="sourceCategories"
type="un:CategoricalValuesType"/>
        <xs:element name="targetCategories"
type="un:CategoricalValuesType"/>
        <xs:element name="counts"
type="un:PositiveNaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Multiple values -->
<un:ConfusionMatrix xmlns:un="http://www.uncertml.org/2.0">
  <un:sourceCategories>Red Green Blue</un:sourceCategories>
  <un:targetCategories>Red Green Blue</un:targetCategories>
  <un:counts>1 2 3 4 5 6 7 8 9</un:counts>
</un:ConfusionMatrix>
```

JSON

```json
// Multiple values
{"ConfusionMatrix": {"sourceCategories": ["Red", "Green", "Blue"],
"targetCategories": ["Red", "Green", "Blue"], "counts": [1, 2, 3, 4,
5, 6, 7, 8, 9]}}
```

Java API

```java
// Multiple value declaration
ConfusionMatrix cm = new ConfusionMatrix(new String[] {"Red",
"Green",  "Blue"},  new String[] {"Red",  "Green",  "Blue"},  new
int[] {1,  2,  3,  4,  5,  6,  7,  8,  9});

// Parsing from an XML file
XMLParser xml = new XMLParser();
ConfusionMatrix cm = (ConfusionMatrix)xml.parse(new File("confusion-
matrix.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
ConfusionMatrix cm = (ConfusionMatrix)json.parse(new
File("confusion-matrix.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(cm,  new File("confusion-matrix.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(cm,  new File("confusion-matrix.json"));
```

Value constraints

categories: *any string*
probabilities: *any real number in the range [0-1] (inclusive)*

Correlation

URI	http://www.uncertml.org/statistics/correlation
UncertML name	Correlation
Alternative names	Correlation coefficient
Definition	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.
Parameters	N/A
Source	N/A
Categories	summary statistic, multivariate, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/Correlation.html,

	http://mathworld.wolfram.com/CorrelationCoefficient.html

Schema

```
<!-- Element -->
<xs:element name="Correlation"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:CorrelationType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="CorrelationType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:NormalisedValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Correlation xmlns:un="http://www.uncertml.org/2.0">
  <un:values>-1.0</un:values>
</un:Correlation>

<!-- Multiple values -->
<un:Correlation xmlns:un="http://www.uncertml.org/2.0">
  <un:values>-1.0 0.0 1.0</un:values>
</un:Correlation>
```

JSON

```
// Single value
{"Correlation": {"values": [-1.0]}}

// Multiple values
{"Correlation": {"values": [-1.0, 0.0, 1.0]}}
```

Java API

```
// Single value declaration
Correlation c = new Correlation(-1.0);

// Multiple value declaration
Correlation c = new Correlation(new double[] {-1.0,  0.0,  1.0});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Correlation c = (Correlation)xml.parse(new File("correlation.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Correlation c = (Correlation)json.parse(new
File("correlation.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(c,  new File("correlation.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(c,  new File("correlation.json"));
```

Value constraints	**value** *any real number in the range [-1.0 - 1.0] (inclusive)*

Covariance matrix

URI	http://www.uncertml.org/statistics/covariance-matrix
UncertML name	CovarianceMatrix

Alternative names	Variance-covariance matrix, Variance matrix
Definition	Given d univariate random variables $(x_1, ..., x_d)$, the covariance matrix of $(x_1, ..., x_d)$ is the $d \times d$ matrix C whose elements $C_{i,j}$ are the covariance between x_i and x_j. That is, $C_{i,j} = E[(x_i - E[x_i])(x_j - E[x_j])]$. In particular, the diagonal elements $C_{i,i}$ are the variances of the x_i. The matrix is stored in row major format in UncertML. It is possible that future versions of UncertML will have specialised types for simpler covariance matrices, such as sparse, outer product, block and n-diagonal matrix types.
Parameters	d (`dimension`) the number of random variables
Source	N/A
Categories	Summary statistic, Multivariate
Further information	http://mathworld.wolfram.com/CovarianceMatrix.html

Schema

```xml
<!-- Element -->
<xs:element name="CovarianceMatrix"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:CovarianceMatrixType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="CovarianceMatrixType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="dimension" type="un:naturalNumber"
use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Multiple values -->
<un:CovarianceMatrix dimension="2"
xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 0.0 0.0 3.14</un:values>
</un:CovarianceMatrix>
```

JSON

```json
// Multiple values
{"CovarianceMatrix": {"dimension":2, "values": [3.14, 0.0, 0.0,
3.14]}}
```

Java API

```java
// Multiple value declaration
CovarianceMatrix cm = new CovarianceMatrix(2,  new double[] {3.14,
0.0,  0.0,  3.14});

// Parsing from an XML file
XMLParser xml = new XMLParser();
CovarianceMatrix cm = (CovarianceMatrix)xml.parse(new
File("covariance-matrix.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
CovarianceMatrix cm = (CovarianceMatrix)json.parse(new
File("covariance-matrix.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(cm,  new File("covariance-matrix.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(cm,  new File("covariance-matrix.json"));
```

Value constraints	**dimension**: *any natural number* **values**: *any real number*

Credible interval

URI	http://www.uncertml.org/statistics/credible-interval
UncertML name	CredibleInterval
Alternative names	Bayesian confidence interval
Definition	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.
Parameters	a (`lower level`) - the lower quantile in the range [0, 1] b (`upper level`) - the upper quantile in the range [0, 1]
Source	N/A
Categories	Summary statistic, Bayesian
Further information	http://en.wikipedia.org/wiki/Credible_interval
Schema	see below

```
<!-- Element -->
<xs:element name="CredibleInterval"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:CredibleIntervalType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="CredibleIntervalType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="lower" type="un:QuantileType"/>
        <xs:element name="upper" type="un:QuantileType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:CredibleInterval xmlns:un="http://www.uncertml.org/2.0">
  <un:lower level="0.05">
    <un:values>3.14</un:values>
  </un:lower>
  <un:upper level="0.95">
    <un:values>6.28</un:values>
  </un:upper>
</un:CredibleInterval>

<!-- Multiple values -->
<un:CredibleInterval xmlns:un="http://www.uncertml.org/2.0">
  <un:lower level="0.05">
    <un:values>3.14 6.28 9.42</un:values>
  </un:lower>
  <un:upper level="0.95">
    <un:values>6.28 12.57 18.85</un:values>
```

	`` </un:upper> </un:CredibleInterval> ```
JSON	``` // Single value {"CredibleInterval": {"lower": {"level":0.05, "values": [3.14]}, "upper": {"level":0.95, "values": [6.28]}}} // Multiple values {"CredibleInterval": {"lower": {"level":0.05, "values": [3.14, 6.28, 9.42]}, "upper": {"level":0.95, "values": [6.28, 12.57, 18.85]}}} ```
Java API	``` // Single value declaration CredibleInterval ci = new CredibleInterval(new Quantile(0.05, 3.14), new Quantile(0.95, 6.28)); // Multiple value declaration CredibleInterval ci = new CredibleInterval(new Quantile(0.05, new double[] {3.14, 6.28, 9.42}), new Quantile(0.95, new double[] {6.28, 12.57, 18.85})); // Parsing from an XML file XMLParser xml = new XMLParser(); CredibleInterval ci = (CredibleInterval)xml.parse(new File("credible-interval.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); CredibleInterval ci = (CredibleInterval)json.parse(new File("credible-interval.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(ci, new File("credible-interval.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(ci, new File("credible-interval.json")); ```
Value constraints	**lower level**: *any real number in the range [0 - 1] (inclusive)* **upper level**: *any real number in the range [0 - 1] (inclusive)* **values** *any real number*

Decile

URI	http://www.uncertml.org/statistics/decile
UncertML name	Decile
Alternative names	N/A
Definition	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.
Parameters	d (`level`) denotes the decile to be considered, in the range [1 - 9]
Source	N/A
Categories	Descriptive statistics, Ordinal variables, Continuous variables
Further information	http://en.wikipedia.org/wiki/Decile
Schema	

```
<!-- Element -->
<xs:element name="Decile"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:DecileType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="DecileType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="level" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"/>
            <xs:maxInclusive value="9"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Decile level="5" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Decile>

<!-- Multiple values -->
<un:Decile level="5" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Decile>
```

JSON

```
// Single value
{"Decile": {"level":0.05, "values": [3.14]}}

// Multiple values
{"Decile": {"level":0.05, "values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
Decile d = new Decile(5,  3.14);

// Multiple value declaration
Decile d = new Decile(5,  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Decile d = (Decile)xml.parse(new File("decile.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Decile d = (Decile)json.parse(new File("decile.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(d,  new File("decile.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(d,  new File("decile.json"));
```

Value constraints

level: *any natural number in the range [1-9] (inclusive)*
values: *any real number*

Discrete probability

URI	http://www.uncertml.org/statistics/discrete-probability
UncertML name	DiscreteProbability

Alternative names	Probability
Definition	Given a random variable x which takes values in a finite number of categories or classes (e.g. x is a colour and can be either 'red, 'green' or 'blue'), the discrete probability of a particular category (e.g. 'red') is the probability that x belongs to that category (e.g. $P(x = red)$). The discrete probabilities for all possible categories a variable can take should sum to 1.
Parameters	N/A
Source	N/A
Categories	Probability
Further information	http://en.wikipedia.org/wiki/Discrete_probability_distribution, http://mathworld.wolfram.com/DiscreteDistribution.html

Schema

```
<!-- Element -->
<xs:element name="DiscreteProbability"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:DiscreteProbabilityType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="DiscreteProbabilityType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="categories">
          <xs:complexType>
            <xs:simpleContent>
              <xs:restriction base="un:CategoricalValuesType"/>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="probabilities"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:DiscreteProbability xmlns:un="http://www.uncertml.org/2.0">
  <un:categories>Red</un:categories>
  <un:probabilities>0.1</un:probabilities>
</un:DiscreteProbability>

<!-- Multiple values -->
<un:DiscreteProbability xmlns:un="http://www.uncertml.org/2.0">
  <un:categories>Red Green Blue</un:categories>
  <un:probabilities>0.1 0.2 0.3</un:probabilities>
</un:DiscreteProbability>
```

JSON

```
// Single value
{"DiscreteProbability": {"categories": ["Red"], "probability":
[0.1]}}

// Multiple values
{"DiscreteProbability": {"categories": ["Red", "Green", "Blue"],
"probability": [0.1, 0.2, 0.3]}}
```

Java API

```
// Single value declaration
DiscreteProbability dp = new DiscreteProbability("Red",  0.1);

// Multiple value declaration
DiscreteProbability dp = new DiscreteProbability(new String[]
{"Red",  "Green",  "Blue"},  new double[] {0.1,  0.2,  0.3});
```

```
// Parsing from an XML file
XMLParser xml = new XMLParser();
DiscreteProbability dp = (DiscreteProbability)xml.parse(new
File("discrete-probability.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
DiscreteProbability dp = (DiscreteProbability)json.parse(new
File("discrete-probability.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(dp,  new File("discrete-probability.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(dp,  new File("discrete-probability.json"));
```

Value constraints

categories: *any string*
probabilities: *any real number in the range [0 - 1] (inclusive)*

Interquartile range

URI	http://www.uncertml.org/statistics/interquartile-range
UncertML name	InterquartileRange
Alternative names	IQR
Definition	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.
Parameters	N/A
Source	N/A
Categories	summary statistic, dispersion, robust
Further information	http://mathworld.wolfram.com/InterquartileRange.html

Schema

```
<!-- Element -->
<xs:element name="InterquartileRange"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:InterquartileRangeType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="InterquartileRangeType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="lower" type="un:ContinuousValuesType"/>
        <xs:element name="upper" type="un:ContinuousValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:InterquartileRange xmlns:un="http://www.uncertml.org/2.0">
  <un:lower>3.14</un:lower>
  <un:upper>6.28</un:upper>
</un:InterquartileRange>

<!-- Multiple values -->
<un:InterquartileRange xmlns:un="http://www.uncertml.org/2.0">
  <un:lower>3.14 6.28 9.42</un:lower>
```

	``<un:upper>6.28 12.57 18.85</un:upper></un:InterquartileRange>```
JSON	```// Single value {"InterquartileRange": {"lower": [3.14], "upper": [6.28]}} // Multiple values {"InterquartileRange": {"lower": [3.14, 6.28, 9.42], "upper": [6.28, 12.57, 18.85]}}```
Java API	```// Single value declaration InterquartileRange ir = new InterquartileRange(); // Multiple value declaration InterquartileRange ir = new InterquartileRange(); // Parsing from an XML file XMLParser xml = new XMLParser(); InterquartileRange ir = (InterquartileRange)xml.parse(new File("interquartile-range.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); InterquartileRange ir = (InterquartileRange)json.parse(new File("interquartile-range.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(ir, new File("interquartile-range.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(ir, new File("interquartile-range.json"));```
Value constraints	**lower** *any real number* *upper* *any real number*

Kurtosis

URI	http://www.uncertml.org/statistics/kurtosis
UncertML name	Kurtosis
Alternative names	N/A
Definition	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 4-th centred moment of the distribution and σ is its standard deviation.
Parameters	N/A
Source	N/A
Categories	summary statistic, ordinal variables, continuous variables, dispersion
Further information	http://mathworld.wolfram.com/Kurtosis.html
Schema	```<!-- Element --> <xs:element name="Kurtosis" substitutionGroup="un:AbstractSummaryStatistic"> <xs:complexType> <xs:complexContent> <xs:extension base="un:KurtosisType"/> </xs:complexContent>```

```
    </xs:complexType>
  </xs:element>

  <!-- Complex type -->
  <xs:complexType name="KurtosisType">
    <xs:complexContent>
      <xs:extension base="un:AbstractSummaryStatisticType">
        <xs:sequence>
          <xs:element name="values" type="un:KurtosisValuesType"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

XML

```
<!-- Single value -->
<un:Kurtosis xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Kurtosis>

<!-- Multiple values -->
<un:Kurtosis xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Kurtosis>
```

JSON

```
// Single value
{"Kurtosis": {"values": [3.14]}}

// Multiple values
{"Kurtosis": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
Kurtosis k = new Kurtosis(3.14);

// Multiple value declaration
Kurtosis k = new Kurtosis(new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Kurtosis k = (Kurtosis)xml.parse(new File("kurtosis.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Kurtosis k = (Kurtosis)json.parse(new File("kurtosis.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(k,  new File("kurtosis.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(k,  new File("kurtosis.json"));
```

