HS^3 - Overview of supported types and components

06. April 2023

1 Introduction

With the introduction of `pyhf` [1], a `JSON` format for likelihood serialization has been put forward. However, an interoperable format that encompasses likelihoods with a scope beyond stacks of binned histograms was sorely lacking. With the release of `ROOT` 6.26/00 [2] and the experimental `RooJSONFactoryWSTool` therein, this gap has now been filled.

This document sets out to document the syntax and features of the HEP Statistics Serialization Standard (HS^3) for likelihoods and statistical models in general, as to be adopted by any HS^3-compatible statistics framework.

Please note that this document as well as the HS^3 standard are still in development and can still undergo minor and major changes in the future. This document describes the syntax of version 0.2 of the HS^3 standard.

1.1 How to read

In the context of this document, any `JSON` object is referred to as a component. A key-value-pair inside such a component is referred to as a component. If not explicitly stated otherwise, all components mentioned are mandatory.

The components located inside the top-level object are referred to as top-level components.

1.2 Terms and Types

This is a list of used types and terms in this document.

`struct`	*key:value* mapping, keys are of type **string**. Represented with { }
`array`	array of items (either **strings** or **numbers**) without keys. Represented with [ ]
`string`	references to objects, names and arbitrary information. Represented with “ ”
`number`	either floating or integer type values
`boolean`	Boolean values; they can be encoded as `true` and `false` or `1` and `0` respectively

All structs inside an array should always have a component `name`.

Within most top-level components, any one `string` given as a value to any component should always refer to the `name` of another, fully qualified component, unless explicitly stated otherwise. Top-level components in which this is not the case are explicitly marked as such.

2 Top-level components

In the following the top-level components of HS^3 and their parameters/arguments are described. Each component is completely *optional*, but certain components might depend on other components, which need to be provided in that case. The only exception is the component `metadata` containing the version of HS^3, which is always required. In short the supported top-level components are

`distributions` (optional) array of distributions

`functions` (optional) array of mathematical functions

`data` (optional) array of data (or simulated data)

`likelihoods` (optional) array of combinations of distributions and data

`domains` (optional) array of domains, describing ranges of parameters;

`parameter_points` (optional) array of parameter points, to be used as starting points for minimizations or to document best-fit-values or nominal truth values of datasets

`analyses` (optional) array of suggested analyses to be run on the models in this file

`metadata` *required* struct containing meta information; *required* HS^3 version number, (optional), e.g., authors, paper references, package versions, data/analysis descriptions

`misc` (optional) struct containing miscellaneous information, e.g. optimizer settings, plotting colors, etc.

In the following each of these are described in more detail with respect to their own structure.

2.1 Distributions

The top-level component `distributions` contains an array of distributions in struct format. Each distribution has to have the components `type` denoting the kind of distribution described and a component `name`. Distributions in general have the following keys:

`name` custom string

`type` string that determines the kind of distribution, e.g. `gaussian_dist`

`...` each distribution has individual parameter components for the various individual parameters. For example, distributions of type `gaussian_dist` have the specific components `mean`, `sigma` and `x`. In general, these components can contain strings as references to other objects, numbers or boolean values. Depending on the parameter and the type of distribution, they appear either in single item or array format.

Example: Distributions

```
"distributions":[
  {
    "name":"gauss1",
    "type":"gaussian_dist",
    "mean":1.0,
    "sigma":"param_sigma",
    "x":"param_x"
  },
  {
    "name":"exp1",
    "type":"exponential_dist",
    "c":-2,
    "x":"data_x"
  },
  ...
]
```

In the following all implemented distributions are listed in detail.

2.1.1 Exponential distribution

The PDF of the exponential distribution is defined as

$$\text{ExponentialPdf}(x, c) = \mathcal{N} \cdot \exp(c \cdot x), \tag{1}$$

where $\mathcal{N}$ is a normalisation constant that depends on the range and values of the arguments.

`name`	custom string
`type`	exponential_dist
`c`	number or name of the parameter used as coefficient c
`x`	number or name of the variable x.

