Dilan Pathirana, Fabian Fröhlich, Sebastian Persson,
Frank T. Bergmann, Matthias König, Paul F. Lang, Severin Bang,
Svenja Kemmer, Jan Hasenauer, and Daniel Weindl*

The Parameter Estimation tables (PEtab) format version 2

https://doi.org/..., Received ...; accepted ...

Abstract:
Parameter estimation is a central task in data-driven computational modeling, and a clear, unambiguous, and commonly understood specification of parameter estimation problems is essential for reproducibility and interoperability. PEtab provides a standardized framework for defining such problems in systems biology and systems medicine, particularly for ordinary differential equation models. This document describes the second version of the PEtab format, PEtab 2.0.
PEtab 2.0 simplifies and clarifies how estimation problems are defined, while also extending the standard to support more complex and realistic experimental setups.

Dilan Pathirana, Bonn Center for Mathematical Life Sciences, University of Bonn, Germany; Life and Medical Sciences (LIMES) Institute, University of Bonn, Germany. https://orcid.org/0000-0001-7000-2659
Fabian Fröhlich, Dynamics of Living Systems Laboratory, The Francis Crick Institute, 1 Midland Rd, London NW1 1AT, United Kingdom. https://orcid.org/0000-0002-5360-4292
Sebastian Persson, Dynamics of Living Systems Laboratory, The Francis Crick Institute, 1 Midland Rd, London NW1 1AT, United Kingdom. https://orcid.org/0009-0001-2304-4263
Frank T. Bergmann, BioQUANT - Heidelberg University, INF 267, 69120 Heidelberg, Germany. https://orcid.org/0000-0001-5553-4702
Matthias König, Humboldt-Universität zu Berlin, Faculty of Life Sciences, Department of Biology, Systems Medicine of the Liver, Unter den Linden 6, 10099 Berlin, Germany; University of Stuttgart, Institute of Structural Mechanics and Dynamics in Aerospace Engineering, Pfaffenwaldring 27, 70569 Stuttgart, Germany. https://orcid.org/0000-0003-1725-179X
Paul F. Lang, Deep Origin, South San Francisco, California, USA; JuliaHub, Cambridge, Massachusetts, USA. https://orcid.org/0000-0002-6388-2405
Severin Bang, Institute of Physics, University of Freiburg, Freiburg im Breisgau, Germany.
Svenja Kemmer, Institute of Physics, University of Freiburg, Freiburg im Breisgau, Germany. https://orcid.org/0000-0001-7708-0545
Jan Hasenauer, Bonn Center for Mathematical Life Sciences, University of Bonn, Germany; Life and Medical Sciences (LIMES) Institute, University of Bonn, Germany. https://orcid.org/0000-0002-4935-3312
***Corresponding author: Daniel Weindl,** Bonn Center for Mathematical Life Sciences, University of Bonn, Germany; Life and Medical Sciences (LIMES) Institute, University of Bonn, Germany. E-mail: daniel.weindl@uni-bonn.de. https://orcid.org/0000-0001-9963-6057

Key improvements include support for additional model formats beyond SBML, expanded and improved specification of parameter priors for Bayesian parameter estimation, and a more flexible description of experiments that allows multiple conditions, time periods, and pre-equilibration steps within a single simulation. Additional updates refine the definition of model parameters, measurement noise, and observation models, and remove ambiguous or confusing features from earlier versions. Version 2 also includes a general extension mechanism, to add functionality for specialized applications not covered by the core standard.

Keywords: parameter estimation; PEtab; community standard; systems biology; dynamic modeling; Bayesian inference; likelihood-based inference

Contents

1 About

This document describes the second version of the PEtab format. The authoritative PEtab 2.0 specification is available at https://petab.readthedocs.io/. This website will provide any future updates, an erratum, as well as additional resources around PEtab. PEtab version 1.0 is described in [7].

2 Scope of PEtab

The scope of PEtab is the complete specification of parameter estimation problems in typical systems biology applications. In practice, data-driven modeling often begins with either (i) a computational model of a biological system that requires calibration or (ii) experimental data that need integration and analysis through a computational model [8].

Measurements are linked to the biological model by an observation and noise model. Often, measurements are taken after some experimental perturbations have been applied, which are represented as variants of a generic model (Figure 1A). Therefore, a goal of PEtab is to specify such a setup in the least redundant way. Furthermore, PEtab aims to provide an intuitive, modular, machine- and human-readable and -writable format that makes use of existing standards.

3 Overview

PEtab builds on existing standards for model specification and defines a parameter estimation problem using multiple text-based files in YAML [1] and Tab-Separated Values (TSV [6]) format (Figure 2). A PEtab problem consists of the following types of files:

- A `problem configuration file` that lists all of the following files and provides additional information including `extensions` [YAML].
- `Parameter file(s)` to set parameter values globally (across all experiments), and to specify the parameters to be estimated as well as their parameter bounds and prior distributions [TSV].
- `Model` file(s) specifying the base model(s) [SBML [4, 5], CellML [2], BNGL [3], ...].
- `Observable file(s)` defining the observation model [TSV].
- `Measurement file(s)` containing experimental data used for model calibration [TSV].
- (optional) `Condition file(s)` specifying model inputs and condition-specific parameters [TSV].
- (optional) `Experiment file(s)` describing sequences of experimental conditions applied to the model [TSV].
- (optional) `Mapping file(s)` assigning PEtab-compatible IDs to model entities that do not have valid PEtab IDs themselves, and providing additional annotations [TSV].

Figure 1B shows how those files relate to a typical data-based modeling setup.

The following sections outline the minimum requirements of these components in the core standard, which should provide all necessary information for defining a parameter estimation problem.