Value constraints	**value** *any real number >= -2*

Mean

URI	http://www.uncertml.org/statistics/mean
UncertML name	Mean
Alternative names	arithmetic mean, average, expectation
Definition	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 i'th 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 UncertML however does not deal with issues of sample size, rather the

	mean is taken to refer to the population mean.
Parameters	N/A
Source	N/A
Categories	summary statistic, central tendency, ordinal variables, continuous variables
Further information	http://en.wikipedia.org/wiki/Arithmetic_mean

Schema

```
<!-- Element -->
<xs:element name="Mean"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MeanType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MeanType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Mean xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Mean>

<!-- Multiple values -->
<un:Mean xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Mean>
```

JSON

```
// Single value
{"Mean": {"values": [3.14]}}

// Multiple values
{"Mean": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
Mean m = new Mean(3.14);

// Multiple value declaration
Mean m = new Mean(new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Mean m = (Mean)xml.parse(new File("mean.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Mean m = (Mean)json.parse(new File("mean.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(m,  new File("mean.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(m,  new File("mean.json"));
```

Value constraints	**value** *any real number*

Median

URI	http://www.uncertml.org/statistics/median
UncertML name	Median
Alternative names	N/A
Definition	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.
Parameters	N/A
Source	N/A
Categories	summary statistics, central tendency, ordinal variables, continuous variables, robust
Further information	http://en.wikipedia.org/wiki/Median

Schema

```
<!-- Element -->
<xs:element name="Median"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MedianType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MedianType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Median xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Median>

<!-- Multiple values -->
<un:Median xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Median>
```

JSON

```
// Single value
{"Median": {"values": [3.14]}}

// Multiple values
{"Median": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
Median m = new Median(3.14);

// Multiple value declaration
```

```
Median m = new Median(new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Median m = (Median)xml.parse(new File("median.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Median m = (Median)json.parse(new File("median.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(m,  new File("median.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(m,  new File("median.json"));
```

Value constraints	**value** *any real number*

Mode

URI	http://www.uncertml.org/statistics/mode
UncertML name	Mode
Alternative names	N/A
Definition	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.
Parameters	N/A
Source	N/A
Categories	summary statistics, central tendency, categorical variables, ordinal variables, continuous variables, robust
Further information	http://en.wikipedia.org/wiki/Mode_%28statistics%29

Schema

```
<!-- Element -->
<xs:element name="Mode"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ModeType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ModeType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:choice>
        <xs:element name="values" type="un:ContinuousValuesType"/>
        <xs:element name="categories"
type="un:CategoricalValuesType"/>
      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Mode xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Mode>

<un:Mode xmlns:un="http://www.uncertml.org/2.0">
  <un:categories>Red</un:categories>
```

```
</un:Mode>

<!-- Multiple values -->
<un:Mode xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Mode>

<un:Mode xmlns:un="http://www.uncertml.org/2.0">
  <un:categories>Red Green Blue</un:categories>
</un:Mode>
```

JSON

```
// Single value
{"Mode": {"values": [3.14]}}
{"Mode": {"categories": ["Red"]}}

// Multiple values
{"Mode": {"values": [3.14, 6.28, 9.42]}}
{"Mode": {"categories": ["Red", "Green", "Blue"]}}
```

Java API

```
// Single value declaration
Mode m = new Mode(3.14);
CategoricalMode cm = new CategoricalMode("Red");

// Multiple value declaration
Mode m = new Mode(new double[] {3.14,  6.28,  9.42});
CategoricalMode cm = new CategoricalMode(new String[] {"Red",
"Green",  "Blue"});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Mode m = (Mode)xml.parse(new File("mode.xml"));
CategoricalMode cm = (CategoricalMode)xml.parse(new
File("categorical-mode.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Mode m = (Mode)json.parse(new File("mode.json"));
CategoricalMode cm = (CategoricalMode)json.parse(new
File("categorical-mode.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(m,  new File("mode.xml"));
xEncoder.encode(cm,  new File("categorical-mode.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(m,  new File("mode.json"));
jEncoder.encode(cm,  new File("categorical-mode.json"));
```

Value constraints **value** *any real number* or **category** *any string*

Moment

URI	http://www.uncertml.org/statistics/moment
UncertML name	Moment
Alternative names	Raw moment, non-centred moment
Definition	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.
Parameters	k (`order`) The order of the moment
Source	N/A

Categories	summary statistic, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/Moment.html

Schema

```
<!-- Element -->
<xs:element name="Moment"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MomentType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MomentType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="order" type="un:naturalNumber"
use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Moment order="2" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Moment>

<!-- Multiple values -->
<un:Moment order="2" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Moment>
```

JSON

```
// Single value
{"Moment": {"order":2, "values": [3.14]}}

// Multiple values
{"Moment": {"order":2,  "values": [3.141,  3.141,  3.141]}}
```

Java API

```
// Single value declaration
Moment m = new Moment(2,  3.14);

// Multiple value declaration
Moment m = new Moment(2,  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Moment m = (Moment)xml.parse(new File("moment.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Moment m = (Moment)json.parse(new File("moment.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(m,  new File("moment.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(m,  new File("moment.json"));
```

Value constraints	**order**: *any natural number* **values**: *any real number*

Percentile

URI	http://www.uncertml.org/statistics/percentile

UncertML name	Percentile
Alternative names	centile
Definition	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 p'th 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 $x_p = x_1\ [\rho = 1];\ x_n\ [\rho = n]$.
Parameters	p (`level`) denotes the percentile to be considered, in the range [0 - 100].
Source	NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/prc/section2/prc252.htm, 29/3/2010
Categories	summary statistics, ordinal variables, continuous variables
Further information	http://en.wikipedia.org/wiki/Percentile

Schema

```
<!-- Element -->
<xs:element name="Percentile"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:PercentileType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="PercentileType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="level" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Percentile level="50" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Percentile>

<!-- Multiple values -->
<un:Percentile level="50" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Percentile>
```

JSON

```
// Single value
{"Percentile": {"level":50, "values": [3.14]}}

// Multiple values
{"Percentile": {"level":50, "values": [3.14, 6.28, 9.42]}}
```

Java API	`// Single value declaration` `Percentile p = new Percentile(50, 3.14);` `// Multiple value declaration` `Percentile p = new Percentile(50, new double[] {3.14, 6.28, 9.42});` `// Parsing from an XML file` `XMLParser xml = new XMLParser();` `Percentile p = (Percentile)xml.parse(new File("percentile.xml"));` `// Parsing from a JSON file` `JSONParser json = new JSONParser();` `Percentile p = (Percentile)json.parse(new File("percentile.json"));` `// Encoding to an XML file` `XMLEncoder xEncoder = new XMLEncoder();` `xEncoder.encode(p, new File("percentile.xml"));` `// Encoding to a JSON file` `JSONEncoder jEncoder = new JSONEncoder();` `jEncoder.encode(p, new File("percentile.json"));`
Value constraints	**level**: *any natural number in the range [0-100] (inclusive)* **values**: *any real number*

Probability

URI	http://www.uncertml.org/statistics/probability
UncertML name	Probability
Alternative names	N/A
Definition	Given a random variable x with probability density function $f(x)$, the probability that x lies in some part of its domain $\mathcal{X}$ is defined as $P(x \in \mathcal{X}) = \int_{x \in \mathcal{X}} f(x)$. $\mathcal{X}$ 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. $P(x \geq 1.7 \cap x < 3.2)$.
Parameters	a (`gt`) - the exclusive lower limit of x, that is $P(x > a)$ b (`lt`) - the exclusive upper limit of x, that is $P(x < b)$ a (`ge`) - the inclusive lower limit of x, that is $P(x \geq a)$ b (`le`) - the inclusive upper limit of x, that is $P(x \leq b)$
Source	N/A
Categories	summary statistic, probability, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/Probability.html
Schema	(see below)

```
<!-- Element -->
<xs:element name="Probability"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ProbabilityType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ProbabilityType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="probabilities"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
      <xs:attribute name="gt" type="xs:double"/>
      <xs:attribute name="lt" type="xs:double"/>
```

```
      <xs:attribute name="ge" type="xs:double"/>
      <xs:attribute name="le" type="xs:double"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Probability lt="3.14" xmlns:un="http://www.uncertml.org/2.0">
  <un:probabilities>0.1</un:probabilities>
</un:Probability>

<!-- Multiple values -->
<un:Probability lt="3.14" xmlns:un="http://www.uncertml.org/2.0">
  <un:probabilities>0.1 0.2 0.3</un:probabilities>
</un:Probability>
```

JSON

```
// Single value
{"Probability": {"constraints": [{"type":"LESS_THAN",
"value":3.14}], "values": [0.1]}}

// Multiple values
{"Probability": {"constraints": [{"type":"LESS_THAN",
"value":3.14}], "values": [0.1, 0.2, 0.3]}}
```

Java API

```
// Single value declaration
Probability p = new Probability(new
ProbabilityConstraint(ConstraintType.LESS_THAN,  3.14),  0.1);

// Multiple value declaration
Probability p = new Probability(new
ProbabilityConstraint(ConstraintType.LESS_THAN,  3.14),  new
double[] {0.1,  0.2,  0.3});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Probability p = (Probability)xml.parse(new File("probability.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Probability p = (Probability)json.parse(new
File("probability.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(p,  new File("probability.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(p,  new File("probability.json"));
```

Value constraints

gt, **lt**, **ge**, **le**: *any real number*
probabilities: *any real number in the range [0 - 1] (inclusive)*

Quantile

URI	http://www.uncertml.org/statistics/quantile
UncertML name	Quantile
Alternative names	N/A
Definition	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. $P(x < q) = p$.

Parameters

p (`level`) the quantile probability (or level)

Source

N/A

Categories

summary statistic, probability, ordinal variables, continuous variables

Further information

http://mathworld.wolfram.com/Quantile.html

Schema

```xml
<!-- Element -->
<xs:element name="Quantile"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:QuantileType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex Type -->
<xs:complexType name="QuantileType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="level" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:double">
            <xs:minInclusive value="0.0"/>
            <xs:maxInclusive value="1.0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Single value -->
<un:Quantile level="0.05" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Quantile>

<!-- Multiple values -->
<un:Quantile level="0.05" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Quantile>
```

JSON

```json
// Single value
{"Quantile": {"level":0.05, "values": [3.14]}}

// Multiple values
{"Quantile": {"level":0.05, "values": [3.14, 6.28, 9.42]}}
```

Java API

```java
// Single value declaration
Quantile q = new Quantile(0.05,  3.14);

// Multiple value declaration
Quantile q = new Quantile(0.05,  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Quantile q = (Quantile)xml.parse(new File("quantile.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Quantile q = (Quantile)json.parse(new File("quantile.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(q,  new File("quantile.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(q,  new File("quantile.json"));
```

Value constraints	**level**: *any real number in the range [0 - 1] (inclusive)* **values**: *any real number* - q in the above.

Quartile

URI	http://www.uncertml.org/statistics/quartile
UncertML name	Quartile
Alternative names	N/A
Definition	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.
Parameters	The proportion of probability (`level`) associated with the quantile (i.e. 0.25, 0.5, 0.75 or 1.0)
Source	N/A
Categories	summary statistics, robust, dispersion, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/Quartile.html
Schema	(see below)
XML	(see below)
JSON	

Schema

```
<!-- Element -->
<xs:element name="Quartile"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:QuartileType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="QuartileType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
      <xs:attribute name="level" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:double">
            <xs:pattern value="0.25"/>
            <xs:pattern value="0.50"/>
            <xs:pattern value="0.75"/>
            <xs:pattern value="1.00"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Quartile level="0.25" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Quartile>

<!-- Multiple values -->
<un:Quartile level="0.25" xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Quartile>
```

JSON

	`` // Single value {"Quartile": {"level":0.25, "values": [3.14]}} // Multiple values {"Quartile": {"level":0.25, "values": [3.14, 6.28, 9.42]}} ```
Java API	``` // Single value declaration Quartile q = new Quartile(0.25, 3.14); // Multiple value declaration Quartile q = new Quartile(0.25, new double[] {3.14, 6.28, 9.42}); // Parsing from an XML file XMLParser xml = new XMLParser(); Quartile q = (Quartile)xml.parse(new File("quartile.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); Quartile q = (Quartile)json.parse(new File("quartile.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(q, new File("quartile.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(q, new File("quartile.json")); ```
Value constraints	**level**: *enumeration [lower / upper]* **values**: *any real number*

Range

URI	http://www.uncertml.org/statistics/range
UncertML name	Range
Alternative names	Statistical range
Definition	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.
Parameters	N/A
Source	N/A
Categories	summary statistic, dispersion, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/StatisticalRange.html
Schema	``` <!-- Element --> <xs:element name="Range" substitutionGroup="un:AbstractSummaryStatistic"> <xs:complexType> <xs:complexContent> <xs:extension base="un:RangeType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="RangeType"> <xs:complexContent> <xs:extension base="un:AbstractSummaryStatisticType"> <xs:sequence> <xs:element name="lower" type="un:ContinuousValuesType"/> ```

```
      <xs:element name="upper" type="un:ContinuousValuesType"/>
    </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Range xmlns:un="http://www.uncertml.org/2.0">
  <un:lower>3.14</un:lower>
  <un:upper>6.28</un:upper>
</un:Range>

<!-- Multiple values -->
<un:Range xmlns:un="http://www.uncertml.org/2.0">
  <un:lower>3.14 6.28 9.42</un:lower>
  <un:upper>6.28 12.57 18.85</un:upper>
</un:Range>
```

JSON

```
// Single value
{"Range": {"lower": [3.14], "upper": [6.28]}}

// Multiple values
{"Range": {"lower": [3.14, 6.28, 9.42], "upper": [6.28, 12.57,
18.85]}}
```

Java API

```
// Single value declaration
Range r = new Range(3.14,  6.28);

// Multiple value declaration
Range r = new Range(new double[] {3.14,  6.28,  9.42},  new double[]
{6.28,  12.57,  18.85});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Range r = (Range)xml.parse(new File("range.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Range r = (Range)json.parse(new File("range.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(r,  new File("range.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(r,  new File("range.json"));
```

Value constraints

lower: *any real number*
upper: *any real number*

Skewness

URI	http://www.uncertml.org/statistics/skewness
UncertML name	Skewness
Alternative names	N/A
Definition	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 3-rd centred moment of the distribution and σ is its standard deviation.
Parameters	N/A
Source	

	N/A
Categories	summary statistic, ordinal variables, continuous variables
Further information	http://mathworld.wolfram.com/Skewness.html
Schema	see below

```
<!-- Element -->
<xs:element name="Skewness"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:SkewnessType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="SkewnessType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:Skewness xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Skewness>

<!-- Multiple values -->
<un:Skewness xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Skewness>
```

JSON

```
// Single value
{"Skewness": {"values": [3.14]}}

// Multiple values
{"Skewness": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
Skewness s = new Skewness(3.14);

// Multiple value declaration
Skewness s = new Skewness(new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Skewness s = (Skewness)xml.parse(new File("skewness.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Skewness s = (Skewness)json.parse(new File("skewness.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(s,  new File("skewness.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(s,  new File("skewness.json"));
```

Value constraints	*any real number*

Standard deviation

URI	http://www.uncertml.org/statistics/standard-deviation

UncertML name: StandardDeviation

Alternative names: population standard deviation, std dev

Definition: 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 $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ where $\bar{x} = \frac{1}{n}\cdot\sum_{i=1}^{n} x_i$, x_i represents with i'th 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.