2.1.2 Gaussian/Normal distribution

The PDF of a Gaussian/Normal distribution is defined as

$$\text{GaussianPdf}(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{(x-\mu)^2}{\sigma^2}\right). \tag{2}$$

`name`	custom string
`type`	gaussian_dist *or* normal_dist
`x`	number or name of the variable x
`mean`	number or name of the parameter used as mean value μ
`sigma`	number or name of the parameter encoding the root-mean-square width σ.

2.1.3 Log-normal Distribution

The PDF of the Log-normal distribution is defined as

$$\text{LogNormalPDF}(x, m_0, k) = \frac{1}{\sqrt{2\pi} \cdot \ln(k) \cdot x} \cdot \exp\left(\frac{-\ln^2(\frac{x}{m_0})}{2\ln^2(k)}\right). \tag{3}$$

`name`	custom string
`type`	lognormal_dist
`x`	number or name of the variable x
`mean`	number or name of the parameter used as mean value m_0
`k`	number or name of the parameter k describing the shape

2.1.4 Multivariate Normal distribution

The PDF of the multivariate normal distribution is defined as

$$\text{MvNormalPDF}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-k/2}\det(\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right), \tag{4}$$

with $\boldsymbol{\Sigma}$ being positive-definite.

`name`	custom string
`type`	multinormal_dist
`x`	array of numbers or names of the variables $\mathbf{x}$
`mean`	array of length k of the parameters used as locations $\boldsymbol{\mu}$
`covariances`	array of arrays of dimension $k \times k$ containing numbers or names for the covariance matrix $\boldsymbol{\Sigma}$. In general, the covariance matrix $\boldsymbol{\Sigma}$ needs to be positive semi-definite.

2.1.5 Poisson distribution

The PDF of a Poisson distribution is defined as

$$P_\lambda(k) = \frac{\lambda^k}{k!}\,\mathrm{e}^{-\lambda}. \tag{5}$$

`type`	poisson_dist
`x`	number or name of the variable x serving as observable k.
`mean`	number or name of the parameter used as mean λ.

2.1.6 Argus Background distribution

The PDF of the Argus background distribution is defined as

$$\text{Argus}(m, m_0, c, p) = \mathcal{N} \cdot m \cdot \left[1 - \left(\frac{m}{m_0}\right)^2\right]^p \cdot \exp\left[c \cdot \left(1 - \left(\frac{m}{m_0}\right)^2\right)\right] \tag{6}$$

and describes the ARGUS background shape.

`type`	argus_dist
`mass`	number or name of the variable m used as mass
`resonance`	number or name of the parameter used as resonance m_0
`slope`	number or name of the parameter used as slope c
`power`	number or name of the parameter used as exponent p.

2.1.7 Addition of distributions

The PDF of this so-called Mixture Distribution is a sum of PDFs f_i:

$$\text{MixturePdf}(x) = \sum_{i=1}^{n-1} c_i * f_i(\vec{x}) \tag{7}$$

where the c_i are coefficients and $\vec{x}$ is the vector of variables. If the number of coefficients is one less than the number of distributions, c_n is computed from the other coefficients as

$$c_n = 1 - \sum_{i=1}^{n-1} c_i \tag{8}$$

`name`	custom string
`type`	mixture_dist
`summands`	array of names referencing distributions
`coefficients`	array of names of coefficients c_i or numbers to be added

2.1.8 Product Distribution

The PDF of the product of PDFs of independent distributions f_i is defined as

$$\text{ProductPdf}(x) = \prod_{i}^{n} f(x). \tag{9}$$

`name`	custom string
`type`	product_dist
`factors`	array of names referencing distributions

2.1.9 Continuous Uniform Distribution

The PDF of a continuous uniform distribution is defined as:

$$\text{UniformPdf}(x, a, b) = \frac{1}{b - a} \quad a \leq x \leq b. \tag{10}$$

`name` custom string

`type` uniform_dist

`max` number or name of the parameter used as upper bound of range b

`min` number or name of the parameter used as lower bound of range a

2.1.10 Polynomial Distribution

The PDF of a polynomial distribution is defined as

$$\text{PolynomialPdf}(x, a_0, a_1, a_2, ...) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + ... \tag{11}$$

`name` custom string

`type` polynomial_dist

`x` number or name of the variable x

`coefficients` array of coefficients a_i. The length of this array implies the degree of the polynomial.