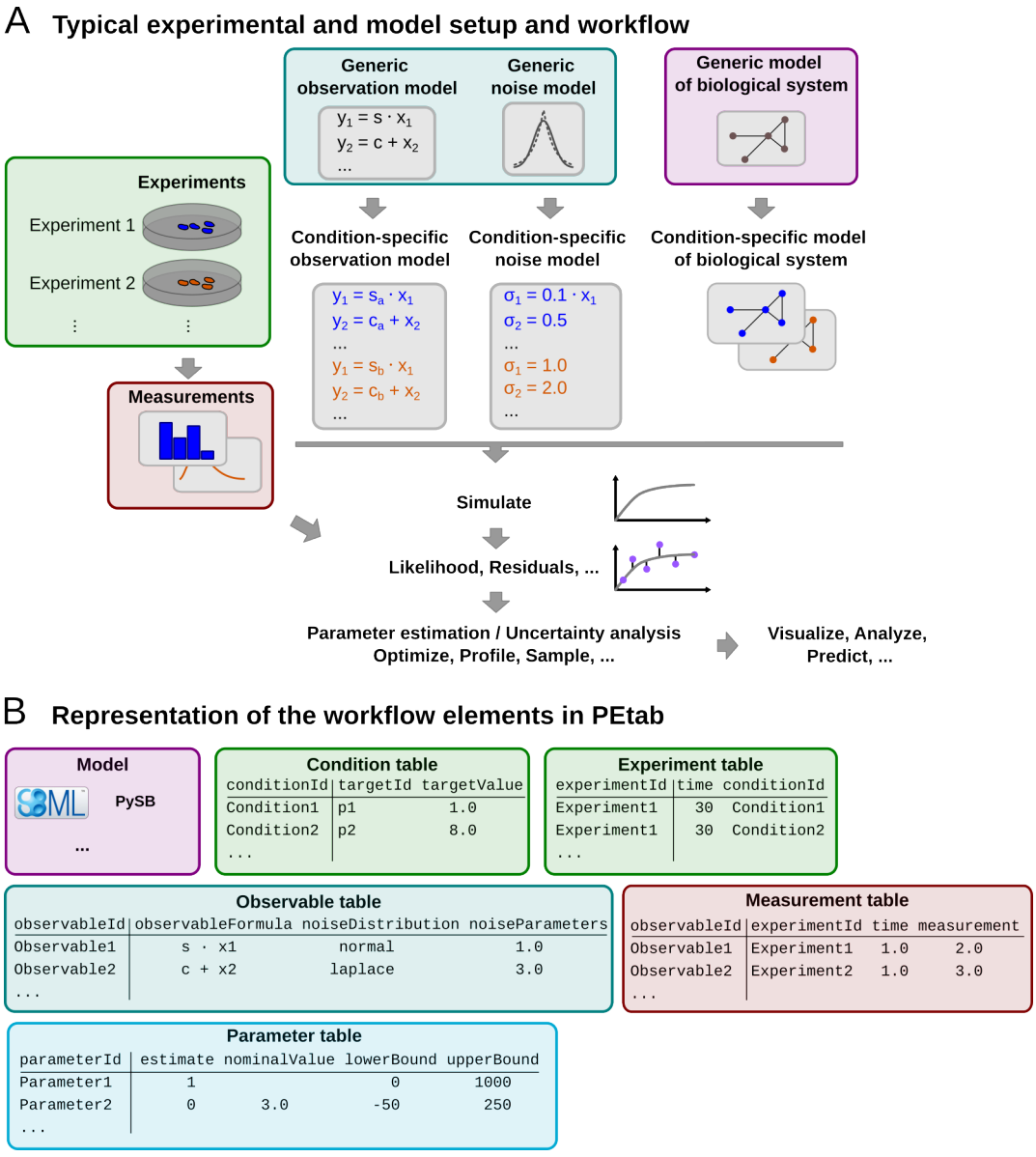

Figure 1: Structure of typical parameter estimation problems in systems biology and their representation in PEtab. A: Different experiments are conducted and measurements are taken. The different experiments are described by different instances of a generic model. These experiment-specific models are simulated to evaluate an objective function. **B:** How the different elements of A are represented in PEtab. Corresponding elements are indicated by the same background color. Modified from [7].

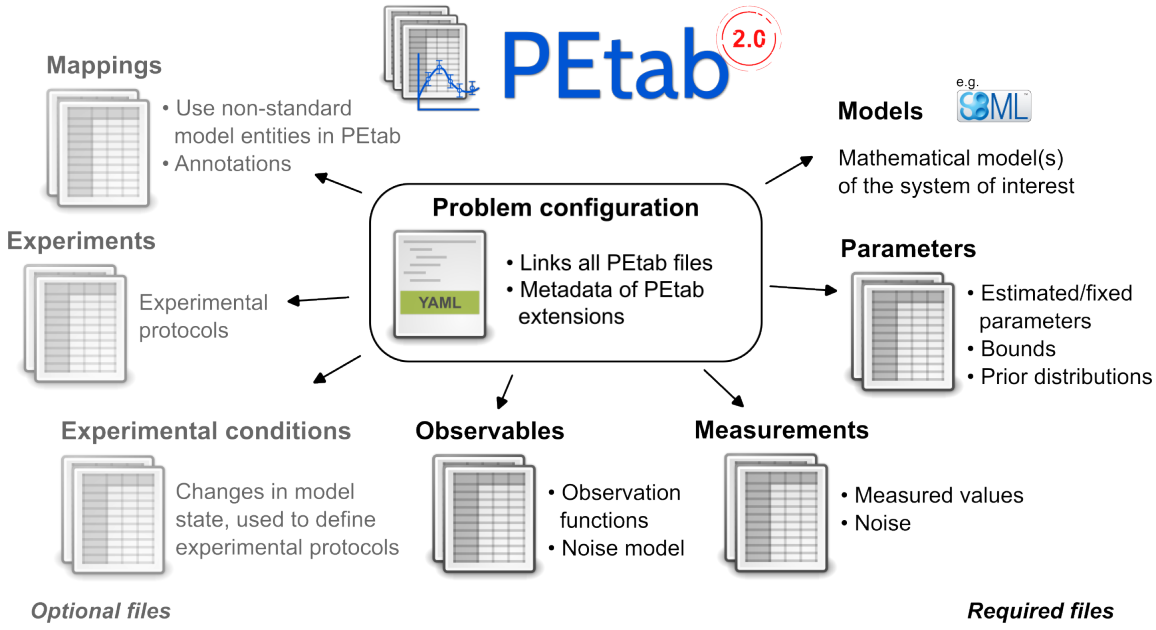

Figure 2: Files constituting a PEtab 2.0 problem. A single YAML file links the different files types. There can be one or more files of each type; the grayed-out files are optional.

Additional non-standard table columns are allowed in all PEtab tables. While such extra columns may, for example, contain metadata to enhance plotting, downstream analysis, or improve efficiency in parameter estimation, they must not alter the definition of the estimation problem itself, unless they are part of a PEtab `extensions`.

General remarks

- All model entities, column names and row names are case-sensitive.
- Fields enclosed in "[]" are optional and may be left empty.
- The following data types are used in descriptions below:
  - `STRING`: Any string.
  - `NUMERIC`: Any number excluding `NaN` / `inf` / `-inf`
  - `MATH_EXPRESSION`: A mathematical expression according to the PEtab math expression syntax (section 15).
  - `PETAB_ID`: A string that is a valid PEtab ID (section 16).
  - `NON_PARAMETER_TABLE_ID`: A valid PEtab ID referring to a constant or differential entity, including PEtab output parameters, but excluding parameters listed in the `parameter table(s)` (independent of their `estimate` value).
  - `LIST[...]`: A `STRING` that is a semicolon-delimited list of values, where each value can be interpreted as the type or value inside the brackets.
  - `OPTIONAL` indicates that a column is optional whereas `NULL` indicates that individual cells in a column may be empty.

4 Changes from PEtab 1.0

PEtab 2.0 is a major update of the PEtab format. The main changes are:

- The PEtab problem configuration file is now mandatory and its structure changed. In particular, the `problems` list has been flattened.
- Support for `models` in other formats than SBML.
- The use of different models for different measurements is now supported via the optional `modelId` column in the `measurement file(s)` (see also section 12.1). This was poorly defined in PEtab 1.0.
- The (now optional) condition table format changed from wide to long (section 6).
- `simulationConditionId` and `preequilibrationConditionId` in the `measurement file(s)` are replaced by `experimentId` and a more flexible way for defining experiments and time courses. This allows arbitrary sequences of conditions and combinations of conditions to be applied to the model (Figure 3 and section 7).