Parameters: N/A

Source: N/A

Categories: summary statistics, dispersion, ordinal variables, continuous variables

Further information: http://en.wikipedia.org/wiki/Standard_deviation

Schema

```
<!-- Element -->
<xs:element name="StandardDeviation"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:StandardDeviationType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="StandardDeviationType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence>
        <xs:element name="values" type="un:ContinuousValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:StandardDeviation xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:StandardDeviation>

<!-- Multiple values -->
<un:StandardDeviation xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:StandardDeviation>
```

JSON

```
// Single value
{"StandardDeviation": {"values": [3.14]}}

// Multiple values
{"StandardDeviation": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
StandardDeviation sd = new StandardDeviation(3.14);

// Multiple value declaration
StandardDeviation sd = new StandardDeviation(new double[] {3.14,
6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
```

	`` StandardDeviation sd = (StandardDeviation)xml.parse(new File("standard-deviation.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); StandardDeviation sd = (StandardDeviation)json.parse(new File("standard-deviation.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(sd, new File("standard-deviation.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(sd, new File("standard-deviation.json")); ```
Value constraints	**value** *any positive real number*

Statistics collection

URI	http://www.uncertml.org/statistics/statistics-collection
UncertML name	StatisticsCollection
Alternative names	N/A
Definition	A statistics collection is a grouping mechanism for statistics. It is provided to combine statistics which naturally belong together, for example mean and variance would often be supplied together. More complex groupings are possible, for example providing the mean, variance and a set of quantiles to provide a more complete summary.
Parameters	N/A
Source	N/A
Categories	summary statistic, grouping
Further information	N/A

Schema

```
<!-- Element -->
<xs:element name="StatisticsCollection"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:StatisticsCollectionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="StatisticsCollectionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSummaryStatisticType">
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="un:AbstractSummaryStatistic"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:StatisticsCollection xmlns:un="http://www.uncertml.org/2.0">
  <un:Mean>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Mean>
  <un:Variance>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Variance>
</un:StatisticsCollection>
```

JSON

```
// Multiple values
{"StatisticCollection": {"members": [{"Mean": {"values": [3.14,
6.28, 9.42]}}, {"Variance": {"values": [3.14, 6.28, 9.42]}}]}}
```

Java API

```
// Multiple value declaration
StatisticCollection sc = new StatisticCollection(new Mean(new
double[] {3.14,  6.28,  9.42}),  new Variance(new double[] {3.14,
6.28,  9.42}));

// Parsing from an XML file
XMLParser xml = new XMLParser();
StatisticCollection sc = (StatisticCollection)xml.parse(new
File("statistic-collection.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
StatisticCollection sc = (StatisticCollection)json.parse(new
File("statistic-collection.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(sc,  new File("statistic-collection.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(sc,  new File("statistic-collection.json"));
```

Value constraints: N/A

Variance

URI	http://www.uncertml.org/statistics/variance
UncertML name	Variance
Alternative names	population variance
Definition	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 $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ where $\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$, x_i represents with i'th 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.
Parameters	N/A
Source	N/A
Categories	summary statistics, dispersion, ordinal variables, continuous variables
Further information	http://en.wikipedia.org/wiki/Variance

Schema

```
<!-- Element -->
<xs:element name="Variance"
substitutionGroup="un:AbstractSummaryStatistic">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:VarianceType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="VarianceType">
  <xs:complexContent>
```

```xml
      <xs:extension base="un:AbstractSummaryStatisticType">
        <xs:sequence>
          <xs:element name="values" type="un:ContinuousValuesType"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

XML

```xml
<!-- Single value -->
<un:Variance xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14</un:values>
</un:Variance>

<!-- Multiple values -->
<un:Variance xmlns:un="http://www.uncertml.org/2.0">
  <un:values>3.14 6.28 9.42</un:values>
</un:Variance>
```

JSON

```
// Single value
{"Variance": {"values": [3.14]}}

// Multiple values
{"Variance": {"values": [3.14, 6.28, 9.42]}}
```

Java API

```java
// Single value declaration
Variance v = new Variance(3.14);

// Multiple value declaration
Variance v = new Variance(new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
Variance v = (Variance)xml.parse(new File("variance.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
Variance v = (Variance)json.parse(new File("variance.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(v,  new File("variance.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(v,  new File("variance.json"));
```

Value constraints: **value** *any positive real number*

Distributions

Bernoulli distribution

URI	http://www.uncertml.org/distributions/bernoulli
UncertML name	BernoulliDistribution
Alternative names	N/A
Definition	A random variable x follows a Bernoulli distribution if the probability mass function (pmf) is of the form shown below. It describes the distribution of a single binary variable x.
Parameters	μ (`probabilities`) a real $\in [0, 1]$, the probability of $x = 1$, real.
Support	$x \in \{0, 1\}$

PDF

$f(x;\mu) = \mu^x(1-\mu)^{1-x}$

Source

Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.

Categories

probability distribution, binary variables

Further information

http://mathworld.wolfram.com/BernoulliDistribution.html

Schema

```xml
<!-- Element -->
<xs:element name="BernoulliDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:BernoulliDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="BernoulliDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="probabilities"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Single value -->
<un:BernoulliDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:probabilities>0.1</un:probabilities>
</un:BernoulliDistribution>

<!-- Multiple values -->
<un:BernoulliDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:probabilities>0.1 0.2 0.3</un:probabilities>
</un:BernoulliDistribution>
```

JSON

```json
// Single value
{"BernoulliDistribution": {"probabilities": [0.1]}}

// Multiple values
{"BernoulliDistribution": {"probabilities": [0.1, 0.2, 0.3]}}
```

Java API

```java
// Single value declaration
BernoulliDistribution bd = new BernoulliDistribution(0.1);

// Multiple value declaration
BernoulliDistribution bd = new BernoulliDistribution(new double[]
{0.1,  0.2,  0.3});

// Parsing from an XML file
XMLParser xml = new XMLParser();
BernoulliDistribution bd = (BernoulliDistribution)xml.parse(new
File("bernoulli-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
BernoulliDistribution bd = (BernoulliDistribution)json.parse(new
File("bernoulli-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(bd,  new File("bernoulli-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(bd,  new File("bernoulli-distribution.json"));
```

Beta distribution

URI	http://www.uncertml.org/distributions/beta
UncertML name	BetaDistribution
Alternative names	N/A
Definition	A random variable x is Beta distributed if the probability density function (pdf) is of the form shown below. The distribution is usually denoted as $x \sim Be(\alpha, \beta)$ with parameters α, β. 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.
Parameters	α (alpha) a positive real β (beta) a positive real
Support	$x \in [0, 1]$
PDF	$f(x; \alpha, \beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ the Beta function, and Γ is the Gamma function.
Source	Johnson, Kotz and Balakrishnan, "Continuous Univariate Distributions", Wiley, 1995.
Categories	continuous variables, distribution, probability
Further information	http://mathworld.wolfram.com/BetaDistribution.html ; http://en.wikipedia.org/wiki/Beta_distribution

Schema

```
<!-- Element -->
<xs:element name="BetaDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:BetaDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="BetaDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="alpha" type="un:PositiveRealValuesType"/>
        <xs:element name="beta" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:BetaDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:alpha>3.14</un:alpha>
  <un:beta>3.14</un:beta>
</un:BetaDistribution>

<!-- Multiple values -->
<un:BetaDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:alpha>3.14 6.28 9.42</un:alpha>
  <un:beta>3.14 6.28 9.42</un:beta>
</un:BetaDistribution>
```

JSON

```
// Single value
{"BetaDistribution": {"alpha": [3.14], "beta": [3.14]}}
```

	`// Multiple values` `{"BetaDistribution": {"alpha": [3.14, 6.28, 9.42], "beta": [3.14, 6.28, 9.42]}}`
Java API	see code below

```
// Single value declaration
BetaDistribution bd = new BetaDistribution(3.14,  3.14);

// Multiple value declaration
BetaDistribution bd = new BetaDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
BetaDistribution bd = (BetaDistribution)xml.parse(new File("beta-
distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
BetaDistribution bd = (BetaDistribution)json.parse(new File("beta-
distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(bd,  new File("beta-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(bd,  new File("beta-distribution.json"));
```

Binomial distribution

URI	http://www.uncertml.org/distributions/binomial
UncertML name	BinomialDistribution
Alternative names	N/A
Definition	A random variable x follows a Binomial distribution if the probability mass function (pmf) is of the form shown below. 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.
Parameters	n (`numberOfTrials`), `positive integer` θ (`=probabilityOfSuccess`) $\in [0,1]$.
Support	x non-negative integer
PDF	$f(x; n, \theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x}$, where $\binom{n}{x}$ denotes n choose x.
Source	Miller and Miller, "Mathematical Statistics with Applications", 2004, 7th Edition, Pearson Prentice Hall
Categories	ordinal variables, distribution
Further information	http://mathworld.wolfram.com/BinomialDistribution.html
Schema	see code below

```
<!-- Element -->
<xs:element name="BinomialDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:BinomialDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
```

	``<xs:complexType name="BinomialDistributionType"> <xs:complexContent> <xs:extension base="un:AbstractDistributionType"> <xs:sequence> <xs:element name="numberOfTrials" type="un:NaturalNumbersType"/> <xs:element name="probabilityOfSuccess" type="un:ProbabilityValuesType"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>```
XML	```<!-- Single value --> <BinomialDistribution> <numberOfTrials>10</numberOfTrials> <probabilityOfSuccess>0.5</probabilityOfSuccess> </BinomialDistribution> <!-- Multiple value --> <BinomialDistribution> <numberOfTrials>10 10 10</numberOfTrials> <probabilityOfSuccess>0.5 0.5 0.5</probabilityOfSuccess> </BinomialDistribution>```
JSON	```// Single value {"BinomialDistribution": {"numberOfTrials": [10], "probabilityOfSuccess": [0.5]}} // Multiple values {"BinomialDistribution": {"numberOfTrials": [10, 10, 10], "probabilityOfSuccess": [0.5, 0.5, 0.5]}}```
Java API	```// Single value declaration BinomialDistribution b = new BinomialDistribution(10, 0.5); // Multiple value declaration BinomialDistribution b = new BinomialDistribution(new int[] {10, 10, 10}, new double[] {0.5, 0.5, 0.5}); // Parsing from an XML file XMLParser p = new XMLParser(); BinomialDistribution b = (BinomialDistribution)p.parse(new File("binomial-distribution.xml")); // Parsing from a JSON file JSONParser p = new JSONParser(); BinomialDistribution b = (BinomialDistribution)p.parse(new File("binomial-distribution.json")); // Encoding to an XML file XMLEncoder e = new XMLEncoder(); e.encode(b, new File("binomial-distribution.xml")); // Encoding to a JSON file JSONEncoder e = new JSONEncoder(); e.encode(b, new File("binomial-distribution.json"));```

Cauchy distribution

URI	http://www.uncertml.org/distributions/cauchy
UncertML name	CauchyDistribution
Alternative names	Cauchy-Lorenz distribution, Lorenz distribution, Breit-Wigner distribution
Definition	A random variable x follows a Cauchy distribution if the probability density function (pdf) is of the form shown below. The Cauchy distribution is equivalent to a Student-t distribution with 1 degree of freedom. It is widely used in physics, optics and astronomy.

Parameters

θ (location) a real,
γ (scale) a positive real.

Support

$x \in \mathcal{R}$

PDF

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

Source

http://en.wikipedia.org/wiki/Cauchy_distribution

Categories

continuous variables, distribution

Further information

http://mathworld.wolfram.com/CauchyDistribution.html,
http://en.wikipedia.org/wiki/Cauchy_distribution

Schema

```
<!-- Element -->
<xs:element name="CauchyDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:CauchyDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="CauchyDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="location" type="un:ContinuousValuesType"/>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:CauchyDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14</un:location>
  <un:scale>3.14</un:scale>
</un:CauchyDistribution>

<!-- Multiple values -->
<un:CauchyDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14 6.28 9.42</un:location>
  <un:scale>3.14 6.28 9.42</un:scale>
</un:CauchyDistribution>
```

JSON

```
// Single value
{"CauchyDistribution": {"location": [3.14], "scale": [3.14]}}

// Multiple values
{"CauchyDistribution": {"location": [3.14, 6.28, 9.42], "scale":
[3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
CauchyDistribution cd = new CauchyDistribution(3.14,  3.14);

// Multiple value declaration
CauchyDistribution cd = new CauchyDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
CauchyDistribution cd = (CauchyDistribution)xml.parse(new
File("cauchy-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
CauchyDistribution cd = (CauchyDistribution)json.parse(new
File("cauchy-distribution.json"));
```