2.1.11 HistFactory Distribution

HistFactory [3] is a language to describe statistical models consisting only of ”histograms” (which is used interchangeably with ”step-functions” in this context). Each HistFactory distribution describes one ”channel” or ”region” of a binned measurement, containing a stack of ”samples”, i. e. binned distributions sharing the same binning (step-functions describing the signal or background of a measurement). Such a model is shown in Figure 1. Each of the contributions may be subject to `modifiers`.

$$\text{HistFactoryPdf}(x, \vec{\theta}) = \prod_{b \in \text{bins}} \text{Poisson}\,(n|\lambda) \tag{12}$$

where λ is the prediction in the given bin. This prediction is computed as

$$\lambda = \sum_{s \in \text{samples}} \left[\left(d_s(x) + \sum_{\delta \in M_\delta} \delta(x, \theta_\delta) \right) \prod_{\kappa \in M_\kappa} \kappa(x, \theta_\kappa) \right] \tag{13}$$

Here $d_s(x)$ is the distribution associated with the sample s, a step function

$$d_s(x) = \chi_b^{y_s}(x) \tag{14}$$

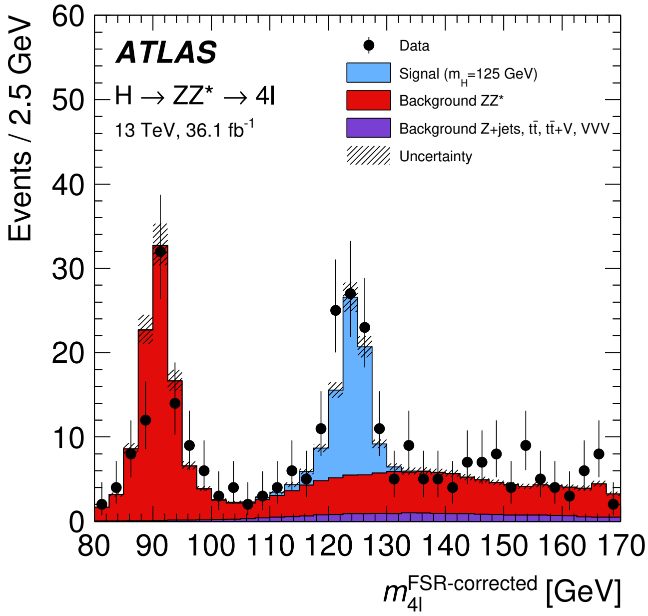

Figure 1: A binned statistical model describing a High Energy Physics measurement, in this case of the $H \to 4\ell$ process by the ATLAS collaboration.

In this section, $\chi_b^{y_s}(x)$ denotes a generic step function in the binning b such that $\chi_b(x) = y_{s,i}$, some constant, if $x \in [b_i, b_{i+1})$. The $y_{s,i}$ in this case are the bin contents (yields) of the histograms.

The M_κ are the multiplicative modifiers, the M_δ are the additive modifiers. Each of the modifiers is either multiplicative (κ) or additive (δ). All samples and modifiers share the same binning b.

The modifiers depend on a set of nuisance parameters θ, where each modifier can only depend on one θ_i, but the θ_i can take the form of vectors and the same θ_i can be shared by several modifiers. By convention, these are denoted α if they affect all bins in a correlated way, and γ if they affect only one bin at a time. The types of modifiers are

- A *uncorrelated shape systematic* or `shapesys` modifier is a multiplicative modifier that scales each single bin by the value of some independent parameter γ. Here, $\theta_i = \vec{\gamma}$, where the length of $\vec{\gamma}$ is equal to the number of bins in this region.
- A *correlated shape systematic* or `histosys` modifier is an additive modifier that adds or subtracts a constant step function χ^f, scaled with a single factor α. The modifier contains a `data` section, which contains the subsections `hi` and `lo` that help to define the step function χ^f. They contain `contents`, which define the bin-wise additions or subtractions for $\alpha = 1$. Here, $\theta_i = \alpha$.
- A *normalization systematic* or `normsys` modifier is a multiplicative modifier that scales the entire sample in this region with the same constant factor f that is a function of α. The modifier contains a `data` section, which contains the values `hi` and `lo` that help to define f. There are different functional forms that can be chosen for f. However, by convention $f(\alpha = 0) = 1$, $f(\alpha = +1) =$"`hi`" and $f(\alpha = -1) =$"`lo`". In this case, $\theta_i = \alpha$.