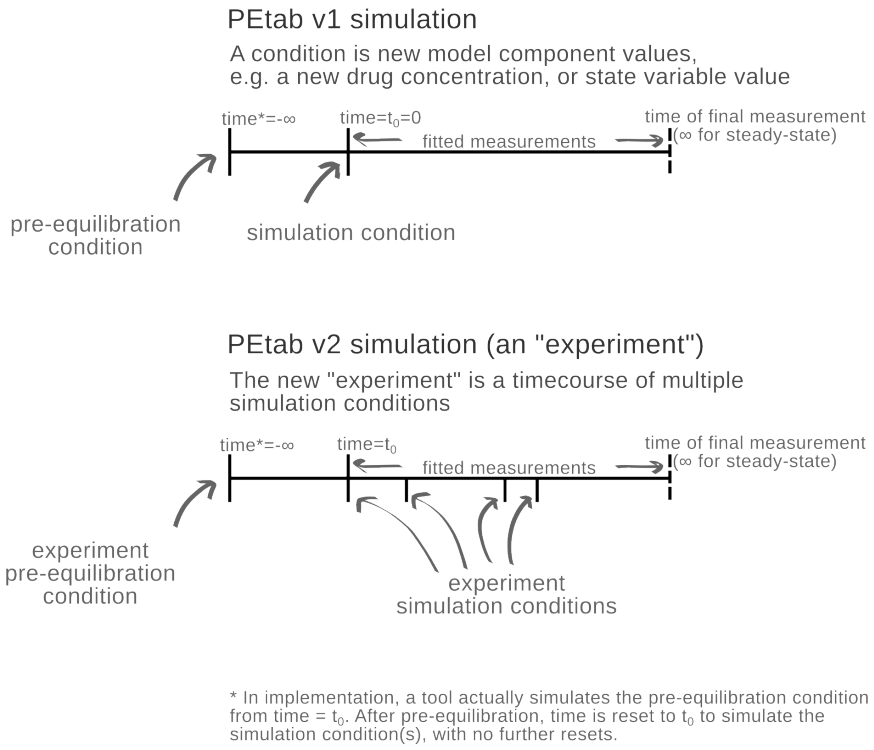

Figure 3: A comparison of simulations in PEtab 1.0 and 2.0. While in PEtab 1.0, a simulation consisted of one or two periods, PEtab 2.0 supports an arbitrary number of periods. Furthermore, PEtab 2.0 allows specifying the initial time of the simulation.

- Support for math expressions in the condition table (section 6, section 15).
- Clarification and specification of various previously underspecified aspects, including overriding values via the condition table (section 13).

- Support for format extensions (section 17).
- Observable IDs can now be used in observable and noise formulas (section 9).
- The `parameterScale` column of the `parameter file(s)` is removed. This change was made to simplify the PEtab format. This feature was a constant source of confusion and the interaction with parameter priors was not well-defined. To obtain the same effect, the model parameters can be transformed in the model file.
- The `initializationPriorType` and `initializationPriorParameters` columns of the `parameter file(s)` are removed. Initialization priors are outside the definition of the parameter estimation problem and were a source of confusion.
- `objectivePriorType` and `objectivePriorParameters` in the `parameter file(s)` are renamed to `priorDistribution` and `priorParameters`, respectively. This change was made to simplify the PEtab format.
- The admissible values for `estimate` in the `parameter file(s)` are now `true` and `false` instead of `1` and `0`.
- Support for new parameter prior distributions in the `parameter file(s)`, and clarification that bounds truncate the prior distributions.
- The `observableTransformation` column of the `observable file(s)` has been combined with the `noiseDistribution` column to make its intent clearer. The `log10` transformation has been removed, since this was mostly relevant for visualization purposes, and the same effect can be achieved by rescaling the parameters of the respective (natural) log-distributions.
- The `observableFormula` field in the `observable file(s)` must not contain any observable IDs. This was previously allowed, but it was not well-defined how to deal with placeholder parameters in this case. The `noiseFormula` field may contain only the observable ID of the respective observable itself, but no other observable IDs for the same reason.
- Placeholders for measurement-specific parameters in `observableFormula` and `noiseFormula` are now declared using the `observablePlaceholders` and `noisePlaceholders` fields in the `observable tables`.
  This replaces the previous `observableParameter${n}_${observableId}` syntax. The new approach is more explicit and allows for more descriptive and shorter names for the placeholders.
- The visualization table has been removed. The PEtab 1.0 visualization table was not well-defined and not widely used.

5 Model definition

PEtab 2.0 is **model-format–agnostic**, meaning it does not depend on a specific model description. The model file is referenced in the problem configuration file by its file name or a URL.

PEtab distinguishes between three types of model entities:

- **Differential entities**: Entities whose time evolution is defined in terms of a time-derivative, e.g., the targets of SBML rate rules or species that change due to participation in reactions (reactants or products).
- **Algebraic entities**: Entities that are defined in terms of algebraic assignments, rather than time derivatives, that are in effect throughout the simulation. They are not necessarily constant, for example, the targets of SBML assignment rules.
- **Constant entities**: Entities that are not differential or algebraic entities. They are defined in terms of an at least piecewise constant value but may be subject to event assignments,

e.g., parameters of an SBML model that are not targets of rate rules or assignment rules or determined by algebraic rules.

6 Condition table

The optional condition table defines discrete changes to the simulated model(s). These (sets of) changes typically represent interventions, perturbations, or changes in the environment of the system of interest. These modifications are referred to as (experimental) conditions.

Conditions are applied at specific time points defined in the `experiment table(s)`. This allows for the specification of time courses or experiments spanning multiple time periods. A time period is the interval between two consecutive time points in the experiment table (including the first, excluding the second) for a given experiment, or the time between the last time point of an experiment and the end of the simulation (usually, the time point of the last measurement for that experiment in the `measurement table(s)`).

The `condition table` only allows changes in the model state, not the model structure. That means that only constant or differential entities (e.g., constant parameters or model states) can be modified. **Algebraic entities** (e.g., the targets of SBML assignment rules) cannot be changed.

The condition table is provided as a tab-separated values (TSV) file with the following structure:

conditionId	**targetId**	**targetValue**
PETAB_ID	NON_PARAMETER_TABLE_ID	MATH_EXPRESSION
e.g.		
conditionId1	modelEntityId1	0.42
conditionId1	modelEntityId2	0.42
conditionId2	modelEntityId1	modelEntityId1 + 3
conditionId2	someSpecies	8
...	...	...

Each row in the condition table represents a single modification to a target entity. The order of rows and columns is arbitrary, but placing `conditionId` first can enhance readability.

6.1 Detailed field description

- `conditionId` [PETAB_ID, REQUIRED]

  A unique identifier for the condition associated with the change. This ID is referenced in the `experiment table(s)`.

- `targetId` [NON_PARAMETER_TABLE_ID, REQUIRED]

  The ID of the entity being modified. The target must be either a constant or differential entity and must not be listed in the `parameter table(s)`.

- `targetValue` [MATH_EXPRESSION, REQUIRED]

The value or mathematical expression used to modify the target. When the corresponding condition is applied, the entity specified in `targetId` is set to the value defined in `targetValue` (see section 6.2 for details).

If the model has a concept of species and a species ID is provided, its value is interpreted as either amount or concentration consistent with its usage elsewhere in the model.

6.2 Detailed semantics

Section 13 describes how the changes for the initial period of an experiment are applied and how the model is initialized. For any subsequent time period, the changes defined in the condition table are applied in five consecutive phases:

1. **Evaluation of `targetValue`s**

   `targetValue`s are first evaluated using the *current* values of all variables in the respective expressions. The *current* values are taken from the simulation results at the end of the preceding time period. A special case is simulation time (`time`), which is set to the start time of the current time period.

2. **Assignment of the evaluated `targetValue`s to their targets**

   All evaluated `targetValues` are simultaneously assigned to their respective targets. It is invalid to apply multiple assignments to the same target at the same time.