```
// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(cd,  new File("cauchy-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(cd,  new File("cauchy-distribution.json"));
```

Chi-square distribution

URI	http://www.uncertml.org/distributions/chi-square
UncertML name	ChiSquareDistribution
Alternative names	N/A
Definition	A random variable x is Chi-square distributed if the probability density function (pdf) is of the form shown below. The distribution is usually denoted as $x \sim \mathcal{X}_\nu$ where ν is known as the degrees of freedom (d.f) parameter. The d.f. 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 $X \sim Gamma(k = \nu/2, \theta = 2)$.
Parameters	ν (`degreesOfFreedom`) a positive integer
Support	x a positive real
PDF	$f(x;\nu) = \frac{1}{\Gamma(\nu/2)2^{\nu/2}} x^{\nu/2-1} exp(-x/2)$
Source	http://en.wikipedia.org/wiki/Chi-square_distribution
Categories	continuous variables, distribution
Further information	http://mathworld.wolfram.com/Chi-SquaredDistribution.html

Schema

```
<!-- Element -->
<xs:element name="ChiSquareDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ChiSquareDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ChiSquareDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="degreesOfFreedom"
type="un:PositiveNaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:ChiSquareDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:degreesOfFreedom>1</un:degreesOfFreedom>
</un:ChiSquareDistribution>

<!-- Multiple values -->
<un:ChiSquareDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:degreesOfFreedom>1 2 3</un:degreesOfFreedom>
</un:ChiSquareDistribution>
```

JSON

```
// Single value
{"ChiSquareDistribution": {"degreesOfFreedom": [1]}}

// Multiple values
{"ChiSquareDistribution": {"degreesOfFreedom": [1, 2, 3]}}
```

Java API

```
// Single value declaration
ChiSquareDistribution csd = new ChiSquareDistribution(1);

// Multiple value declaration
ChiSquareDistribution csd = new ChiSquareDistribution(new int[] {1,
2,  3});

// Parsing from an XML file
XMLParser xml = new XMLParser();
ChiSquareDistribution csd = (ChiSquareDistribution)xml.parse(new
File("chi-square-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
ChiSquareDistribution csd = (ChiSquareDistribution)json.parse(new
File("chi-square-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(csd,  new File("chi-square-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(csd,  new File("chi-square-distribution.json"));
```

Dirichlet distribution

URI	http://www.uncertml.org/distributions/dirichlet
UncertML name	DirichletDistribution
Alternative names	N/A
Definition	A K dimensional random variable x follows a Dirichlet distribution if the probability density function (pdf) is of the form shown below. It is the multivariate extension of the Beta distribution to higher dimensions with K a positive integer greater than or equal to 2.
Parameters	$\alpha = \{\alpha_1, \ldots, \alpha_K\}$ (`concentration`) vector with a separate component (each a positive real) for each dimension of the input domain.
Support	K dimensional input space $\mathbf{x} = \{x_1, \ldots, x_K\}$ such that $\sum_{i=1}^{K} x_i = 1$ and $x_i > 0 \forall i$.
PDF	$f(\mathbf{x}; \alpha) = \frac{1}{B(\mathbf{a})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$ where $B(\mathbf{a}) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}$ and $\Gamma(.)$ the Gamma function.
Source	http://en.wikipedia.org/wiki/Dirichlet_distribution
Categories	continuous variables, distribution, multivariate
Further information	http://en.wikipedia.org/wiki/Dirichlet_distribution

Schema

```
<!-- Element -->
<xs:element name="DirichletDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
```

```xml
      <xs:complexContent>
        <xs:extension base="un:DirichletDistributionType"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <!-- Complex type -->
  <xs:complexType name="DirichletDistributionType">
    <xs:complexContent>
      <xs:extension base="un:AbstractDistributionType">
        <xs:sequence>
          <xs:element name="concentration"
type="un:PositiveRealValuesType"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

XML

```xml
<!-- Multiple values -->
<un:DirichletDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:concentration>3.14 6.28 9.42 12.56</un:concentration>
</un:DirichletDistribution>
```

JSON

```json
// Multiple values
{"DirichletDistribution": {"concentration": [3.14, 6.28, 9.42,
12.56]}}
```

Java API

```java
// Multiple value declaration
DirichletDistribution dd = new DirichletDistribution(new double[]
{3.14,  6.28,  9.42, 12.56});

// Parsing from an XML file
XMLParser xml = new XMLParser();
DirichletDistribution dd = (DirichletDistribution)xml.parse(new
File("dirichlet-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
DirichletDistribution dd = (DirichletDistribution)json.parse(new
File("dirichlet-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(dd,  new File("dirichlet-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(dd,  new File("dirichlet-distribution.json"));
```

Exponential distribution

URI	http://www.uncertml.org/distributions/exponential
UncertML name	ExponentialDistribution
Alternative names	N/A
Definition	A random variable x follows an exponential distribution if the probability density function (pdf) is of the form shown below. It is often represented as $x \sim \mathrm{Exp}(\lambda)$. It is used to model the time between events for a Poisson process and is used in simulation of stochastic systems.
Parameters	λ (`rate`) a positive real. Note that sometimes a scale parameter $\beta = \frac{1}{\lambda}$ is used as the parameter - in UncertML we use the rate.
Support	$x \geq 0$, a non-negative real.

PDF	$f(x;\lambda) = \lambda e^{-\lambda x}.$
Source	D.J.Wilkinson, "Stochastic Modelling for Systems Biology", 2006, CRC press.
Categories	continuous variables, distribution
Further information	http://mathworld.wolfram.com/ExponentialDistribution.html; http://en.wikipedia.org/wiki/Exponential_distribution

Schema

```
<!-- Element -->
<xs:element name="ExponentialDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ExponentialDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ExponentialDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="rate" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:ExponentialDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:rate>3.14</un:rate>
</un:ExponentialDistribution>

<!-- Multiple values -->
<un:ExponentialDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:rate>3.14 6.28 9.42</un:rate>
</un:ExponentialDistribution>
```

JSON

```
// Single value
{"ExponentialDistribution": {"rate": [3.14]}}

// Multiple values
{"ExponentialDistribution": {"rate": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
ExponentialDistribution ed = new ExponentialDistribution(3.14);

// Multiple value declaration
ExponentialDistribution ed = new ExponentialDistribution(new double[]
{3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
ExponentialDistribution ed = (ExponentialDistribution)xml.parse(new
File("exponential-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
ExponentialDistribution ed = (ExponentialDistribution)json.parse(new
File("exponential-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(ed,  new File("exponential-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(ed,  new File("exponential-distribution.json"));
```

F distribution

URI

http://www.uncertml.org/distributions/f

UncertML name

FDistribution

Alternative names

Snedecor's F distribution, Fisher-Snedecor distribution

Definition

A random variable x follows an F distribution if the probability density function (pdf) is of the form shown below. It often arises as the ratio of two random variables that are identically Chi-Square distributed.

Parameters

ν_1 (numerator degrees of freedom) a positive integer,
ν_2 (denominator degrees of freedom) a positive integer.

Support

x a non-negative real.

PDF

$f(x; \nu_1, \nu_2) = \frac{1}{B(\nu_1/2, \nu_2/2)} \left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} x^{\nu_1/2-1} \left(1 + \frac{\nu_1}{\nu_2} x\right)^{-\frac{\nu_1+\nu_2}{2}}$ where $B(.)$ the Beta function (see description in Beta distribution).

Source

Johnson, Kotz and Balakrishnan, "Continuous univariate Distributions", Volume 2, 1995, John Wiley and Sons.

Categories

continuous variables, distribution

Further information

http://mathworld.wolfram.com/F-Distribution.html;
http://en.wikipedia.org/wiki/F-distribution

Schema

```
<!-- Element -->
<xs:element name="FDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:FDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="FDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="denominator"
type="un:NaturalNumbersType"/>
        <xs:element name="numerator" type="un:NaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:FDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:denominator>1</un:denominator>
  <un:numerator>1</un:numerator>
</un:FDistribution>

<!-- Multiple values -->
<un:FDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:denominator>1 2 3</un:denominator>
  <un:numerator>1 2 3</un:numerator>
</un:FDistribution>
```

JSON

```
// Single value
{"FDistribution": {"denominator": [1], "numerator": [1]}}

// Multiple values
{"FDistribution": {"denominator": [1, 2, 3], "numerator": [1, 2,
```

	`` 3]}} ```
Java API	(see code below)

```
// Single value declaration
FDistribution fd = new FDistribution(1,  1);

// Multiple value declaration
FDistribution fd = new FDistribution(new int[] {1,  2,  3},  new
int[] {1,  2,  3});

// Parsing from an XML file
XMLParser xml = new XMLParser();
FDistribution fd = (FDistribution)xml.parse(new File("f-
distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
FDistribution fd = (FDistribution)json.parse(new File("f-
distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(fd,  new File("f-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(fd,  new File("f-distribution.json"));
```

Gamma distribution

URI	http://www.uncertml.org/distributions/gamma
UncertML name	GammaDistribution
Alternative names	N/A
Definition	A random variable x is Gamma distributed if the probability density function (pdf) is of the form shown below. The distribution is usually denoted as $x \sim Gamma(k,\theta)$ where 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. The Chi-square distribution is a special case of the Gamma distribution where $X \sim Gamma(k=\nu/2, \theta=2)$ where ν the degrees of freedom.
Parameters	k (`shape`) a positive real, θ (`scale`) a positive real
Support	x a non-negative real
PDF	The standard form used is $f(x;k,\theta)=\frac{1}{\Gamma(k)\theta^k}x^{k-1}exp(-x/\theta)$ with $\Gamma(\cdot)$ the Gamma function.
Source	http://en.wikipedia.org/wiki/Gamma_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Gamma_distribution
Schema	(see code below)

```
<!-- Element -->
<xs:element name="GammaDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:GammaDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
```

```
</xs:element>

<!-- Complex type -->
<xs:complexType name="GammaDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="shape" type="un:PositiveRealValuesType"/>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:GammaDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:shape>3.14</un:shape>
  <un:scale>3.14</un:scale>
</un:GammaDistribution>

<!-- Multiple values -->
<un:GammaDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:shape>3.14 6.28 9.42</un:shape>
  <un:scale>3.14 6.28 9.42</un:scale>
</un:GammaDistribution>
```

JSON

```
// Single value
{"GammaDistribution": {"shape": [3.14], "scale": [3.14]}}

// Multiple values
{"GammaDistribution": {"shape": [3.14, 6.28, 9.42], "scale": [3.14,
6.28, 9.42]}}
```

Java API

```
// Single value declaration
GammaDistribution gd = new GammaDistribution(3.14,  3.14);

// Multiple value declaration
GammaDistribution gd = new GammaDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
GammaDistribution gd = (GammaDistribution)xml.parse(new File("gamma-
distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
GammaDistribution gd = (GammaDistribution)json.parse(new
File("gamma-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(gd,  new File("gamma-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(gd,  new File("gamma-distribution.json"));
```

Geometric distribution

URI	http://www.uncertml.org/distributions/geometric
UncertML name	GeometricDistribution
Alternative names	N/A
Definition	A random variable x follows a geometric distribution if the probability mass function (pmf) is of the form shown below. It is often represented as $x \sim \mathrm{Geom}(p)$. It is the discrete analogue of the exponential distribution.
Parameters	p (`probability`) a real $\in [0,1]$.

<table>
<tr><td>Support</td><td>x positive integer, number of independent trials until the first success is achieved.</td></tr>
<tr><td>PDF</td><td>$f(x;p) = (1-p)^{x-1}p.$</td></tr>
<tr><td>Source</td><td>D.J.Wilkinson, "Stochastic Modelling for Systems Biology", 2006, CRC press.</td></tr>
<tr><td>Categories</td><td>ordinal variables, distribution</td></tr>
<tr><td>Further information</td><td>http://mathworld.wolfram.com/GeometricDistribution.html</td></tr>
<tr><td>Schema</td><td>

```
<!-- Element -->
<xs:element name="GeometricDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:GeometricDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="GeometricDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="probability"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

</td></tr>
<tr><td>XML</td><td>

```
<!-- Single value -->
<un:GeometricDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:probability>0.1</un:probability>
</un:GeometricDistribution>

<!-- Multiple values -->
<un:GeometricDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:probability>0.1 0.2 0.3</un:probability>
</un:GeometricDistribution>
```

</td></tr>
<tr><td>JSON</td><td>

```
// Single value
{"GeometricDistribution": {"probability": [0.1]}}

// Multiple values
{"GeometricDistribution": {"probability": [0.1, 0.2, 0.3]}}
```

</td></tr>
<tr><td>Java API</td><td>

```
// Single value declaration
GeometricDistribution gd = new GeometricDistribution(0.1);

// Multiple value declaration
GeometricDistribution gd = new GeometricDistribution(new double[]
{0.1,  0.2,  0.3});

// Parsing from an XML file
XMLParser xml = new XMLParser();
GeometricDistribution gd = (GeometricDistribution)xml.parse(new
File("geometric-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
GeometricDistribution gd = (GeometricDistribution)json.parse(new
File("geometric-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(gd,  new File("geometric-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(gd,  new File("geometric-distribution.json"));
```

</td></tr>
</table>

Hypergeometric distribution

URI	http://www.uncertml.org/distributions/hypergeometric
UncertML name	HypergeometricDistribution
Alternative names	N/A
Definition	A random variable x follows a hypergeometric distribution if the probability mass function (pmf) is of the form shown below. It describes the number of successes in a sequence of draws without replacement.
Parameters	N (`populationSize`) a positive integer, n (`numberOfTrials`) a positive in integer such that $1 \leq n \leq N$, m (`numberOfSuccesses`) a positive integer.
Support	x a positive integer, the number of successes $\in \{\max(0, n+m-N), \ldots, \min(m, n)\}$.
PDF	$f(x; N, n, m) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$, probability of getting x successes.
Source	http://en.wikipedia.org/wiki/Hypergeometric_distribution
Categories	ordinal variables, distribution
Further information	http://mathworld.wolfram.com/HypergeometricDistribution.html

Schema

```
<!-- Element -->
<xs:element name="HypergeometricDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:HypergeometricDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="HypergeometricDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="numberOfSuccesses"
type="un:NaturalNumbersType"/>
        <xs:element name="numberOfTrials"
type="un:NaturalNumbersType"/>
        <xs:element name="populationSize"
type="un:NaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:HypergeometricDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:numberOfSuccesses>1</un:numberOfSuccesses>
  <un:numberOfTrials>2</un:numberOfTrials>
  <un:populationSize>3</un:populationSize>
</un:HypergeometricDistribution>