- A *normalization factor* or `normfactor` modifier is a multiplicative modifier that scales the entire sample in this region with the value of the parameter μ itself. In this case, $\theta_i = \mu$.
- The `staterror` modifier is a shorthand for encoding uncorrelated statistical uncertainties on the values of the step-functions, using a variant[1] of the Barlow-Beeston Method [4]. Here, the relative uncertainty on the sum of all samples in this region containing the `staterror` modifier is computed bin-by-bin. Then, a constrained *uncorrelated shape systematic* (`shapesys`) is created, encoding these relative uncertainties in the corresponding `Poisson` (or `Gaussian`) constraint term.

type of modifier	description	definition	free parameters
`shapesys`	Uncorrelated Shape systematic	$\kappa(x, \vec{\gamma}) = \chi_b^\gamma \cdot \chi_b^f$	$\gamma_0, ..., \gamma_n$
`histosys`	Correlated Shape systematic	$\delta(x, \alpha) = \alpha * \chi_b^f$	α
`normsys`	Normalization systematic	$\kappa(x, \alpha) = f(\alpha)$	α
`normfactor`	Normalization factor	$\kappa(x, \mu) = \mu$	μ
`shapefactor`, `stat_error`	Shape factor	$\kappa(x, \vec{\gamma}) = \chi_b^\gamma$	$\gamma_0, ..., \gamma_n$

Modifiers can be constrained. In essence, the likelihood picks up a penalty term for changing the corresponding parameter too far away from its nominal value. If a modifier has a `constraint`, which can be of the type `Gauss` for a unit gaussian, `Poisson` for a Poissonian, or `LogNormal`, a corresponding constraint term will be considered in addition to the `aux_likelihood` section of the likelihood, constraining the parameter to its nominal value. The nominal values are $\alpha = 0$, $\gamma = 1$, $\mu = 1$.

name custom string

type histfactory_dist

axes array of structs representing the axes. If given each struct needs to have the component `name`. Further, (optional) components are `max`, `min` and `nbins`.

samples array of structs containing the samples of this channel. For details see below.

Struct of one sample:

name custom string

data struct containing the components `contents` and `errors`, depicting the data contents and their errors. Both components are arrays of the same length.

modifiers array of structs with each struct containing a component `name` and a `type` of modifier. Further (optional) components are `data` and `constraints` both depending on the type of modifier. For details on these components, see the description above.

HistFactory

```
{
    "name": "myAnalysisChannel",
    "type": "histfactory_dist",
```

[1]The variation consists of summarizing all contributions in the stack to a single contribution as far as treatment of the statistical uncertainties is concerned.

```
        "axes": [
            {
                "max": 1.0,
                "min": 0.0,
                "name": "myRegion",
                "nbins": 2
            }
        ],
        "samples": [
            {
                "name": "mySignal",
                "data": {
                    "contents": [ 0.5, 0.7 ],
                    "errors": [ 0.1, 0.1 ]
                },
                "modifiers": [
                    {
                        "name": "Lumi",
                        "type": "normfactor"
                    },
                    {
                        "name": "mu_signal_strength",
                        "type": "normfactor"
                    },
                    {
                        "constraint": "Gauss",
                        "data": { "hi": 1.1, "lo": 0.9 },
                        "name": "my_normalization_systematic_1",
                        "type": "normsys"
                    },
                    {
                        "constraint": "Poisson",
                        "name": "staterror",
                        "type": "staterror"
                    },
                    {
                        "constraint": "Gauss",
                        "data": { "hi": { "contents": [ -2.5, -3.1 ] }, "lo": { "
                            contents": [ 2.2, 3.7 ] } },
                        "name": "my_correlated_shape_systeamtic_1",
                        "type": "histosys"
                     },
                     {
                        "constraint": "Poisson",
                        "data": { "vals": [ 0.0, 1.2 ] },
                        "name": "my_uncorrelated_shape_systematic_2",
                        "type": "shapesys"
                     }
                ]
            },
            {
                "name": "myBackground"
                ...
            }
        ]
    }
```