   The interpretation of the assigned value depends on the type of the model and the target entity. For example, for an SBML model and a `targetId` referring to a species, the `targetValue` will override the current amount of the species if `hasOnlySubstanceUnits=true` (*amount-based species*), or the current concentration if `hasOnlySubstanceUnits=false` (*concentration-based species*).

   If the target is a compartment, the compartment size is set to the evaluated value of the `targetValue` expression. The value of a species in such a compartment is not changed by this assignment. I.e., for an amount-based species, the amount of the species will be preserved (the concentration may change accordingly), and for a concentration-based species, the concentration will be preserved (the amount may change accordingly). Note that this differs from the behavior of an SBML event assignment where compartment size changes always preserve the amounts of species in that compartment.

3. **Update of derived variables**

   After target values have been assigned, all derived variables (i.e., algebraic entities such as SBML assignment rules or PySB expressions) are updated.

4. **Events and finalization**

   At this stage, all remaining state- or time-dependent changes encoded in the model are applied, including SBML events. The model state used for the evaluation of trigger functions or the execution of any event assignments (including events with delays) is the model state after (3).

   The initial model state for the new period at *time* = `time` (experiment table) is the model state after all these updates have been applied.

5. **Evaluation of observables**

   If measurements exist for the current timepoint, the observables are evaluated after all changes have been applied. The resulting values are then compared against the corresponding measurements in the measurement table.

7 Experiment table

The optional experiment table defines a sequence (Figure 3, lower) of experimental conditions (i.e., discrete changes; see section 6) applied to the model.

The experiment table is provided as a tab-separated values (TSV) file with the following structure:

experimentId	**time**	**conditionId**
PETAB_ID	NUMERIC or '-inf'	CONDITION_ID
e.g.		
timecourse_1	0	condition_1
timecourse_1	10	condition_2
timecourse_1	250	condition_3
patient_3	-inf	condition_1
patient_3	0	condition_2
intervention_effect	-20	no_lockdown
intervention_effect	20	mild_lockdown
intervention_effect	40	severe_lockdown

The order of the rows is arbitrary, but specifying the rows in ascending order of time may improve human readability.

7.1 Detailed field description

The experiment table has three mandatory columns `experimentId`, `time`, and `conditionId`:

- `experimentId` [PETAB_ID, REQUIRED]

  A unique identifier for the experiment, referenced by the `experimentId` column in the `measurement table(s)`.

- `time`: [NUMERIC or `-inf`, REQUIRED]

  The time at which the specified condition will become active, in the time unit specified in the model. `-inf` indicates that the respective condition will be simulated until a steady state is reached.

  Note: In PEtab, a steady state is defined as a state in which *all* differential entities have reached, and will remain at, a constant value. Determining whether differential entities are at steady state is left to the simulator and the user. Reasonable numerical criteria should be used, and users are encouraged to share their chosen criteria when sharing their model to ensure reproducibility. Determining steady state can be nontrivial; for example, events or other discontinuities may occur after an apparent steady state has been reached. It is the user's responsibility to avoid situations where this ambiguity is problematic.

If the simulation of an experiment requiring steady state fails to reach a steady state, the evaluation of the model at measurement points is not well-defined. In such cases, PEtab interpreters should notify the user, for example by returning `NaN` or `inf` values for the objective function. PEtab does not prescribe a numerical criterion for steady state. Any event triggers defined in the model must also be checked during this pre-simulation.

Multiple conditions may be applied simultaneously by specifying the same `experimentId` and `time` for multiple conditions. The changes associated with these conditions must be disjoint, i.e., no two conditions applied at the same time may involve the same `targetId`. The union of these changes is applied to the model as if they were specified as a single condition.

`time` will override any initial time specified in the model, except in the case of `time` = `-inf`, in which case the model-specified time will be used (or 0, if the model does not explicitly specify an initial time).

- `conditionId`: [PETAB_ID or empty, REQUIRED, REFERENCES(condition.conditionId)]

  A reference to a condition ID in the `condition table(s)` that will be applied at the given `time`. If empty, no changes are applied. For further details, see section 6.2.

8 Measurement table

A tab-separated values file containing all measurements to be used for model training or validation.

Expected to have the following named columns in any (but preferably this) order:

observableId	**experimentId**	**measurement**	**time**
PETAB_ID	PETAB_ID	NUMERIC	NUMERIC|inf
e.g.			
observable1	experiment1	0.42	0
...	...	...	...

(wrapped for readability)

...	**[observableParameters]**	**[noiseParameters]**
...	LIST[parameterId|NUMERIC]	LIST[parameterId|NUMERIC]
...		
...		
...	...	...

Additional (non-standard) columns may be added.

8.1 Detailed field description

- `observableId` [PETAB_ID, REQUIRED, REFERENCES(observable.observableId)]

  Observable ID as defined in the observable table described below.

- `experimentId` [PETAB_ID or empty, REQUIRED, REFERENCES(experiment.experimentId)]

  Experiment ID as defined in the experiment table described below. This column may have empty cells, which are interpreted as *use the model as is*. This avoids the need for "dummy" conditions and experiments if only a single condition is required. If no experiment is specified, the model simulation is assumed to start at time 0, unless the model specifies a different initial time.

- `measurement` [NUMERIC, REQUIRED]

  The measured value in the same units/scale as the model output.

- `time` [NUMERIC OR `inf`, REQUIRED]

  Time point of the measurement in the time unit specified in the employed model, a finite numeric value, or `inf` (lower-case) for steady-state measurements (the same definition of steady state as in the `experiment table(s)` applies here, with the additional sanity check that the steady state occurs during the final experiment period). This value must be greater than or equal to the first time point of the experiment referenced in the `experimentId` column that is not `-inf`.

  If this time point coincides with the time point of a condition change, the condition change is applied before the observable is evaluated (see section 6.2 for details).

- `observableParameters` [LIST[parameterId|NUMERIC]|NULL, OPTIONAL]

  Measurement-specific overrides for placeholder parameters in the `observableFormula` declared in the `observable table(s)`.

  The `observable table(s)` allows marking some parameters as measurement-specific (see `observablePlaceholders` there). Their values for a given measurement are specified in this column. The values are separated by semicolons. The order and number of values must match the order and number of placeholders in the `observablePlaceholders` field of the corresponding observable in the observable table. I.e., if the observable table contains no observable placeholders for the given observable, this field must be empty. The values may be either numeric values or the IDs of parameters from the `parameter table(s)`.

  Different measurements for the same `observableId` may specify different values. This may be used to account for condition-specific or batch-specific parameters.

  If none of the observables referenced in a given measurement table use any noise placeholders, this column may be omitted there.

- `noiseParameters` [LIST[parameterId|NUMERIC]|NULL, OPTIONAL]

  Measurement-specific overrides for placeholder parameters in the `noiseFormula` declared in the `observable table(s)`.

  The `observable table(s)` allows marking some parameters as measurement-specific (see `noisePlaceholders` there). Their values for a given measurement are specified in this column. The values are separated by semicolons. The order and number of values must match the order and number of placeholders in the `noisePlaceholders` field of the corresponding observable in the observable table. I.e., if the observable table contains no noise placeholders for the given

observable, this field must be empty. The values may be either numeric values or the IDs of parameters from the `parameter table(s)`.

Different measurements for the same `observableId` may specify different values. This may be used to account for condition-specific or batch-specific parameters.