<!-- Multiple values -->
<un:HypergeometricDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:numberOfSuccesses>1 2 3</un:numberOfSuccesses>
  <un:numberOfTrials>2 4 6</un:numberOfTrials>
  <un:populationSize>3 6 9</un:populationSize>
</un:HypergeometricDistribution>
```

JSON

```
// Single value
{"HypergeometricDistribution": {"numberOfTrials": [2],
"numberOfSuccesses": [1], "populationSize": [3]}}

// Multiple values
{"HypergeometricDistribution": {"numberOfTrials": [2, 4, 6],
"numberOfSuccesses": [1, 2, 3], "populationSize": [3, 6, 9]}}
```

Java API

```
// Single value declaration
HypergeometricDistribution hd = new HypergeometricDistribution(2,  1,
3);

// Multiple value declaration
HypergeometricDistribution hd = new HypergeometricDistribution(new
int[] {2,  4,  6},  new int[] {1,  2,  3},  new int[] {3,  6,  9});

// Parsing from an XML file
XMLParser xml = new XMLParser();
HypergeometricDistribution hd =
(HypergeometricDistribution)xml.parse(new File("hypergeometric-
distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
HypergeometricDistribution hd =
(HypergeometricDistribution)json.parse(new File("hypergeometric-
distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(hd,  new File("hypergeometric-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(hd,  new File("hypergeometric-distribution.json"));
```

Inverse-gamma distribution

URI	http://www.uncertml.org/distributions/inverse-gamma
UncertML name	InverseGammaDistribution
Alternative names	N/A
Definition	A random variable x is Inverse Gamma distributed if the probability density function (pdf) is of the form shown below. 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.
Parameters	α (`shape`) a positive real, β (`scale`) a positive real
Support	x a positive real.
PDF	The standard form used is $f(x;k,\theta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{-\alpha-1} exp(-\beta/x)$.
Source	http://en.wikipedia.org/wiki/Inverse-gamma_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Inverse-gamma_distribution

Schema

```
<!-- Element -->
```

	`` <xs:element name="InverseGammaDistribution" substitutionGroup="un:AbstractDistribution"> <xs:complexType> <xs:complexContent> <xs:extension base="un:InverseGammaDistributionType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="InverseGammaDistributionType"> <xs:complexContent> <xs:extension base="un:AbstractDistributionType"> <xs:sequence> <xs:element name="shape" type="un:PositiveRealValuesType"/> <xs:element name="scale" type="un:PositiveRealValuesType"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType> ```
XML	``` <!-- Single value --> <un:InverseGammaDistribution xmlns:un="http://www.uncertml.org/2.0"> <un:shape>3.14</un:shape> <un:scale>3.14</un:scale> </un:InverseGammaDistribution> <!-- Multiple values --> <un:InverseGammaDistribution xmlns:un="http://www.uncertml.org/2.0"> <un:shape>3.14 6.28 9.42</un:shape> <un:scale>3.14 6.28 9.42</un:scale> </un:InverseGammaDistribution> ```
JSON	``` // Single value {"InverseGammaDistribution": {"shape": [3.14], "scale": [3.14]}} // Multiple values {"InverseGammaDistribution": {"shape": [3.14, 6.28, 9.42], "scale": [3.14, 6.28, 9.42]}} ```
Java API	``` // Single value declaration InverseGammaDistribution igd = new InverseGammaDistribution(3.14, 3.14); // Multiple value declaration InverseGammaDistribution igd = new InverseGammaDistribution(new double[] {3.14, 6.28, 9.42}, new double[] {3.14, 6.28, 9.42}); // Parsing from an XML file XMLParser xml = new XMLParser(); InverseGammaDistribution igd = (InverseGammaDistribution)xml.parse(new File("inverse-gamma-distribution.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); InverseGammaDistribution igd = (InverseGammaDistribution)json.parse(new File("inverse-gamma-distribution.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(igd, new File("inverse-gamma-distribution.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(igd, new File("inverse-gamma-distribution.json")); ```

Laplace distribution

URI	http://www.uncertml.org/distributions/laplace
UncertML name	LaplaceDistribution
Alternative names	Double exponential distribution

Definition

A random variable x is Laplace distributed if the probability density function (pdf) is of the form shown below. It can be thought of as a combination of two Exponential distributions.

Parameters

μ (`location`) a real,
b (`scale`) a positive real.

Support

x a real.

PDF

$f(x;\mu,b) = \frac{1}{2b}\exp\left(-\frac{\mathrm{abs}(x-\mu)}{b}\right)$ where abs denotes the absolute value.

Source

http://en.wikipedia.org/wiki/Laplace_distribution

Categories

continuous variables, distribution

Further information

http://en.wikipedia.org/wiki/Laplace_distribution

Schema

```
<!-- Element -->
<xs:element name="LaplaceDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:LaplaceDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="LaplaceDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="location" type="un:ContinuousValuesType"/>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:LaplaceDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14</un:location>
  <un:scale>3.14</un:scale>
</un:LaplaceDistribution>

<!-- Multiple values -->
<un:LaplaceDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14 6.28 9.42</un:location>
  <un:scale>3.14 6.28 9.42</un:scale>
</un:LaplaceDistribution>
```

JSON

```
// Single value
{"LaplaceDistribution": {"location": [3.14], "scale": [3.14]}}

// Multiple values
{"LaplaceDistribution": {"location": [3.14, 6.28, 9.42], "scale":
[3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
LaplaceDistribution ld = new LaplaceDistribution(3.14,  3.14);

// Multiple value declaration
LaplaceDistribution ld = new LaplaceDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
LaplaceDistribution ld = (LaplaceDistribution)xml.parse(new
File("laplace-distribution.xml"));
```

```
// Parsing from a JSON file
JSONParser json = new JSONParser();
LaplaceDistribution ld = (LaplaceDistribution)json.parse(new
File("laplace-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(ld,  new File("laplace-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(ld,  new File("laplace-distribution.json"));
```

Log-normal distribution

URI	http://www.uncertml.org/distributions/log-normal
UncertML name	LogNormalDistribution
Alternative names	N/A
Definition	A random variable x is Log Normal distributed if the probability density function (pdf) is of the form shown below. If variable x is Normal 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.
Parameters	μ (`logScale`) a real, often called the mean, σ^2 (`shape`) a positive real, often called the variance.
Support	x a positive real
PDF	$f(x;\mu,\sigma^2) = \frac{1}{x\sqrt{2\pi\sigma^2}} exp(-\frac{(\ln(x)-\mu)^2}{2\sigma^2})$.
Source	http://en.wikipedia.org/wiki/Log-normal_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Log-normal_distribution
Schema	see below

```
<!-- Element -->
<xs:element name="LogNormalDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:LogNormalDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="LogNormalDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="logScale" type="un:ContinuousValuesType"/>
        <xs:element name="shape" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:LogNormalDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:logScale>3.14</un:logScale>
  <un:shape>3.14</un:shape>
</un:LogNormalDistribution>

<!-- Multiple values -->
<un:LogNormalDistribution xmlns:un="http://www.uncertml.org/2.0">
```

	`` <un:logScale>3.14 6.28 9.42</un:logScale> <un:shape>3.14 6.28 9.42</un:shape> </un:LogNormalDistribution> ```
JSON	``` // Single value {"LogNormalDistribution": {"logScale": [3.14], "shape": [3.14]}} // Multiple values {"LogNormalDistribution": {"location": [3.14, 6.28, 9.42], "scale": [3.14, 6.28, 9.42]}} ```
Java API	``` // Single value declaration LogNormalDistribution lnd = new LogNormalDistribution(3.14, 3.14); // Multiple values {"LogNormalDistribution": {"logScale": [3.14, 6.28, 9.42], "shape": [3.14, 6.28, 9.42]}} // Parsing from an XML file XMLParser xml = new XMLParser(); LogNormalDistribution lnd = (LogNormalDistribution)xml.parse(new File("log-normal-distribution.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); LogNormalDistribution lnd = (LogNormalDistribution)json.parse(new File("log-normal-distribution.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(lnd, new File("log-normal-distribution.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(lnd, new File("log-normal-distribution.json")); ```

Logistic distribution

URI	http://www.uncertml.org/distributions/logistic
UncertML name	LogisticDistribution
Alternative names	N/A
Definition	A random variable x is Logistic distributed if the probability density function (pdf) is of the form shown below.
Parameters	μ (`location`) a real, s (`scale`) a positive real.
Support	x a real
PDF	$f(x;\mu,s) = \frac{\exp(-(x-\mu)/s)}{s(1+\exp(-(x-\mu)/s))^2}$
Source	http://en.wikipedia.org/wiki/Logistic_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Logistic_distribution
Schema	``` <!-- Element --> <xs:element name="LogisticDistribution" substitutionGroup="un:AbstractDistribution"> ```

	``<xs:complexType> <xs:complexContent> <xs:extension base="un:LogisticDistributionType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="LogisticDistributionType"> <xs:complexContent> <xs:extension base="un:AbstractDistributionType"> <xs:sequence> <xs:element name="location" type="un:ContinuousValuesType"/> <xs:element name="scale" type="un:PositiveRealValuesType"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>```
XML	```<!-- Single value --> <un:LogisticDistribution xmlns:un="http://www.uncertml.org/2.0"> <un:location>3.14</un:location> <un:scale>3.14</un:scale> </un:LogisticDistribution> <!-- Multiple values --> <un:LogisticDistribution xmlns:un="http://www.uncertml.org/2.0"> <un:location>3.14 6.28 9.42</un:location> <un:scale>3.14 6.28 9.42</un:scale> </un:LogisticDistribution>```
JSON	```// Single value {"LogisticDistribution": {"location": [3.14], "scale": [3.14]}} // Multiple values {"LogisticDistribution": {"location": [3.14, 6.28, 9.42], "scale": [3.14, 6.28, 9.42]}}```
Java API	```// Single value declaration LogisticDistribution ld = new LogisticDistribution(3.14, 3.14); // Multiple value declaration LogisticDistribution ld = new LogisticDistribution(new double[] {3.14, 6.28, 9.42}, new double[] {3.14, 6.28, 9.42}); // Parsing from an XML file XMLParser xml = new XMLParser(); LogisticDistribution ld = (LogisticDistribution)xml.parse(new File("logistic-distribution.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); LogisticDistribution ld = (LogisticDistribution)json.parse(new File("logistic-distribution.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(ld, new File("logistic-distribution.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(ld, new File("logistic-distribution.json"));```

Mixture model

URI	http://www.uncertml.org/distributions/mixture-model
UncertML name	MixtureModel
Alternative names	N/A
Definition	A Mixture model is a linear combination of more basic distributions. A widely used case is where the basic distributions are Gaussian in which

	case the model is known as the Gaussian Mixture Model. The EM algorithm is widely used to determine the values of the mixing coefficients.
Parameters	$\pi = \{\pi_1, \ldots, \pi_K\}$ (`weight`) or mixing coefficients such that $\sum_{i=1}^{K} \pi_i = 1$ and $0 \leq \pi_i \leq 1 \forall i$.
Support	The support depends on the type of basic distributions used.
PDF	$f(x; \pi, \theta) = \sum_{i=1}^{K} \pi_i p_i(x; \theta_i)$ where $p_i(x; \theta_i)$ the pdf for the basic distribution for the i^{th} component with parameters θ_i.
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.
Categories	distribution
Further information	http://en.wikipedia.org/wiki/Mixture_model

Schema

```
<!-- Element -->
<xs:element name="MixtureModel"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MixtureModelType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MixtureModelType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="component" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="un:AbstractDistribution"/>
            </xs:sequence>
            <xs:attribute name="weight" type="un:probability"
use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:MixtureModel xmlns:un="http://www.uncertml.org/2.0">
  <un:component weight="0.5">
    <un:NormalDistribution>
      <un:mean>3.14 6.28 9.42</un:mean>
      <un:standardDeviation>3.14 6.28 9.42</un:standardDeviation>
    </un:NormalDistribution>
  </un:component>
  <un:component weight="0.5">
    <un:NormalDistribution>
      <un:mean>3.14 6.28 9.42</un:mean>
      <un:standardDeviation>3.14 6.28 9.42</un:standardDeviation>
    </un:NormalDistribution>
  </un:component>
</un:MixtureModel>
```

JSON

```
// Multiple values
{"MixtureModel": {"components": [{"weight":0.5, "distribution":
{"NormalDistribution": {"mean": [3.14, 6.28, 9.42],
"standardDeviation": [3.14, 6.28, 9.42]}}}, {"weight":0.5,
"distribution": {"NormalDistribution": {"mean": [3.14, 6.28, 9.42],
"standardDeviation": [3.14, 6.28, 9.42]}}}]}}
```

Java API

```
// Multiple value declaration
```

```
WeightedDistribution dist1 = new WeightedDistribution(0.5,  new
NormalDistribution(3.14,  3.14));
WeightedDistribution dist2 = new WeightedDistribution(0.5,  new
NormalDistribution(3.14,  3.14));
MixtureModel mm = new MixtureModel(new WeightedDistribution[]
{dist1,  dist2});

// Parsing from an XML file
XMLParser xml = new XMLParser();
MixtureModel mm = (MixtureModel)xml.parse(new File("mixture-
model.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
MixtureModel mm = (MixtureModel)json.parse(new File("mixture-
model.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(mm,  new File("mixture-model.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(mm,  new File("mixture-model.json"));
```

Multinomial distribution

URI	http://www.uncertml.org/distributions/multinomial
UncertML name	MultinomialDistribution
Alternative names	N/A
Definition	A random variable x follows a Multinomial distribution if the probability mass function (pmf) is of the form shown below. The Multinomial distribution is a multivariate generalisation of the Binomial distribution for a K state variable to be in state k given N observations.
Parameters	N (numberOfTrials) a positive integer, $\mathbf{p} = \{p_1, \ldots, p_K\}$ (probabilityOfSuccess) vector with elements reals $\in [0, 1]$ such that $\sum_{i=1}^{K} p_i = 1$.
Support	$x_i \in \{0, \ldots, N\}$ non-negative integers such that $\sum_{i=1}^{K} x_i = N$.
PDF	$f(\mathbf{x}; N, \mathbf{p}) = \frac{N!}{x_1! \ldots x_k!} \prod_{i=1}^{K} p_i^{x_i}$.
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer
Categories	categorical variables, ordinal variables, distribution, multivariate
Further information	http://en.wikipedia.org/wiki/Multinomial_distribution
Schema	(see below)

