HS^3 - Overview of supported types and components

06. March 2023

1 Introduction

With the introduction of `pyhf` [3], 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 [1] 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, 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 draft.

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`	strings are represented with “ ”, they can be used to refer to objects or contain names and arbitrary information
`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 key `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) dictionary 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 keys for the various individual parameters. For example, distributions of type `gaussian_dist` have the parameter keys `mean` and `sigma`. In general, these keys can describe strings as references to other objects or values. Depending on the parameter and the type of distribution, they appear either in single item or array format.

All other components are specific to the type of distribution and can contain numbers, strings as reference to other objects or Boolean values.

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

Exponential distributions. The PDF 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

Gaussian/Normal distributions The PDF of a Gaussian 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 variable encoding the root-mean-square width σ.

2.1.3 Addition of distributions

This PDF encodes a sum of PDFs f_i:

$$\text{SumPdf}(x) = \sum_{i=1}^{n-1} c_i * f_i(\vec{x}) \tag{3}$$

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

$$c_n = 1 - \sum_{i=1}^{n-1} c_i \tag{4}$$

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

2.1.4 Product Distribution

A product of independent distributions f_i is defined as

$$\text{ProductPdf}(x) = \prod_{i}^{n} f(x) \tag{5}$$

`name`	custom string
`type`	product_dist
`factors`	array of distributions or strings referencing distributions

2.1.5 Continuous Uniform Distribution

The PDF of a continuous uniform distribution is defined as:

$$\text{UniformPdf}(x, a, b) = \frac{1}{b-a} \quad a \leq x \leq b \tag{6}$$

`name`	custom string
`type`	uniform_dist
`max`	upper bound of range b
`min`	lower bound of range a

2.1.6 Polynomial Distribution

The PDF of a polynomial distribution is defined as

$$\text{PolynomialPdf}(x, a_0, a_1, a_2, ...) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + ... \tag{7}$$

`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.

2.1.7 HistFactory Distribution

HistFactory [2] is a language to describe statistical models of ”histograms” (step-functions). Each HistFactory distribution describes one ”channel” or ”region” of a binned measurement, containing a stack of binned signal- and background-contributions sharing the same binning. Each of the contributions may be subject to *modifiers*.

$$\text{HistFactoryPdf}(x, \vec{\theta}) = \sum_{s \in \text{samples}} (d_s(x) + \sum_{\delta \in \text{modifiers}} \delta(x, \theta_\delta) \prod_{\kappa \in \text{modifiers}} \kappa(x, \theta_\kappa) \tag{8}$$

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

$$d_s(x) = \chi_b^{y_s}(x) \tag{9}$$

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

All samples and modifiers share the same binning b.

Each of the modifiers is either multiplicative (κ) or additive (δ). They depend on a set of nuisance parameters θ. 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.

key	description	definition	free parameters
`shapesys`	Uncorrelated Shape systematic	$\kappa(x, \vec{\gamma}) = \chi_b^\gamma \cdot \chi_b^f$	γ_0, ..., γ_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

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 depended on the kind of functions. Functions in general have the following keys:

`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 list 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.

$$\text{Prod} = \prod_i^n a_i \tag{10}$$

`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.

$$\text{Sum} = \sum_i^n a_i \tag{11}$$

`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 axes with keys `name` and (optional) `max` and `min`. Further (optional) keys include `nbins` and `value` for consistency with other notations of variables and parameters.
`weights`	(optional) 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 values containing the errors/uncertainties of each entry. If given, the array needs to be of the same length 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 counts of the binned data set
`axes`	(optional) array of structs with keys `name`, `max`, `min` and `nbins`. Further, custom keys are also possible.
`uncertainty`	(optional) struct representing the uncertainty of the contents. It can contain up to three components:

`type`	denoting the kind of uncertainty, for now only `gaussian_uncertainty` is 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. Description of the keys:

`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 in struct format. Each domain must contain a `name` and a `type` although right now only the `product_domain` type is supported. Description of the components:

name	custom string
type	product_domain
axes	array of parameters 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 parameters

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 containing the keys

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 keys:

name	custom string
likelihood	name as reference to a likelihood defined in the top-level component `likelihoods`
aux_likelihood_terms	names of some distributions defined under `distributions` to be used aus *auxiliary likelihood terms* (see below)
parameters_of_interest	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	name of a constraint term to be used, defined in the top-level component `distributions`
init_value	name of an initial value to be used, defined in the top-level component `parameter_points`
prior	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 `domain`, `constraint` and `init_value` imply a prior distribution.

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 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 `version` stores the HS3 version and is required for now. Overview of possible components:

`version`	(required) HS3 version number for reference
`packages`	(optional) array of structs defining packages used in the creation of this file
`authors`	(optional) array of authors, either individual persons, or collaborations
`publications`	(optional) document identifiers of publications associated with this file
`description`	(optional) short abstract/description for this file

Example: Metadata

```
    "metadata" :
```

```
    {
        "hs3_version" : 0.2,
        "packages" : [
          {
          "name": "ROOT",
          "version": 6.28
          }
        ],
        authors : ["The ATLAS Collaboration", "The CMS Collaboration"],
        publications: ["doiABCDEFG"]
    }
```

2.9 Miscellaneous

The top-level component `metadata` 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] Sitong An, Simone Azeglio, Rahul Balasubramanian, Bertrand Bellenot, Josh Bendavid, Jakob Blomer, Patrick Bos, Rene Brun, Carsten D. Burgard, Will Buttinger, Philippe Canal, Olivier Couet, Mattias Ellert, Gerri Ganis, Andrei Gheata, Enrico Guiraud, Stephan Hageboeck, Jonas Hahnfeld, Ahmat Hamdan, Fernando Hueso-González, Ivan Kabadzhov, Shamrock Lee, Sergey Linev, Javier Lopez-Gomez, Pere Mato, Emmanouil Michalainas, Lorenzo Moneta, Nicolas Morange, Axel Naumann, Vincenzo Eduardo Padulano, Max Orok, Alexander Penev, Danilo Piparo, Fons Rademakers, Jonas Rembser, Enric Tejedor Saavedra, Aaradhya Saxena, Oksana Shadura, Sanjiban Sengupta, Federico Sossai, Harshal Shende, Matevz Tadel, Vassil Vassilev, Wouter Verkerke, Zef Wolffs, and Stefan Wunsch and. root-project/root: v6.26, March 2022.

[2] Kyle Cranmer, George Lewis, Lorenzo Moneta, Akira Shibata, and Wouter Verkerke. HistFactory: A tool for creating statistical models for use with RooFit and RooStats. Technical report, New York U., New York, Jan 2012.

[3] Lukas Heinrich, Matthew Feickert, and Giordon Stark. pyhf: v0.6.3.