If none of the observables referenced in a given measurement table use any noise placeholders, this column may be omitted there.

- `modelId` [PETAB_ID, OPTIONAL, REFERENCES(yaml.models.model_id)]

  Which model to simulate for each data point. Model IDs are defined by the keys of the `models` object in the PEtab problem YAML file. This column is required when multiple models are defined in the PEtab problem (see section 12.1). For problems with a single model, this column is optional, and its values default to the ID of the only model present.

8.2 Simulation table

For some applications, it is useful to provide a simulation file that contains the simulation results of the model for the same time points as the measurements. This is useful, for example, for comparing the simulation results with the measurements, or for assessing simulation reproducibility across tools. The simulation table is provided as a tab-separated values (TSV) file with the same structure as the measurement table, only that the `measurement` column is replaced by the `simulation` column that contains the simulation results.

9 Observable table

Parameter estimation requires linking experimental observations to the model of interest. Therefore, one needs to define observables (model outputs) and respective noise models, which represent the measurement process. Since parameter estimation is beyond the scope of SBML, there exists no standard way to specify observables (model outputs) and respective noise models. Therefore, in PEtab observables are specified in a separate table as described in the following. This allows for a clear separation of the observation model and the underlying dynamic model, which allows, in most cases, to reuse any existing SBML model without modifications.

The observable table has the following columns:

observableId	**[observableName]**	**observableFormula**
STRING e.g. relativeTotalProtein1	[STRING] Relative abundance of Protein1	STRING scale_relTotProt1 * (protein1 + phospho_protein1)
...	...	...

(wrapped for readability, continued on next page)

...	**noiseFormula**	**[noiseDistribution]**	**[observablePlaceholders]**	**[noisePlaceholders]**
...	STRING|NUMBER	*see below*	*see below*	*see below*
...				
...	sd_relTotProt1	normal	scale_relTotProt1	sd_relTotProt1
...	...	...	...	...

9.1 Detailed field description

- `observableId` [STRING]

  Unique identifier for the given observable. Must consist only of upper and lower case letters, digits and underscores, and must not start with a digit. This is referenced by the `observableId` column in the measurement table.

- `observableName` [STRING, OPTIONAL]

  Name of the observable. Only used for output, not for identification.

- `observableFormula` [STRING]

  Observation function as plain text formula expression. The expression may contain any symbol defined in a model, the mapping table or the parameter table. Often, this is just the ID of a state variable. Furthermore, any parameters introduced through the `observablePlaceholders` field for the given observable may be used (see below).

- `observablePlaceholders` [LIST[PETAB_ID], OPTIONAL]

  A semicolon-separated list of valid PEtab identifiers that have not been introduced elsewhere, marking them as placeholders for measurement-specific parameters. The actual values for these parameters are specified in the `observableParameters` field of the measurement table. The ordering and number of values in `observableParameters` must match the ordering and number of placeholders in `observablePlaceholders`. For an example, see the description of `noisePlaceholders` below.

- `noiseFormula` [NUMERIC|STRING]

  The scale parameter of the noise distribution for the given observable.

  Measurement noise can be specified as a numerical value which will default to a Gaussian noise model if not specified differently in `noiseDistribution` with standard deviation as provided here. In this case, the same standard deviation is assumed for all measurements for the given observable.

  Alternatively, some formula expression can be provided to specify more complex noise models. The same rules as for `observableFormula` apply here. In addition, the current observable ID may be used to refer to the `observableFormula` expression. In addition to the placeholders declared in `observablePlaceholders`, placeholders for noise parameters that are declared in `noisePlaceholders` may be used.

- `noisePlaceholders` [LIST[PETAB_ID], OPTIONAL]

  A semicolon-delimited list of valid PEtab identifiers that have not been introduced elsewhere, marking them as placeholders for measurement-specific noise parameters. The actual values

for these parameters are specified in the `noiseParameters` field of the measurement table. The ordering and number of values in `noiseParameters` must match the ordering and number of placeholders in `noisePlaceholders`.

For example, a noise model that accounts for measurement-specific relative and absolute contributions for some observable with the ID `observable_pErk` could be defined as:

```
noise_offset_pErk + noise_scale_pErk * observable_pErk
```

with `noise_offset_pErk` denoting the absolute and `noise_scale_pErk` the relative contribution. The corresponding `noisePlaceholders` would be:

```
noise_offset_pErk;noise_scale_pErk
```

and the `noiseParameters` in the measurement table could be, for example, `0.1;0.2` resulting in:

```
noise_offset_pErk = 0.1
noise_scale_pErk = 0.2
```

- `noiseDistribution` [STRING, OPTIONAL]

  Assumed noise distribution for the measurements of the given observable. The supported noise distributions and the respective interpretation of `noiseFormula` are given in the table below. Defaults to '`normal`'.

9.2 Noise distributions

Let m denote the measured value, $y :=$ observableFormula the simulated value (the median of the noise distribution), and $\sigma :=$ noiseFormula the noise parameter (the standard deviation and the scale parameter for the Normal and Laplace distributions, respectively). Then we have the following effective noise distributions:

Type	**noiseDistribution**	**Probability density function (PDF)**
Gaussian distribution	`normal`	$\pi(m\|y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(m-y)^2}{2\sigma^2}\right)$
Log-normal distribution (i.e., $\log(m)$ is normally distributed)	`log-normal`	$\pi(m\|y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma m} \exp\left(-\frac{(\log m - \log y)^2}{2\sigma^2}\right)$
Laplace distribution	`laplace`	$\pi(m\|y,\sigma) = \frac{1}{2\sigma} \exp\left(-\frac{\|m-y\|}{\sigma}\right)$
Log-Laplace distribution (i.e., $\log(m)$ is Laplace distributed)	`log-laplace`	$\pi(m\|y,\sigma) = \frac{1}{2\sigma m} \exp\left(-\frac{\|\log m - \log y\|}{\sigma}\right)$

The distributions above are for a single data point. For a collection $D = \{m_i\}_i$ of data points and corresponding simulations $Y = \{y_i\}_i$ and noise parameters $\Sigma = \{\sigma_i\}_i$, the current specification assumes independence, i.e. the full distribution is

$$\pi(D|Y,\Sigma) = \prod_i \pi(m_i|y_i,\sigma_i)$$

10 Parameter table

A tab-separated value text file containing information on model parameters.

This table *must* include the following parameters:

- Named parameter overrides introduced in `targetValue` expressions in the *condition table*, unless already defined in the model
- Named parameter overrides introduced in `observableParameters` or `noiseParameters` in the *measurement table*
- Parameters introduced in the `observableFormula` or `noiseFormula` expressions in the *observable table*

and *must not* include:

- Placeholder parameters (see `observableParameters` and `noiseParameters` above)
- Parameters occurring as `targetId` in the *condition table*
- "Parameters" that are not *constant* entities (e.g., in an SBML model, the targets of *AssignmentRules* or *EventAssignments*)
- Any parameters that do not have valid PEtab IDs. (For example, *local* parameters in an SBML model, which are not globally accessible, cannot be used in PEtab; if they are to be estimated or changed, they must be converted to global parameters first.)

it *may* include:

- Any model parameter that was not excluded above

One row per parameter with arbitrary order of rows and columns:

parameterId	**[parameterName]**	**lowerBound**	**upperBound**	**nominalValue**	**estimate**	...
PETAB_ID	[STRING]	NUMERIC	NUMERIC	NUMERIC	true|false	...
k1		1e-5	1e5	100	true	...
...	...	...	...	...	...	...