2.2 Functions

The top-level component `functions` describes an array of mathematical functions in struct format to be used as helper objects. Similar to distributions each entry is required to have the components `type` and `name`. Other components are dependent on the kind of functions. Functions in general have the following components:

name custom string

type string that determines the kind of function, e.g. `sum`

... each function has individual parameter keys for the various individual parameters. For example, functions of type `sum` have the parameter key `summands`. In general, these keys can describe strings as references to other objects or numbers. Depending on the parameter and the type of function, they appear either in single item or array format.

Example: Functions

```
"functions": [
        {
        "name" : "sum1",
        "type" : "sum",
        "summands" : [1.8, 4, "param_xy"]
        },
        ...
]
```

In the following the implemented functions are described in detail.

2.2.1 Product

A product of values or functions a_i.

$$\text{Prod} = \prod_{i}^{n} a_i \tag{15}$$

name custom string

type product

factors array of names of the elements of the product or numbers.

2.2.2 Sum

A sum of values or functions a_i.

$$\text{Sum} = \sum_{i}^{n} a_i \tag{16}$$

name custom string

type sum

summands array of names of the elements of the sum or numbers.

2.2.3 Generic Function

Note: Use of this function type is discouraged in favour of more specific function types.

A generic mathematical function, encoded by a string.

This function type is intended to encapsulate elementary mathematical operations such as addition("+"), subtraction("-"), multiplication("*") and division("/"). For any function-calls, such as `exp`, `pow`, `min`, `max` or similar, the behavior is undefined. Frameworks developers are encouraged to use their frameworks built-in just-in-time-compiler or interpreter to parse this expression without pre-processing and raise an exception when this fails.

`name`	custom string
`type`	generic_function
`expression`	a string with a generic mathematical expression. Simple mathematical syntax common to programming languages should be used here, such as `x-2*y+z`. For any non-elementary operations, the behavior is undefined.

2.3 Data

The component `data` contains an array of data sets in struct format. Each data set needs to contain the components `type` and `name`. Other components are dependent on the type of data set as depicted in the following

`name`	custom string
`type`	string that determines the format of the observations
`...`	each type of observations has different parameter keys. Some of these are optional and marked accordingly in the more detailed description below

A detailed description of the different types with examples can be found below.

2.3.1 Point Data

Point data describes a measurement of a single number, with a possible uncertainty (error).

`name`	custom string
`type`	point
`value`	value of this data point
`error`	(optional) error of this data point

Example: Point Data

```
"data":[
  {
    "name":"data1",
    "type":"point",
    "value":0.,
    "error":1.
  }
]
```

2.3.2 Unbinned Data

Unbinned data describes a measurement of multiple data points in a possibly multi-dimensional space of variables. These data points can be weighted.

name	custom string
type	unbinned
entries	array of arrays containing the coordinates/entries of the data
axes	array of structs representing the axes. If given each struct needs to have the component `name`. Further, (optional) components are `max`, `min` and `nbins`.
weights	(optional) array of values containing the weights of the individual data points, to be used for χ^2 comparisons and fits. If this component is not given, weight 1 is assumed for all data points. If given, the array needs to be of the same length as `entries`.
entries_error	(optional) array of arrays containing the errors/uncertainties of each entry. If given, the array needs to be of the same shape as `entries`.