```
<!-- Element -->
<xs:element name="MultinomialDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MultinomialDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MultinomialDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="numberOfTrials"
type="un:positiveNaturalNumber"/>
        <xs:element name="probabilities"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML	<pre><!-- Multiple values --> <un:MultinomialDistribution xmlns:un="http://www.uncertml.org/2.0"> <un:numberOfTrials>2</un:numberOfTrials> <un:probabilities>0.1 0.2 0.3</un:probabilities> </un:MultinomialDistribution></pre>
JSON	<pre>// Multiple values {"MultinomialDistribution": {"numberOfTrials":2, "probabilities": [0.1, 0.2, 0.3]}}</pre>
Java API	<pre>// Multiple value declaration MultinomialDistribution md = new MultinomialDistribution(2, new double[] {0.1, 0.2, 0.3}); // Parsing from an XML file XMLParser xml = new XMLParser(); MultinomialDistribution md = (MultinomialDistribution)xml.parse(new File("multinomial-distribution.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); MultinomialDistribution md = (MultinomialDistribution)json.parse(new File("multinomial-distribution.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(md, new File("multinomial-distribution.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(md, new File("multinomial-distribution.json"));</pre>

Multivariate normal distribution

URI	http://www.uncertml.org/distributions/multivariate-normal
UncertML name	MultivariateNormalDistribution
Alternative names	Multivariate Gaussian Distribution
Definition	The Multivariate Normal is an extension of the univariate Normal distribution to higher dimensional vector spaces. A random vector variable of dimension k denoted $\mathbf{x}$ is normally distributed if the probability density function (pdf) is of the form shown below. The distribution is usually denoted as $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ where μ is known as the mean vector parameter and Σ the covariance matrix parameter.
Parameters	μ (`mean`) vector in the k-dimensional reals, Σ (`covarianceMatrix`) a non-negative definite $k \times k$ matrix.
Support	$\mathbf{x}$ a k-dimensional vector of reals
PDF	$f(\mathbf{x}; \mu, \Sigma) = (2\pi)^{-k/2}\det(\Sigma)^{-1/2} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu))$ where $\det(.)$ denotes the determinant and $(.)^T$ the matrix transpose.
Source	http://en.wikipedia.org/wiki/Multivariate_normal_distribution
Categories	continuous variables, distribution, multivariate

Further information	http://en.wikipedia.org/wiki/Multivariate_normal_distribution

Schema

```
<!-- Element -->
<xs:element name="MultivariateNormalDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MultivariateNormalDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MultivariateNormalDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="mean" type="un:ContinuousValuesType"/>
        <xs:element name="covarianceMatrix"
type="un:CovarianceMatrixType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:MultivariateNormalDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14 6.28</un:mean>
  <un:covarianceMatrix dimension="2">
    <un:values>3.14 0.0 0.0 3.14</un:values>
  </un:covarianceMatrix>
</un:MultivariateNormalDistribution>
```

JSON

```
// Multiple values
{"MultivariateNormalDistribution": {"mean": [3.14, 6.28],
"covarianceMatrix": {"dimension":2, "values": [3.14, 0.0, 0.0,
3.14]}}}}
```

Java API

```
// Multiple value declaration
MultivariateNormalDistribution mnd = new
MultivariateNormalDistribution(new double[] {3.14,  6.28},  new
CovarianceMatrix(2,  new double[] {3.14,  0.0,  0.0,  3.14}));

// Parsing from an XML file
XMLParser xml = new XMLParser();
MultivariateNormalDistribution mnd =
(MultivariateNormalDistribution)xml.parse(new File("multivariate-
normal-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
MultivariateNormalDistribution mnd =
(MultivariateNormalDistribution)json.parse(new File("multivariate-
normal-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(mnd,  new File("multivariate-normal-
distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(mnd,  new File("multivariate-normal-
distribution.json"));
```

Multivariate Student T distribution

URI	http://www.uncertml.org/distributions/multivariate-student-t
UncertML name	MultivariateStudentTDistribution

Alternative names	N/A
Definition	A random variable $\mathbf{x}$ follows a multivariate Student-t distribution if the probability density function (pdf) is of the form shown below. The distribution is usually denoted as $x \sim St(\mu, \Sigma, \nu)$. It is the extension of the univariate student-t distribution to higher dimensions. Student-t distributions are often used when tails are expected to be heavier than Gaussian or Normal, and can result from applying Bayesian inference.
Parameters	μ (`mean`) vector in the k-dimensional reals, Σ (`covarianceMatrix`) a non-negative definite $k \times k$ matrix, ν (`degreesOfFreedom`) a positive integer.
Support	$\mathbf{x}$ a k-dimensional vector of reals
PDF	$f(\mathbf{x}; \mu, \Sigma, \nu) = \frac{\Gamma(\nu/2+k/2)}{\Gamma(\nu/2)(\pi\nu)^{k/2}\det(\Sigma)^{1/2}}\left[1 + \frac{\Delta^2}{\nu}\right]^{-\nu/2-k/2}$ where $\Delta^2 = (\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$ is the squared Mahalanobis distance.
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.
Categories	continuous variables, distribution, multivariate
Further information	http://en.wikipedia.org/wiki/Multivariate_Student_distribution

Schema

```
<!-- Element -->
<xs:element name="MultivariateStudentTDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:MultivariateStudentTDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="MultivariateStudentTDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="mean" type="un:ContinuousValuesType"/>
        <xs:element name="covarianceMatrix"
type="un:CovarianceMatrixType"/>
        <xs:element name="degreesOfFreedom"
type="un:PositiveNaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:MultivariateStudentTDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14 6.28</un:mean>
  <un:covarianceMatrix dimension="2">
    <un:values>3.14 0.0 0.0 3.14</un:values>
  </un:covarianceMatrix>
  <un:degreesOfFreedom>1 2</un:degreesOfFreedom>
</un:MultivariateStudentTDistribution>
```

JSON

```
// Multiple values
{"MultivariateStudentTDistribution": {"mean": [3.14, 6.28],
"covarianceMatrix": {"dimension":2, "values": [3.14, 0.0, 0.0,
3.14]}, "degreesOfFreedom": [1, 2]}}
```

Java API

```
// Multiple value declaration
MultivariateStudentTDistribution mstd = new
MultivariateStudentTDistribution(new double[] {3.14,  6.28},  new
CovarianceMatrix(2,  new double[] {3.14,  0.0,  0.0,  3.14}),  new
int[] {1,  2});

// Parsing from an XML file
XMLParser xml = new XMLParser();
MultivariateStudentTDistribution mstd =
(MultivariateStudentTDistribution)xml.parse(new File("multivariate-
student-t-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
MultivariateStudentTDistribution mstd =
(MultivariateStudentTDistribution)json.parse(new File("multivariate-
student-t-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(mstd,  new File("multivariate-student-t-
distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(mstd,  new File("multivariate-student-t-
distribution.json"));
```

Negative binomial distribution

URI	http://www.uncertml.org/distributions/negative-binomial
UncertML name	NegativeBinomialDistribution
Alternative names	Pascal Distribution
Definition	A random variable x follows a Negative Binomial distribution if the probability mass function (pmf) is of the form shown below. 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.
Parameters	r (`numberOfFailures`) a positive integer, p (`probability`) of success, a real value $\in [0, 1]$.
Support	x a non-negative integer
PDF	$f(x; r, p) = \binom{x+r-1}{x} p^x (1-p)^r.$
Source	http://en.wikipedia.org/wiki/Negative_binomial_distribution
Categories	ordinal variables, distribution
Further information	http://en.wikipedia.org/wiki/Negative_binomial_distribution
Schema	(see below)

```
<!-- Element -->
<xs:element name="NegativeBinomialDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:NegativeBinomialDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="NegativeBinomialDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="numberOfFailures"
type="un:NaturalNumbersType"/>
        <xs:element name="probability"
type="un:ProbabilityValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

	`</xs:complexType>`
XML	`<!-- Single value -->` `<un:NegativeBinomialDistribution xmlns:un="http://www.uncertml.org/2.0">` `  <un:numberOfFailures>1</un:numberOfFailures>` `  <un:probability>0.1</un:probability>` `</un:NegativeBinomialDistribution>` `<!-- Multiple values -->` `<un:NegativeBinomialDistribution xmlns:un="http://www.uncertml.org/2.0">` `  <un:numberOfFailures>1 2 3</un:numberOfFailures>` `  <un:probability>0.1 0.2 0.3</un:probability>` `</un:NegativeBinomialDistribution>`
JSON	`// Single value` `{"NegativeBinomialDistribution": {"numberOfFailures": [1], "probability": [0.1]}}` `// Multiple values` `{"NegativeBinomialDistribution": {"numberOfFailures": [1, 2, 3], "probability": [0.1, 0.2, 0.3]}}`
Java API	`// Single value declaration` `NegativeBinomialDistribution nbd = new NegativeBinomialDistribution(1, 0.1);` `// Multiple value declaration` `NegativeBinomialDistribution nbd = new NegativeBinomialDistribution(new int[] {1, 2, 3}, new double[] {0.1, 0.2, 0.3});` `// Parsing from an XML file` `XMLParser xml = new XMLParser();` `NegativeBinomialDistribution nbd = (NegativeBinomialDistribution)xml.parse(new File("negative-binomial-distribution.xml"));` `// Parsing from a JSON file` `JSONParser json = new JSONParser();` `NegativeBinomialDistribution nbd = (NegativeBinomialDistribution)json.parse(new File("negative-binomial-distribution.json"));` `// Encoding to an XML file` `XMLEncoder xEncoder = new XMLEncoder();` `xEncoder.encode(nbd, new File("negative-binomial-distribution.xml"));` `// Encoding to a JSON file` `JSONEncoder jEncoder = new JSONEncoder();` `jEncoder.encode(nbd, new File("negative-binomial-distribution.json"));`

Normal distribution

URI	http://www.uncertml.org/distributions/normal
UncertML name	NormalDistribution
Alternative names	Gaussian distribution
Definition	A random variable x is normally distributed if the probability density function (pdf) is of the form shown below. 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.
Parameters	μ (mean) a real, also called the location parameter, σ^2 (variance) a positive real also called the scale parameter. Note the square root of the variance σ is known as the standard deviation, but in UncertML we use the variance as the scale parameter.
Support	x a real.
PDF	$f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$.
Source	http://en.wikipedia.org/wiki/Normal_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Normal_distribution

Schema

```
<!-- Element -->
<xs:element name="NormalDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:NormalDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="NormalDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="mean" type="un:ContinuousValuesType"/>
        <xs:element name="variance"
type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:NormalDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14</un:mean>
  <un:variance>3.14</un:variance>
</un:NormalDistribution>

<!-- Multiple values -->
<un:NormalDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14 6.28 9.42</un:mean>
  <un:variance>3.14 6.28 9.42</un:variance>
</un:NormalDistribution>
```

JSON

```
// Single value
{"NormalDistribution": {"mean": [3.14], "variance": [3.14]}}

// Multiple values
{"NormalDistribution": {"mean": [3.14, 6.28, 9.42], "variance":
[3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
NormalDistribution nd = new NormalDistribution(3.14,  3.14);

// Multiple value declaration
NormalDistribution nd = new NormalDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
NormalDistribution nd = (NormalDistribution)xml.parse(new
File("normal-distribution.xml"));
```

```
// Parsing from a JSON file
JSONParser json = new JSONParser();
NormalDistribution nd = (NormalDistribution)json.parse(new
File("normal-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(nd,  new File("normal-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(nd,  new File("normal-distribution.json"));
```

Normal inverse-gamma distribution

URI	http://www.uncertml.org/distributions/normal-inverse-gamma
UncertML name	NormalInverseGammaDistribution
Alternative names	Normal scaled Inverse Gamma distribution or Gaussian Inverse Gamma distribution
Definition	A Normal Inverse Gamma distribution is the conjugate prior of a Normal distribution with unknown mean and variance. It is the coupled product of an Inverse Gamma distribution and a Normal distribution. In particular if $p(\mathbf{X}; \mu, \sigma^2)$ is the likelihood function of a Normally distributed set of random variables with mean μ and variance σ^2 and if both the mean and variance are considered unknown, the conjugate prior is $p(\mu, \sigma^2) = p(\mu; \sigma^2)p(\sigma^2)$ where $p(\mu; \sigma^2) = \mathcal{N}(\mu; \mu_0, \sigma^2/\nu)$ a Normal prior on the mean and $p(\sigma^2) = \mathrm{IG}(\sigma^2; \alpha, \beta)$ an Inverse Gamma prior on the variance. Note that the priors are not independent as the prior variance of the mean is a linear function of of the variance σ^2. It is also common to use a Normal-Gamma distribution where a conjugate prior is placed on the unknown mean and precision (i.e. inverse variance) of the Normal likelihood in which case the prior is a product of a Normal and Gamma distributions. In the case of a Multivariate Normal likelihood, the corresponding conjugate prior is a Normal-Wishart distribution.
Parameters	μ_0 (`mean`) a real, ν (`varianceScaling`) a positive real, for the prior on mean. α (`shape`) a positive real, β (`scale`) a positive real, for the prior on variance.
Support	μ a real, σ^2 a positive real.
PDF	$f(\mu, \sigma^2; \mu_0, \nu, \alpha, \beta) = \mathcal{N}(\mu; \mu_0, \sigma^2/\nu)\mathrm{IG}(\sigma^2; \alpha, \beta)$
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer
Categories	continuous variables, distribution, bivariate
Further information	http://en.wikipedia.org/wiki/Normal-scaled_inverse_gamma_distribution
Schema	(see below)

```
<!-- Element -->
<xs:element name="NormalInverseGammaDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:NormalInverseGammaDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="NormalInverseGammaDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="mean" type="un:ContinuousValuesType"/>
```

```
        <xs:element name="varianceScaling"
type="un:PositiveRealValuesType"/>
        <xs:element name="shape" type="un:PositiveRealValuesType"/>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:NormalInverseGammaDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14</un:mean>
  <un:varianceScaling>3.14</un:varianceScaling>
  <un:shape>3.14</un:shape>
  <un:scale>3.14</un:scale>
</un:NormalInverseGammaDistribution>