(wrapped for readability)

...	**[priorDistribution]**	**[priorParameters]**
...	*see below*	*see below*
...	normal	1000;100
...	...	...

Additional columns may be added.

See section 13 for details on how and when the values of parameter table parameters are applied.

10.1 Detailed field description

- `parameterId` [PETAB_ID, REQUIRED]

The `parameterId` of the parameter described in this row. This has to match the ID of a parameter specified in at least one model, a parameter introduced as override in the condition table, or a parameter occurring in the `observableParameters` or `noiseParameters` column of the measurement table (see above).

- `parameterName` [STRING, OPTIONAL]

  Parameter name to be used, e.g., for plotting etc. Can be chosen freely.

- `lowerBound` [NUMERIC]

  Lower bound of the parameter used for estimation. Optional, if `estimate==false`.

- `upperBound` [NUMERIC]

  Upper bound of the parameter used for estimation. Optional, if `estimate==false`.

- `nominalValue` [NUMERIC]

  Some parameter value to be used if the parameter is not subject to estimation (see `estimate` below). Optional, unless `estimate==false`.

- `estimate` [`true` | `false`]

  `true` or `false` (case-sensitive), depending on, if the parameter is estimated (`true`) or set to a fixed value (`false`) (see `nominalValue`).

- `priorDistribution` [STRING, OPTIONAL]

  Prior types used for the MAP objective function and for Bayesian inference (section 14). It is valid to have no priors. However, if priors are specified for a subset of parameters, this defaults to the `uniform` prior (with `priorParameters` set to `lowerBound;upperBound`) for the other parameters.

  Prior distributions are truncated by the `lowerBound` and `upperBound` if the prior's domain exceeds the parameter bounds. A non-truncated prior can be created by setting the parameter bounds to match the prior's domain (e.g., `0` and `inf` for `log-normal`). For supported prior distributions see section 10.2.

- `priorParameters` [STRING, OPTIONAL]

  Prior parameters used for the MAP objective function and for Bayesian inference (section 14). `priorParameters` is required if `priorDistribution` is non-empty.

  Only numeric values are supported (no parameter IDs). For available parameters see section 10.2.

10.2 Prior distributions

The prior distributions supported for the MAP objective function and for Bayesian inference (section 14) are listed below. PEtab only supports univariate prior distributions. The probability density functions (PDFs) below assume that the parameter bounds are wide enough to not truncate the distributions. If the parameter bounds are narrower than the distribution's support, the distributions are truncated, resulting in the following truncated prior distribution:

$$\pi_{\text{trunc}}(x) = \frac{\pi(x)}{\text{CDF(upperBound)} - \text{CDF(lowerBound)}}$$

where $\pi(x)$ is the PDF of the non-truncated distribution and $\text{CDF}(\cdot)$ its cumulative distribution function.

Let x denote the parameter value and Γ the Gamma function, then the following prior distributions are supported:

priorDistribution	**priorParameters**	**Probability density function (PDF)**	**Domain**
`cauchy`	location (μ); scale (σ)	$\pi(x\|\mu,\sigma) = \frac{1}{\pi\sigma\left(1+\left(\frac{x-\mu}{\sigma}\right)^2\right)}$	$(-\infty,\infty)$
`chisquare`	degrees of freedom (ν)	$\pi(x\|\nu) = \frac{x^{\nu/2-1}e^{-x/2}}{2^{\nu/2}\Gamma(\nu/2)}$	$(0,\infty)$
`exponential`	scale (θ)	$\pi(x\|\theta) = \frac{1}{\theta}e^{-x/\theta}$	$(0,\infty)$
`gamma`	shape (α); scale (θ)	$\pi(x\|\alpha,\theta) = \frac{x^{\alpha-1}e^{-x/\theta}}{\Gamma(\alpha)\theta^{\alpha}}$	$(-\infty,\infty)$
`laplace`	location (μ); scale (σ)	$\pi(x\|\mu,\sigma) = \frac{1}{2\sigma}\exp\left(-\frac{\|x-\mu\|}{\sigma}\right)$	$(-\infty,\infty)$
`log-laplace`	location (μ); scale (σ)	$\pi(x\|\mu,\sigma) = \frac{1}{2\sigma x}\exp\left(-\frac{\|\log(x)-\mu\|}{\sigma}\right)$	$(0,\infty)$
`log-normal`	mean (μ); standard deviation (σ)	$\pi(x\|\mu,\sigma) = \frac{1}{x\sqrt{2\pi}\sigma}\exp\left(-\frac{(\log(x)-\mu)^2}{2\sigma^2}\right)$	$(0,\infty)$
`log-uniform`	lower bound (a); upper bound (b)	$\pi(x\|a,b) = \frac{1}{x(\log(b)-\log(a))}$	$[a,b]$
`normal`	mean (μ); standard deviation (σ)	$\pi(x\|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	$(-\infty,\infty)$
`rayleigh`	scale (σ)	$\pi(x\|\sigma) = \frac{x}{\sigma^2}\exp\left(\frac{-x^2}{2\sigma^2}\right)$	$(0,\infty)$
`uniform`	lower bound (a); upper bound (b)	$\pi(x\|a,b) = \frac{1}{b-a}$	$[a,b]$

11 Mapping table

The mapping table maps PEtab entity IDs to model entity IDs, and may be used for additional annotations of model or PEtab entities. This file is optional.

This file may be used to provide PEtab-compatible aliases to model entities whose ID in the model would not be a valid identifier in PEtab (e.g., due to inclusion of blanks, dots, or other special characters), and thus, would not be allowed in the PEtab problem files.

The TSV file has two mandatory columns, `petabEntityId` and `modelEntityId`. Additional columns are allowed.

petabEntityId	**modelEntityId**	**[name]**
PETAB_ID	STRING	STRING
reaction1_k1	reaction1.k1	reaction1 k1

11.1 Detailed field description

- `petabEntityId` [PETAB_ID, REQUIRED]

A valid PEtab identifier (see section 16) that is not defined in any other part of the PEtab problem. This identifier may be referenced in condition, measurement, parameter and observable tables, but cannot be referenced in the model itself.

The `petabEntityId` may be the same as the `modelEntityId`, but it must not be used to alias an entity that already has a valid PEtab identifier. This restriction is to avoid unnecessary complexity in the PEtab problem files.

- `modelEntityId` [STRING or empty, REQUIRED]

  A globally unique identifier defined in any model, or empty if the entity is not present in any model. This does not have to be a valid PEtab identifier. Rows with empty `modelEntityId` serve as annotations only.

  This table may be used, for example, to reference specific species in a BNGL model that may contain many unsupported characters such as ',', '(' or '.'. However, please note that IDs must exactly match the species names in the BNGL-generated network file, and no pattern matching will be performed.

- `name` [STRING or empty, OPTIONAL]

  A human-readable name for the entity. This is optional and may be used for reporting or visualization purposes. Any tool making use of this value should default to the `petabEntityId` if this field is empty.

12 Problem configuration file

To link the model, measurement table, condition table, etc. in an unambiguous way, we use a YAML file. This file also allows specifying a PEtab version and employed PEtab `extensions`.