Example: Unbinned Data

```
"data":[
  {
    "name":"data1",
    "type":"unbinned",
    "weights":[
      9.0,
      18.4
    ],
    "entries":[
      [1],
      [2]
    ],
    "entries_errors":[
      [0.3],
      [0.6]
    ],
    "axes":[
      {
        "name":"variable1",
        "min":1,
        "max":3
      },
      ...
      ]
    },
    ...
  ]
```

2.3.3 Binned Data

Binned data describes a histogram of data points with bin contents in a possibly multi-dimensional space of variables.

name custom string

type binned

contents array of values representing the contents of the binned data set

axes array of structs representing the axes. If given each struct needs to have the component `name`. Further, (optional) components are `max`, `min` and `nbins`.

uncertainty (optional) struct representing the uncertainty of the contents. It consists of up to three components:

- **type** denoting the kind of uncertainty, for now only Gaussian distributed uncertainties denoted as `gaussian_uncertainty` are supported
- **sigma** array of the standard deviation of the entries in `contents`. Needs to be of the same length as `contents`
- **correlation** (optional) array of arrays denoting the correlation between the contents in matrix format. Must be of dimension length of `contents` $\times$ length of `contents`. It can also be set to 0 to indicate no correlation.

Example: Binned Data

```
"data":[
  {
    "name":"data2",
    "type":"binned",
    "contents":[
      9.0,
      18.4
    ],
    "axes":[
      {
        "name":"variable1",
        "nbins":2,
        "min":1,
        "max":3
      },
      ...
    ]
  },
  ...
]
```

This type can also be used to store pre-processed data utilizing the `uncertainty` component

Example: Pre-processed binned Data

```
"data":[
  {
    "name":"data4",
    "type":"binned",
    "contents":[
       9.0,
       18.4
    ],
```

```
    "uncertainty" : {
       "type": "gaussian_uncertainty",
       "correlation" : 0,
       "sigma" : [ 3, 4 ]
    },
    "axes":[
      {
        "name":"variable1",
        "nbins":2,
        "min":1,
        "max":3
      },
      ...
    ]
  },
  ...
]
```

2.4 Likelihoods

The component `likelihoods` contains an array of likelihoods in struct format which are defined as combinations of distributions and observations. Distributions and observations can only be integrated as keys in string format as references to distributions and observations implemented in their respective components. Thus, the components of a likelihood struct are:

`name` custom string

`distributions` array of strings referencing the used distributions

`data` array of strings referencing the used data, must be of the same length as the array of `distributions`

Example: Likelihoods

```
"likelihoods":[
   {
      "name":"likelihood1",
      "distributions":[
         "dist1",
         "dist2",
         ...
      ],
      "data":[
         "data1",
         "data2",
         ...
      ]
   },
   ...
]
```

2.5 Domains

The component `domains` contains an array of domains. These contain information on ranges of parameters and variables in struct format. Each domain must contain a `name` and a `type` although right now only the `product_domain` type is supported. A domain consists of the following components:

`name` custom string

`type` product_domain

`axes` array of parameters and varibales in this domain (see below)

Example: Domains

```
"domains":[
  {
    "name":"domain1",
    "type":"product_domain",
    "axes": [
        {
            "name" : "par_1",
            "max" : 1,
            "min" : 8
        },
        {
            "name" : "par_2",
            "max" : 4.78,
            "min" : 6
        },
        ...
    ]
  },
  ...
]
```

The component `axes` itself is an array of ranges each containing the components `min`, `max` and `name`.

`name` custom string

`max` upper bound of range

`min` lower bound of range

2.6 Parameter points

The component `parameter_points` contains an array of parameter configurations. These can be starting values for minimizations, parameter settings used to generate toy data, best-fit-values obtained, or points in parameter space used for different purposes.

`name` custom string

`parameters` array of parameter structs (see below)

Example: Parameter points

```
    "parameter_points":[
        {
```

```
        "name" : "starting_values",
        "parameters": [
          {
            "name" : "par_1",
            "value": 3
          },
          {
            "name" : "par_2",
            "value": 7,
            "const": true
          },
          ...
        ]
        },
        ...
    ]
```

The component `parameters` is an array of components each containing

`name`	custom string
`value`	number, value of variable
`const`	(optional) boolean, whether variable is constant or not. Default is `false`.