<!-- Multiple values -->
<un:NormalInverseGammaDistribution
xmlns:un="http://www.uncertml.org/2.0">
  <un:mean>3.14 6.28 9.42</un:mean>
  <un:varianceScaling>3.14 6.28 9.42</un:varianceScaling>
  <un:shape>3.14 6.28 9.42</un:shape>
  <un:scale>3.14 6.28 9.42</un:scale>
</un:NormalInverseGammaDistribution>
```

JSON

```
// Single value
{"NormalInverseGammaDistribution": {"mean": [3.14],
"varianceScaling": [3.14], "shape": [3.14], "scale": [3.14]}}

// Multiple values
{"NormalInverseGammaDistribution": {"mean": [3.14, 6.28, 9.42],
"varianceScaling": [3.14, 6.28, 9.42], "shape": [3.14, 6.28, 9.42],
"scale": [3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
NormalInverseGammaDistribution nigd = new
NormalInverseGammaDistribution(3.14,  3.14,  3.14,  3.14);

// Multiple value declaration
NormalInverseGammaDistribution nigd = new
NormalInverseGammaDistribution(new double[] {3.14,  6.28,  9.42},
new double[] {3.14,  6.28,  9.42},  new double[] {3.14,  6.28,
9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
NormalInverseGammaDistribution nigd =
(NormalInverseGammaDistribution)xml.parse(new File("normal-inverse-
gamma-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
NormalInverseGammaDistribution nigd =
(NormalInverseGammaDistribution)json.parse(new File("normal-inverse-
gamma-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(nigd,  new File("normal-inverse-gamma-
distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(nigd,  new File("normal-inverse-gamma-
distribution.json"));
```

Pareto distribution

URI	http://www.uncertml.org/distributions/pareto
UncertML name	ParetoDistribution
Alternative names	Bradford distribution

Definition	A random variable x follows a Pareto distribution if the probability density function is of the form shown below. The distribution allows for the specification of a minimum value below which the density is 0. It is a skewed heavy-tailed distribution.
Parameters	x_m (`scale`) a positive real (minimum value), α (`shape`) a positive real.
Support	$x \geq x_m$ real.
PDF	$f(x; x_m, \alpha) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}}$
Source	http://en.wikipedia.org/wiki/Pareto_distribution
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Pareto_distribution

Schema

```xml
<!-- Element -->
<xs:element name="ParetoDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:ParetoDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="ParetoDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
        <xs:element name="shape" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Single value -->
<un:ParetoDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:scale>3.14</un:scale>
  <un:shape>3.14</un:shape>
</un:ParetoDistribution>

<!-- Multiple values -->
<un:ParetoDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:scale>3.14 6.28 9.42</un:scale>
  <un:shape>3.14 6.28 9.42</un:shape>
</un:ParetoDistribution>
```

JSON

```
// Single value
{"ParetoDistribution": {"scale": [3.14], "shape": [3.14]}}

// Multiple values
{"ParetoDistribution": {"scale": [3.14, 6.28, 9.42], "shape": [3.14,
6.28, 9.42]}}
```

Java API

```java
// Single value declaration
ParetoDistribution pd = new ParetoDistribution(3.14,  3.14);

// Multiple value declaration
ParetoDistribution pd = new ParetoDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
ParetoDistribution pd = (ParetoDistribution)xml.parse(new
File("pareto-distribution.xml"));
```

```
// Parsing from a JSON file
JSONParser json = new JSONParser();
ParetoDistribution pd = (ParetoDistribution)json.parse(new
File("pareto-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(pd,  new File("pareto-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(pd,  new File("pareto-distribution.json"));
```

Poisson distribution

URI	http://www.uncertml.org/distributions/poisson
UncertML name	PoissonDistribution
Alternative names	N/A
Definition	A random variable x follows a Poisson distribution if the probability mass function (pmf) is of the form shown below. The Poisson distribution can be used to model the number of events occurring within fixed time period of time.
Parameters	λ (`rate`) a positive real.
Support	x a non-negative integer.
PDF	$f(x;\lambda) = \frac{\lambda^x}{x!}\exp(-\lambda)$
Categories	ordinal variables, distribution
Categories	http://en.wikipedia.org/wiki/Poisson_distribution
Further information	http://en.wikipedia.org/wiki/Poisson_distribution

Schema

```
<!-- Element -->
<xs:element name="PoissonDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:PoissonDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="PoissonDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="rate" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:PoissonDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:rate>3.14</un:rate>
</un:PoissonDistribution>

<!-- Multiple values -->
<un:PoissonDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:rate>3.14 6.28 9.42</un:rate>
</un:PoissonDistribution>
```

JSON	`// Single value` `{"PoissonDistribution": {"rate": [3.14]}}` `// Multiple values` `{"PoissonDistribution": {"rate": [3.14, 6.28, 9.42]}}`
Java API	see code below

```
// Single value declaration
PoissonDistribution pd = new PoissonDistribution(3.14);

// Multiple value declaration
PoissonDistribution pd = new PoissonDistribution(new double[] {3.14,
6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
PoissonDistribution pd = (PoissonDistribution)xml.parse(new
File("poisson-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
PoissonDistribution pd = (PoissonDistribution)json.parse(new
File("poisson-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(pd,  new File("poisson-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(pd,  new File("poisson-distribution.json"));
```

Student T distribution

URI	http://www.uncertml.org/distributions/student-t
UncertML name	StudentTDistribution
Alternative names	T distribution
Definition	A random variable x follows a Student-t distribution if the probability density function (pdf) is of the form shown below. 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, degrees of freedom (d.f) and variance. 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 multivariate extension to higher dimensions is also described in this dictionary.
Parameters	μ (`location`) or mean, a real, σ^2 (`scale`) or variance, a positive real, ν (`degreesOfFreedom`) a positive integer.
Support	x a real.
PDF	$f(x; \mu, \sigma^2, \nu) = \frac{\Gamma(\nu/2+1/2)}{\Gamma(\nu/2)(\pi\nu\sigma^2)^{1/2}} \left[1 + \frac{(x-\mu)^2}{\nu\sigma^2}\right]^{-\nu/2-1/2}$
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.

Categories

continuous variables, distribution

Further information

http://en.wikipedia.org/wiki/Student's_t-distribution

Schema

```xml
<!-- Element -->
<xs:element name="StudentTDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:StudentTDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="StudentTDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="location" type="un:ContinuousValuesType"/>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
        <xs:element name="degreesOfFreedom"
type="un:PositiveNaturalNumbersType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```xml
<!-- Single value -->
<un:StudentTDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14</un:location>
  <un:scale>3.14</un:scale>
  <un:degreesOfFreedom>3</un:degreesOfFreedom>
</un:StudentTDistribution>

<!-- Multiple values -->
<un:StudentTDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:location>3.14 6.28 9.42</un:location>
  <un:scale>3.14 6.28 9.42</un:scale>
  <un:degreesOfFreedom>3 6 9</un:degreesOfFreedom>
</un:StudentTDistribution>
```

JSON

```
// Single value
{"StudentTDistribution": {"mean": [3.14], "variance": [3.14],
"degreesOfFreedom": [3]}}

// Multiple values
{"StudentTDistribution": {"mean": [3.14, 6.28, 9.42], "variance":
[3.14, 6.28, 9.42], "degreesOfFreedom": [3, 6, 9]}}
```

Java API

```java
// Single value declaration
StudentTDistribution std = new StudentTDistribution(3.14,  3.14,
3);

// Multiple value declaration
StudentTDistribution std = new StudentTDistribution(new double[]
{3.14,  6.28,  9.42},  new double[] {3.14,  6.28,  9.42},  new int[]
{3,  6,  9});

// Parsing from an XML file
XMLParser xml = new XMLParser();
StudentTDistribution std = (StudentTDistribution)xml.parse(new
File("student-t-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
StudentTDistribution std = (StudentTDistribution)json.parse(new
File("student-t-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(std,  new File("student-t-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(std,  new File("student-t-distribution.json"));
```

Uniform distribution

URI	http://www.uncertml.org/distributions/uniform
UncertML name	UniformDistribution
Alternative names	N/A
Definition	A random variable x follows a uniform distribution if the probability density function (pdf) is of the form shown below. The distribution assigns equal probability to all events within the chosen domain.
Parameters	a (`minimum`) a real, b (`maximum`) a real, such that $a < b$.
Support	x a real $\in [a, b]$
PDF	$f(x; a, b) = \frac{1}{b-a}$
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.
Categories	continuous variables, ordinal variables, distribution
Further information	http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)

Schema

```
<!-- Element -->
<xs:element name="UniformDistribution"
substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:UniformDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="UniformDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:choice>
        <xs:sequence>
          <xs:element name="minimum"
type="un:ContinuousValuesType"/>
          <xs:element name="maximum"
type="un:ContinuousValuesType"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="numberOfClasses"
type="un:PositiveNaturalNumbersType"/>
        </xs:sequence>
      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:UniformDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:minimum>3.14</un:minimum>
  <un:maximum>6.28</un:maximum>
</un:UniformDistribution>

<!-- Multiple values -->
<un:UniformDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:minimum>3.14 6.28 9.42</un:minimum>
  <un:maximum>6.28 12.57 18.85</un:maximum>
</un:UniformDistribution>
```

JSON

```
// Single value
```

	`{"UniformDistribution": {"minimum": [3.14], "maximum": [6.28]}}` `// Multiple values` `{"UniformDistribution": {"minimum": [3.14, 6.28, 9.42], "maximum": [6.28, 12.57, 18.85]}}`
Java API	`// Single value declaration` `UniformDistribution ud = new UniformDistribution(3.14,  6.28);` `// Multiple value declaration` `UniformDistribution ud = new UniformDistribution(new double[] {3.14, 6.28,  9.42},  new double[] {6.28,  12.57,  18.85});` `// Parsing from an XML file` `XMLParser xml = new XMLParser();` `UniformDistribution ud = (UniformDistribution)xml.parse(new File("uniform-distribution.xml"));` `// Parsing from a JSON file` `JSONParser json = new JSONParser();` `UniformDistribution ud = (UniformDistribution)json.parse(new File("uniform-distribution.json"));` `// Encoding to an XML file` `XMLEncoder xEncoder = new XMLEncoder();` `xEncoder.encode(ud,  new File("uniform-distribution.xml"));` `// Encoding to a JSON file` `JSONEncoder jEncoder = new JSONEncoder();` `jEncoder.encode(ud,  new File("uniform-distribution.json"));`

Weibull distribution

URI	http://www.uncertml.org/distributions/weibull
UncertML name	WeibullDistribution
Alternative names	Frechet distribution, Weibull-Gnedenko distribution
Definition	A random variable x follows an Weibull distribution if the probability density function (pdf) is of the form shown below. It includes the exponential distribution as a special case. It is often used in engineering and finance.
Parameters	λ (`scale`) a positive real, k (`shape`) a positive real.
Support	x a non-negative real.
PDF	$f(x;\lambda,k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}\exp(-x/\lambda)^k$
Source	Johnson, Kotz and Balakrishnan, "Continuous univariate Distributions", Volume 2, 1995, John Wiley and Sons.
Categories	continuous variables, distribution
Further information	http://en.wikipedia.org/wiki/Weibull_distribution
Schema	`<!-- Element -->` `<xs:element name="WeibullDistribution" substitutionGroup="un:AbstractDistribution">` `  <xs:complexType>` `    <xs:complexContent>` `      <xs:extension base="un:WeibullDistributionType"/>` `    </xs:complexContent>` `  </xs:complexType>` `</xs:element>` `<!-- Complex type -->`

```
<xs:complexType name="WeibullDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="scale" type="un:PositiveRealValuesType"/>
        <xs:element name="shape" type="un:PositiveRealValuesType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Single value -->
<un:WeibullDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:scale>3.14</un:scale>
  <un:shape>3.14</un:shape>
</un:WeibullDistribution>

<!-- Multiple values -->
<un:WeibullDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:scale>3.14 6.28 9.42</un:scale>
  <un:shape>3.14 6.28 9.42</un:shape>
</un:WeibullDistribution>
```

JSON

```
// Single value
{"WeibullDistribution": {"scale": [3.14], "shape": [3.14]}}

// Multiple values
{"WeibullDistribution": {"scale": [3.14, 6.28, 9.42], "shape":
[3.14, 6.28, 9.42]}}
```

Java API

```
// Single value declaration
WeibullDistribution wd = new WeibullDistribution(3.14,  3.14);

// Multiple value declaration
WeibullDistribution wd = new WeibullDistribution(new double[] {3.14,
6.28,  9.42},  new double[] {3.14,  6.28,  9.42});

// Parsing from an XML file
XMLParser xml = new XMLParser();
WeibullDistribution wd = (WeibullDistribution)xml.parse(new
File("weibull-distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
WeibullDistribution wd = (WeibullDistribution)json.parse(new
File("weibull-distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(wd,  new File("weibull-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(wd,  new File("weibull-distribution.json"));
```

Wishart distribution

URI	http://www.uncertml.org/distributions/wishart
UncertML name	WishartDistribution
Alternative names	N/A
Definition	A random matrix variable $\mathbf{X}$ of size $D \times D$ follows a Wishart distribution if the probability density function is of the form shown below. The Wishart distributon is the conjugate prior for the inverse of a covariance matrix of a Multivariate Normal distribution. It is a generalistion of the Gamma distribution to higher dimensions. In one dimesion the Wishart distribution is equivalent to a Gamma with parameters $k = \nu/2$ and $\theta = 1/2\mathbf{W}$.
Parameters	$\mathbf{W}$ (`scaleMatrix`), a D dimensional positive definite symmetric matrix,

	ν (degreesOfFreedom) a positive real $> D-1$
Support	Space of $\mathbf{X}$ positive definite matrices of size $D \times D$.
PDF	$f(\mathbf{X};\mathbf{W},\nu) = \det(\mathbf{W})^{-\nu/2}\left(2^{\nu D/2}\pi^{D(D-1)/4}\prod_{i=1}^{D}\Gamma\left(\frac{\nu+1-i}{2}\right)\right)^{-1}\det(\mathbf{X})^{(\nu-D-1)/2}\exp\left(-\frac{1}{2}\mathrm{Tr}(\mathbf{W}^{-1}\mathbf{X})\right)$ where $\det$ denotes the determinant, Tr the matrix trace.
Source	Christopher M. Bishop, "Pattern Recognition and Machine Learning", 2006, Springer.
Categories	continuous variables, distribution, multivariate
Further information	http://en.wikipedia.org/wiki/Wishart_distribution

Schema