The format is described by the following JSON schema [9], which allows for easy validation:

Listing 1: JSON schema [9] (in YAML format) for PEtab problem configuration files.

```yaml
# For syntax see: https://json-schema.org/understanding-json-schema
$schema: "https://json-schema.org/draft/2020-12/schema"
description: PEtab 2.0 parameter estimation problem configuration schema.

definitions:
  list_of_files:
    type: array
    description: List of files.
    items:
      type: string
      description: |
        File name or URL, absolute or relative to the location of the PEtab
        problem configuration file.
  version_number:
    type: string
    pattern: ^([1-9][0-9]*!)?(0|[1-9][0-9]*)(\.(0|[1-9][0-9]*))*((a|b|rc)(0|[1-9][0-9]*))
        ?(\.post(0|[1-9][0-9]*))?(\.dev(0|[1-9][0-9]*))?$
    description: Version number.
```

```
properties:
  format_version:
    anyOf:
      - $ref: "#/definitions/version_number"
      - type: integer
    description: Version of the PEtab format.
  id:
    type: string
    description: |
      Identifier of the PEtab problem.
      This is optional and has no effect on the PEtab problem itself.
    pattern: "^[a-zA-Z_]\\w*$"
  parameter_files:
    description: List of PEtab parameter files.
    $ref: "#/definitions/list_of_files"
  model_files:
    type: object
    description: One or multiple models.
    # the model ID
    patternProperties:
      "^[a-zA-Z_]\\w*$":
        type: object
        properties:
          location:
            type: string
            description: |
              Model file name or URL, absolute or relative to the location of
              the PEtab problem configuration file.
          language:
            type: string
            description: |
              Model language, e.g., 'sbml', 'cellml', 'bngl', 'pysb'
        required:
          - location
          - language
    additionalProperties: false
  measurement_files:
    description: List of PEtab measurement files.
    $ref: "#/definitions/list_of_files"
  condition_files:
    description: List of PEtab condition files.
    $ref: "#/definitions/list_of_files"
  experiment_files:
    description: List of PEtab experiment files.
    $ref: "#/definitions/list_of_files"
  observable_files:
    description: List of PEtab observable files.
    $ref: "#/definitions/list_of_files"
  mapping_files:
```

```
    description: List of PEtab mapping files.
    $ref: "#/definitions/list_of_files"
  extensions:
    type: object
    description: |
      PEtab extensions being used.
    patternProperties:
      "^[a-zA-Z][\\-\\w]*$":
        type: object
        description: |
          Information on a specific extension
        properties:
          version:
            $ref: "#/definitions/version_number"
          required:
            type: boolean
            description: |
              Indicates whether the extension is required for the
              mathematical interpretation of the problem.
        required:
          - version
          - required
      additionalProperties: true
    additionalProperties: false

required:
  - format_version
  - parameter_files
  - model_files
  - observable_files
  - measurement_files
additionalProperties: false
```

12.1 Parameter estimation problems combining multiple models

12.1.1 Purpose

PEtab supports defining multiple models within a single problem specification. This feature is designed to enable users to define experiment-specific model variants or submodels. Rather than implementing a single global, parameterized model, users can define multiple smaller, self-contained models that differ structurally as needed.

This approach offers several benefits:

- Simplified model definition for users, as each variant can be independently specified.
- Improved simulation performance for tool developers, as smaller models can be simulated more efficiently.

12.1.2 Scope and Application

While multiple models are intended to be applied to different experiments, model selection is specified at the level of individual data points in the `measurement table(s)`. This design enables:

- Reuse of experiments across models.
- Fine-grained model-to-data assignment.

With the exception of the `measurement table(s)`, all other PEtab tables apply to all models. Parameters listed in the parameter table are defined globally and shared across all models. In contrast, entries in all other tables implicitly define model-specific instances of observables, conditions, experiments, etc., with their respective PEtab IDs existing in local, model-specific namespaces. Each PEtab subproblem defined in this way must constitute a valid PEtab problem on its own.

This design has several implications:

- A single experiment may need to be simulated with different models for different measurements. However, a single simulation of a given experiment is always performed using one single model.
- Each model may be associated with a distinct subset of experiments.
- The number of conditions to be simulated for a model-specific instance of an experiment may vary across models.
- Each parameter defined in the `parameter table(s)` has a shared value across all models. Parameters not listed in the parameter table(s) do not share values, which can result in model-specific instantiations of model observables referencing these parameters.

12.1.3 Validation Rules

For any given model, only those experiments and observables that appear in the same rows of the `measurement table(s)` need to be valid. This means that all symbols used in the corresponding `observableFormula` and all symbols assigned in the associated condition definitions must be defined in the model.

Conditions and observables that are not applied to a model do not need to be valid for that model.

13 Initialization and parameter application

The following describes how the model is initialized and how other PEtab parameter values are applied before the simulation starts. This procedure is executed for each model and experiment combination defined in the PEtab problem.

1. Pre-initialization
    1. Parameter values for parameters that occur in the parameter table are applied to the uninitialized model. *Uninitialized* means that no model-internal initial values have been computed yet (e.g., in SBML models, no initial assignments have been evaluated and no derived initial quantities such as concentrations have been computed yet). For estimated parameters, the respective externally-provided values are applied and for non-estimated parameters, the nominal values from the parameter table are applied.
    2. The time is set to the start time of the first period of the given experiment (or 0 if there is no explicit experiment), and any experiment-specific conditions provided for the first

period are applied (see section 6.2 for details). Target value expressions for the first period must not refer to model entities that are not also listed in the parameter table.

This pre-initialization replaces any model-internal initial values (or initialization constructs such as SBML's *InitialAssignments*) that would alter the initial time, the values of the parameters contained in the parameter table or any target values from the condition table.

2. Model initialization

   The model is initialized according to the model semantics, e.g., by evaluating the initial assignments or applying the initial values of state variables.

3. Simulation

   Model simulation starts with evaluating any event triggers at the initial time, applying pending event assignments, and evaluating the observables at the initial time (Steps 4 and 5 in section 6.2).

   For any subsequent experiment periods, the steps described in section 6.2 are applied at their start times.

14 Frequentist vs. Bayesian inference

PEtab supports maximum likelihood estimation maximum a posteriori (MAP) estimation , and Bayesian inference.

For MAP estimation and Bayesian inference, the prior distributions $p(\theta)$ of the model parameters θ are specified in the parameter table (`priorDistribution` and `priorParameters` columns, as described above), while for maximum likelihood estimation, the prior distributions are not specified. If priors are only specified for a subset of inferred parameters, the remaining parameters are assumed to have a uniform prior between their lower and upper bounds.

For maximum likelihood estimation, the objective function is the negative log-likelihood of the data $\mathcal{D}$ given the model $\mathcal{M}$ with parameters:

$$\mathcal{L}_{\mathrm{ML}}(\theta) = -\log p(\mathcal{D} \mid \mathcal{M}, \theta)$$
$$\theta_{\mathrm{ML}} = \arg\min_{\theta} \mathcal{L}_{\mathrm{ML}}(\theta)$$

Where $p(\mathcal{D} \mid \mathcal{M}, \theta)$ is the likelihood as described in section 9.2.

For MAP estimation, the objective function is the unnormalized negative log-posterior of the data given the model and the parameters:

$$\mathcal{L}_{\mathrm{MAP}}(\theta) = -\log p(\mathcal{D} \mid \mathcal{M}, \theta) - \log p(\theta)$$
$$\theta_{\mathrm{MAP}} = \arg\min_{\theta} \mathcal{L}_{\mathrm{MAP}}(\theta)$$

Different tools *may* use different objective function formulations internally, as long as they preserve the optima. However, tools *should* provide the final objective function value, as the negative log-likelihood or unnormalized negative log-posterior, to facilitate comparison and reproducibility.