2.7 Analyses

The component `analyses` contains an array of possible (automated) analyses. To that extent, likelihoods, parameters of interest and the affiliated domains are listed. Description of the components:

`name`	custom string
`likelihood`	name as reference to a likelihood defined in the top-level component `likelihoods`
`aux_likelihood_terms`	(optional) array of names of some distributions defined under `distributions` to be used as *auxiliary likelihood terms* (see below)
`parameters_of_interest`	(optional) array of names as reference to parameters that are interesting for the analysis at hand
`parameter_domain`	name of a domain to be used for the parameters, defined in the top-level component `domains`
`data_domain`	name of a domain to be used for the variables in data, defined in the top-level component `domains`
`aux_likelihood`	(optional) name of a constraint term to be used, defined in the top-level component `distributions`
`init_value`	(optional) name of an initial value to be used, defined in the top-level component `parameter_points`

prior (optional) name of a prior distribution, defined in the top-level component `distributions`. This could, for example, be a product distribution of all the individual priors.

All parameters of all distributions in the likelihood must either be listed under the domain referenced, or set to `const` in the parameter point referenced. Interpretation in Bayesian context of the components `parameter_domain`, `aux_likelihood`, `init_value` and `aux_likelihood_terms` imply a prior distribution. Alternatively, a prior distribution may also be directly defined through the component `prior`.

Auxiliary likelihood terms are additional terms to be added to the likelihood. They are intended to reflect subsidiary measurements, typically employing some simple distributions types like Gaussian, Poisson or log-normal. While these subsidiary measurements could easily be stored as full likelihood terms in the corresponding `likelihood`, this is often inconvenient, since these subsidiary measurements often follow a conventional form of being centered around some default value, mostly 0 or 1, which would then need to be stored as a separate dataset under `data`. Thus, by convention, distributions listed under `aux_likelihood_terms` use the following default-values as `data`:

gaussian_dist 0

poisson_dist 1

lognormal_dist 1

Example: Analyses

```
"analyses": [
    {
    "name" : "analysis1",
    "likelihood" : "likelihood1"
    "aux_likelihood_terms" : ["dist1", "dist2"]
    "parameters_of_interest" : ["param1"],
    "parameter_domain" : "domain1" ,
    "data_domain" : "domain1" ,
    "init_value" : "starting_values",
    "prior" : "prior_dist"
    },
    ...
]
```

2.8 Metadata

The top-level component `metadata` contains meta-information related to the creation of the file. The component `hs3_version` stores the HS^3 version and is required for now. Overview of the components:

hs3_version (required) HS^3 version number for reference

packages (optional) array of structs defining packages and their version number used in the creation of this file, depicted with the components `name` and `version` respectively

authors (optional) array of authors, either individual persons, or collaborations

publications (optional) array of document identifiers of publications associated with this file

description (optional) short abstract/description for this file

Example: Metadata

```
"metadata" :
{
    "hs3_version" : "0.2.0",
    "packages" : [
      {
      "name": "ROOT",
      "version": "6.28.02"
      }
    ],
    "authors": ["The ATLAS Collaboration", "The CMS Collaboration"],
    "publications": ["doiABCDEFG"]
}
```

2.9 Miscellaneous

The top-level component `misc` contains arbitrary, user-created information in struct format.

Example: Miscellaneous

```
"misc" :
{
    "custom key 1" : "custom information 1"
}
```

This top-level component is intended to store any and all additional information, including user- or backend-specific meta-information. Examples include, but are not limited to:

- colors and styles for drawing distributions in this file
- suggested settings for samplers or minimizers when working with the distributions in this file
- comments explaining design choices made when building the model in this file
- suggested names and paths for output files to be used by backends working with this file

References

1. Heinrich, L., Feickert, M. & Stark, G. *pyhf* version 0.6.3. `https://github.com/scikit-hep/pyhf/releases/tag/v0.6.3`.
2. An, S. *et al. root-project/root* version 6.26. Mar. 2022.
3. Cranmer, K., Lewis, G., Moneta, L., Shibata, A. & Verkerke, W. *HistFactory: A tool for creating statistical models for use with RooFit and RooStats* tech. rep. (New York U., New York, Jan. 2012). `https://cds.cern.ch/record/1456844`.
4. Barlow, R. & Beeston, C. Fitting using finite Monte Carlo samples. *Computer Physics Communications* **77**, 219–228. ISSN: 0010-4655. `https://www.sciencedirect.com/science/article/pii/001046559390005W` (1993).