```
<!-- Element -->
<xs:element name="WishartDistribution" substitutionGroup="un:AbstractDistribution">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:WishartDistributionType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="WishartDistributionType">
  <xs:complexContent>
    <xs:extension base="un:AbstractDistributionType">
      <xs:sequence>
        <xs:element name="degreesOfFreedom" type="un:positiveRealNumber"/>
        <xs:element name="scaleMatrix" type="un:CovarianceMatrixType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:WishartDistribution xmlns:un="http://www.uncertml.org/2.0">
  <un:degreesOfFreedom>3.14</un:degreesOfFreedom>
  <un:scaleMatrix dimension="2">
    <un:values>3.14 0.0 0.0 3.14</un:values>
  </un:scaleMatrix>
</un:WishartDistribution>
```

JSON

```
// Multiple values
{"WishartDistribution": {"degreesOfFreedom":3.14, "scaleMatrix": {"dimension":2,
"values": [3.14, 0.0, 0.0, 3.14]}}}
```

Java API

```
// Multiple value declaration
WishartDistribution wd = new WishartDistribution(3.14,  new CovarianceMatrix(2,  new
double[] {3.14,  0.0,  0.0,  3.14}));

// Parsing from an XML file
XMLParser xml = new XMLParser();
WishartDistribution wd = (WishartDistribution)xml.parse(new File("wishart-
distribution.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
WishartDistribution wd = (WishartDistribution)json.parse(new File("wishart-
distribution.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(wd,  new File("wishart-distribution.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(wd,  new File("wishart-distribution.json"));
```

Samples

Realisation

URI

http://www.uncertml.org/samples/realisation

UncertML name

Realisation

Alternative names

Sample, Observed value, Simulated value

Definition

A realisation is a singe instance of a random variable and can be used to mean an observed value, or, as more widely used in UncertML, a single draw, x_i, from a probability distribution, $p(x)$. x can uni- or multi-variate, and can have values that are continuous, ordinal or categorical. Most Monte Carlo methods produce realisations from the predictive or posterior distribution of the variable(s) of interest.

Parameters

`ID` (optional) identifier of the sample which could be needed in tracking samples in Monte Carlo
`weight` (optional) weight of the realisation - for use where the samples are not drawn from the underlying distribution, typically these should be normalised, but do not need to be so.

Source

N/A

Categories

sample, continuous variables, ordinal variables, categorical variables, multivariate

Further information

http://en.wikipedia.org/wiki/Realization_(probability)

Schema

```
<!-- Element -->
<xs:element name="Realisation"
substitutionGroup="un:AbstractUncertainty">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:RealisationType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="RealisationType">
  <xs:complexContent>
    <xs:extension base="un:AbstractUncertaintyType">
      <xs:sequence>
        <xs:element name="weight" type="xs:double" minOccurs="0"/>
        <xs:choice>
          <xs:element name="values" type="un:ContinuousValuesType"/>
          <xs:element name="categories"
type="un:CategoricalValuesType"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:Realisation id="ID" xmlns:un="http://www.uncertml.org/2.0">
  <un:weight>0.5</un:weight>
  <un:values>3.14 6.28 9.42</un:values>
</un:Realisation>

<!-- Categorical values -->
<un:Realisation id="ID" xmlns:un="http://www.uncertml.org/2.0">
  <un:weight>0.5</un:weight>
  <un:categories>Red Green Blue</un:categories>
</un:Realisation>
```

JSON

	`` // Multiple values {"Realisation": {"values": [3.14, 6.28, 9.42], "id":"ID", "weight":0.5}} // Categorical values {"Realisation": {"categories": ["Red", "Green", "Blue"], "id":"ID", "weight":0.5}} ```
Java API	``` // Multiple value declaration Realisation r = new Realisation(new double[] {3.14, 6.28, 9.42}, 0.5, "ID"); // Categorical value declaration Realisation r = new Realisation(new String[] {"Red", "Green", "Blue"}, 0.5, "ID"); // Parsing from an XML file XMLParser xml = new XMLParser(); Realisation r = (Realisation)xml.parse(new File("realisation.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); Realisation r = (Realisation)json.parse(new File("realisation.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(r, new File("realisation.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(r, new File("realisation.json")); ```
Value constraints	**values**: *any real number* or *any integer* or *any category* which can be vectors for multivariate distributions

Random sample

URI	http://www.uncertml.org/samples/random
UncertML name	[RandomSample]()
Alternative names	Sample
Definition	A random sample is a set of independent realisations, x_i, drawn from a probability distribution $p(x)$ (or alternatively a population where every member has an equal chance of being drawn, but is randomly selected). The sample will typically be obtained using some form of simulation algorithm for base distributions using a random number generator of some sort. For more complex probability distributions they might typically be obtained using Monte Carlo methods or Markov Chain Monte Carlo methods. It is possible that some form of weighting is given to each sample, for example derived from an importance sampling method, such as a particle filter. For finite samples there will be some induced sampling error.
Parameters	`samplingMethodDescription` (optional) provides a textual description of the method used to obtain the samples.
Source	N/A
Categories	sample, continuous variables, ordinal variables, categorical variables, multivariate
Further information	http://en.wikipedia.org/wiki/Random_sample
Schema	``` <!-- Element --> <xs:element name="RandomSample" substitutionGroup="un:AbstractSample"> <xs:complexType> ```

	`` <xs:complexContent> <xs:extension base="un:RandomSampleType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="RandomSampleType"> <xs:complexContent> <xs:extension base="un:AbstractSampleType"> <xs:sequence> <xs:element ref="un:Realisation" maxOccurs="unbounded"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType> ```
XML	``` <!-- Multiple values --> <un:RandomSample xmlns:un="http://www.uncertml.org/2.0"> <un:samplingMethodDescription>Importance sampler with uniform proposal distribution</un:samplingMethodDescription> <un:Realisation id="ID"> <un:weight>0.5</un:weight> <un:values>3.14 6.28 9.42</un:values> </un:Realisation> <un:Realisation id="ID"> <un:weight>0.25</un:weight> <un:values>3.14 6.28 9.42</un:values> </un:Realisation> <un:Realisation id="ID"> <un:weight>0.25</un:weight> <un:values>3.14 6.28 9.42</un:values> </un:Realisation> </un:RandomSample> ```
JSON	``` // Multiple values {"RandomSample": {"samplingMethodDescription":"Importance sampler with uniform proposal distribution", "realisations": [{"values": [3.14, 6.28, 9.42], "id":"ID", "weight":0.5}, {"values": [3.14, 6.28, 9.42], "id":"ID", "weight":0.25}, {"values": [3.14, 6.28, 9.42], "id":"ID", "weight":0.25}]}} ```
Java API	``` // Multiple value declaration RandomSample rs = new RandomSample(new Realisation[] {new Realisation(new double[] {3.14, 6.28, 9.42}, 0.5, "ID"), new Realisation(new double[] {3.14, 6.28, 9.42}, 0.25, "ID"), new Realisation(new double[] {3.14, 6.28, 9.42}, 0.25, "ID")}, "Importance sampler with uniform proposal distribution"); // Parsing from an XML file XMLParser xml = new XMLParser(); RandomSample rs = (RandomSample)xml.parse(new File("random-sample.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); RandomSample rs = (RandomSample)json.parse(new File("random-sample.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(rs, new File("random-sample.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(rs, new File("random-sample.json")); ```
Value constraints	**values**: *any real number* or *any integer* or *any category* which can be vectors for multivariate distributions

Systematic sample

URI	http://www.uncertml.org/samples/systematic
UncertML name	SystematicSample

Alternative names	Deterministic sample, Unscented sample
Definition	A systematic sample is a set of often carefully chosen realisations, x_i, taken from a probability distribution $p(x)$ (or alternatively a population where selection is used preferentially sample from certain targets). The sample will typically be obtained using some form of deterministic algorithm such as the [unscented sampling methods](#) for Gaussian random variables. Commonly some form of weighting is given to each sample. Note that for finite samples there will be some induced sampling error.
Parameters	`samplingMethodDescription` (optional) provides a textual description of the method used to obtain the samples.
Source	N/A
Categories	sample, continuous variables, ordinal variables, categorical variables, multivariate
Further information	http://en.wikipedia.org/wiki/Systematic_sampling

Schema

```
<!-- Element -->
<xs:element name="SystematicSample"
substitutionGroup="un:AbstractSample">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="un:SystematicSampleType"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<!-- Complex type -->
<xs:complexType name="SystematicSampleType">
  <xs:complexContent>
    <xs:extension base="un:AbstractSampleType">
      <xs:sequence>
        <xs:element ref="un:Realisation" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML

```
<!-- Multiple values -->
<un:SystematicSample xmlns:un="http://www.uncertml.org/2.0">
  <un:samplingMethodDescription>Unscented
sample</un:samplingMethodDescription>
  <un:Realisation id="ID">
    <un:weight>0.5</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
  <un:Realisation id="ID">
    <un:weight>0.25</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
  <un:Realisation id="ID">
    <un:weight>0.25</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
</un:SystematicSample>
```

JSON

```
// Multiple values
{"SystematicSample": {"samplingMethodDescription":"Unscented sample",
"realisations": [{"values": [3.14, 6.28, 9.42], "id":"ID",
"weight":0.5}, {"values": [3.14, 6.28, 9.42], "id":"ID",
"weight":0.25}, {"values": [3.14, 6.28, 9.42], "id":"ID",
"weight":0.25}]}}
```

Java API

```
// Multiple value declaration
SystematicSample ss = new SystematicSample(new Realisation[] {new
Realisation(new double[] {3.14,  6.28,  9.42},  0.5,  "ID"),  new
Realisation(new double[] {3.14,  6.28,  9.42},  0.25,  "ID"),  new
Realisation(new double[] {3.14,  6.28,  9.42},  0.25,  "ID")},
"Unscented sample");
```

	``// Parsing from an XML file XMLParser xml = new XMLParser(); SystematicSample ss = (SystematicSample)xml.parse(new File("systematic-sample.xml")); // Parsing from a JSON file JSONParser json = new JSONParser(); SystematicSample ss = (SystematicSample)json.parse(new File("systematic-sample.json")); // Encoding to an XML file XMLEncoder xEncoder = new XMLEncoder(); xEncoder.encode(ss, new File("systematic-sample.xml")); // Encoding to a JSON file JSONEncoder jEncoder = new JSONEncoder(); jEncoder.encode(ss, new File("systematic-sample.json"));```
Value constraints	**values**: *any real number* or *any integer* or *any category* which can be vectors for multivariate distributions

Unknown sample

URI	http://www.uncertml.org/samples/unknown
UncertML name	UnknownSample
Alternative names	Scenarios, Ensembles
Definition	An unknown sample is a set of independent realisations, x_i, drawn from some distribution $p(x)$ although without any known properties. For example in many applications of ensemble methods in atmospheric science there is no clear identification of what the ensemble is actually sampled from. It is possible that some form of weighting is given to each sample. For finite samples there will be some induced sampling error, but here there are also issues of what the sample represents. This type should only be used where these is no knowledge of the sampling method used to obtain the realisations. It is weaker than RandomSample and SystematicSample which should be used if the sampling method is known.
Parameters	`samplingMethodDescription` (optional) provides a textual description of the method used to obtain the samples.
Source	N/A
Categories	sample, continuous variables, ordinal variables, categorical variables, multivariate
Further information	N/A
Schema	```<!-- Element --> <xs:element name="UnknownSample" substitutionGroup="un:AbstractSample"> <xs:complexType> <xs:complexContent> <xs:extension base="un:UnknownSampleType"/> </xs:complexContent> </xs:complexType> </xs:element> <!-- Complex type --> <xs:complexType name="UnknownSampleType"> <xs:complexContent> <xs:extension base="un:AbstractSampleType"> <xs:sequence> <xs:element ref="un:Realisation" maxOccurs="unbounded"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>```
XML	

```
<!-- Multiple values -->
<un:UnknownSample xmlns:un="http://www.uncertml.org/2.0">
  <un:samplingMethodDescription>Scenarios from expert
consultation</un:samplingMethodDescription>
  <un:Realisation id="ID">
    <un:weight>0.5</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
  <un:Realisation id="ID">
    <un:weight>0.25</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
  <un:Realisation id="ID">
    <un:weight>0.25</un:weight>
    <un:values>3.14 6.28 9.42</un:values>
  </un:Realisation>
</un:UnknownSample>
```

JSON

```
// Multiple values
{"UnknownSample": {"samplingMethodDescription":"Scenarios from expert
consultation", "realisations": [{"values": [3.14, 6.28, 9.42],
"id":"ID", "weight":0.5}, {"values": [3.14, 6.28, 9.42], "id":"ID",
"weight":0.5}, {"values": [3.14, 6.28, 9.42], "id":"ID",
"weight":0.5}]}}
```

Java API

```
// Multiple value declaration
UnknownSample us = new UnknownSample(new Realisation[] {new
Realisation(new double[] {3.14,  6.28,  9.42},  0.5,  "ID"),  new
Realisation(new double[] {3.14,  6.28,  9.42},  0.5,  "ID"),  new
Realisation(new double[] {3.14,  6.28,  9.42},  0.5,  "ID")},
"Scenarios from expert consultation");

// Parsing from an XML file
XMLParser xml = new XMLParser();
UnknownSample us = (UnknownSample)xml.parse(new File("unknown-
sample.xml"));

// Parsing from a JSON file
JSONParser json = new JSONParser();
UnknownSample us = (UnknownSample)json.parse(new File("unknown-
sample.json"));

// Encoding to an XML file
XMLEncoder xEncoder = new XMLEncoder();
xEncoder.encode(us,  new File("unknown-sample.xml"));

// Encoding to a JSON file
JSONEncoder jEncoder = new JSONEncoder();
jEncoder.encode(us,  new File("unknown-sample.json"));
```

Value constraints

values: *any real number* or *any integer* or *any category* which can be vectors for multivariate distributions