Example In the case of independently and normally distributed noise, the negative log-likelihood would be:

$$\mathcal{L}_{\mathrm{ML}}(\theta) = \frac{1}{2} \sum_{i=1}^{N} \left(\log(2\pi\sigma_i^2) + \frac{(m_i - y_i)^2}{\sigma_i^2} \right)$$

where m_i is the measurement, y_i is the model output, and σ_i is the standard deviation of the noise for the i-th measurement.

15 Math expressions syntax

This section describes the syntax of math expressions used in PEtab files, such as the observable formulas.

Supported symbols, literals, and operations are described in the following. Whitespace is ignored in math expressions.

15.1 Symbols

- The supported identifiers are:
  - parameter IDs from the `parameter table(s)`
  - model entity IDs that are globally unique and have a clear interpretation in a math expression context
  - observable IDs from the observable table
  - PEtab placeholder IDs in the observable and noise formulas
  - PEtab entity IDs in the `mapping table(s)`
  - `time` for the model time (section 15.1.1)
  - PEtab function names (Table 13)

  Identifiers are not supported if they do not match the PEtab identifier format. PEtab expressions may have further context-specific restrictions on supported identifiers.
- The functions defined in PEtab are listed in Table 13. Other functions, including those defined in the model, remain undefined in PEtab expressions.
- Special symbols (such as e and π) are not supported, and neither is NaN (not-a-number).

15.1.1 Model time

The model time is represented by the symbol `time`, which is the current simulated time, not the current duration of simulated time; if the simulation starts at $t_0 \neq 0$, then `time` is *not* the time since t_0.

15.2 Literals

15.2.1 Numbers

All numbers, including integers, are treated as floating point numbers of undefined precision (although no less than double precision should be used. Only decimal notation is supported. Scientific notation is supported, with the exponent indicated by `e` or `E`. The decimal separator is indicated by `.`. Examples of valid numbers are: `1`, `1.0`, `-1.0`, `1.0e-3`, `1.0e3`, `1e+3`. The general syntax in PCRE2 regex is `\d*(\.\d+)?([eE][-+]?\d+)?`. `inf` and `-inf` are supported as positive and negative infinity.

15.2.2 Booleans

Boolean literals are `true` and `false`.

15.3 Operations

15.3.1 Operators

The supported operators are listed in Table 12.

Note that operator precedence might be unexpected, compared to other programming languages. Use parentheses to enforce the desired order of operations.

Operators must be specified; there are no implicit operators. For example, `a b` is invalid, unlike `a * b`.

15.3.2 Functions

The supported functions are listed in Table 13.

15.3.3 Boolean ⇔ float conversion

Boolean and float values are implicitly convertible. The following rules apply:

- bool → float: `true` is converted to `1.0`, `false` is converted to `0.0`.
- float → bool: `0.0` is converted to `false`, all other values are converted to `true`.

Operands and function arguments are implicitly converted as needed. If there is no signature compatible with the given types, Boolean values are promoted to float. If there is still no compatible signature, float values are demoted to boolean values. For example, in `1 + true`, `true` is promoted to `1.0` and the expression is interpreted as `1.0 + 1.0 = 2.0`, whereas in `1 && true`, `1` is demoted to `true` and the expression is interpreted as `true && true = true`.

Table 12: Supported operators in PEtab math expressions.

Operator	Precedence	Interpretation	Associativity	Arguments	Evaluates to

`f(`
`  arg1[,`
`  arg2,`
`  ...]`
`)`
	1	call to function *f* with arguments *arg1*, *arg2*, ...	left-to-right	any	input-dependent

`()`	1	parentheses for grouping acts like identity		any single expression	argument
`^`	2	exponentiation (shorthand for pow)	right-to-left	float, float	float
`+` `-`	3	unary plus unary minus	right-to-left	float	float
`!`	3	logical *not*		bool	bool
`*` `/`	4	multiplication division	left-to-right	float, float	float
`+` `-`	5	binary plus, addition binary minus, subtraction	left-to-right	float, float	float

`<`
`<=`
`>`
`>=`
	6	less than
		less than or equal to
		greater than
		greater than or equal to
			left-to-right	float, float	bool

`==` `!=`	6	is equal to is not equal to	left-to-right	(float, float) or (bool, bool)	bool
`&&` `||`	7	logical *and* logical *or*	left-to-right	bool, bool	bool
`,`	8	function argument separator	left-to-right	any	

Table 13: Supported functions in PEtab math expressions.

Function	Comment	Argument types	Evaluates to
`pow(a, b)`	power function b-th power of a	float, float	float
`exp(x)`	exponential function pow(e, x) (e itself not a supported symbol, but `exp(1)` can be used instead)	float	float
`sqrt(x)`	square root of `x` `pow(x, 0.5)`	float	float

`log(a, b)`
`log(x)`
`ln(x)`
`log2(x)`
`log10(x)`
	logarithm of `a` with base `b`
	`log(x, e)`
	`log(x, e)`
	`log(x, 2)`
	`log(x, 10)`
	(`log(0)` is defined as `-inf`)
	(NOTE: `log` without explicit base is `ln`, not `log10`)
		float[, float]	float

`sin`
`cos`
`tan`
`cot`
`sec`
`csc`
	trigonometric functions	float	float

`arcsin`
`arccos`
`arctan`
`arccot`
`arcsec`
`arccsc`
	inverse trigonometric functions	float	float

`sinh`
`cosh`
`tanh`
`coth`
`sech`
`csch`
	hyperbolic functions	float	float

`arcsinh`
`arccosh`
`arctanh`
`arccoth`
`arcsech`
`arccsch`
	inverse hyperbolic functions	float	float

`piecewise(`
`  true_value_1,`
`  condition_1,`
`  [true_value_2,`
`  condition_2,]`
`  [...]`
`  [true_value_n,`
`  condition_n,]`
`  otherwise`
`)`
	The function value is the `true_value*` for the first `true condition*` or `otherwise` if all conditions are `false`.	`*value*`: all float or all bool
		`condition*`: all bool
			float

`abs(x)`	absolute value `piecewise(x, x>=0, -x)`	float	float
`sign(x)`	sign of `x` `piecewise(1, x>0, -1, x<0, 0)`	float	float

`min(a, b)`
`max(a, b)`
	minimum / maximum of {`a`, `b`}
	`piecewise(a, a<=b, b)`
	`piecewise(a, a>=b, b)`
		float, float	float

16 Identifiers

- All identifiers in PEtab may only contain upper and lower case ASCII letters, digits and underscores, and must not start with a digit. In PCRE2 regex, they must match `[a-zA-Z_][a-zA-Z_\d]*`.
- Identifiers are case-sensitive.
- Identifiers must not be a reserved keyword (section 16.1).
- Identifiers must be globally unique within the PEtab problem. PEtab does not put any further restrictions on the use of identifiers within the model, which means modelers could potentially use model-format-specific (e.g. SBML) function names as identifiers. However, this is strongly discouraged.

16.1 Reserved keywords

The following keywords, case-insensitive, are reserved and must not be used as identifiers:

- `true`, `false`: Boolean literals, used in PEtab expressions.
- `inf`: Infinity, used in PEtab expressions and for steady-state measurements
- `time`: Model time, used in PEtab expressions.
- `nan`: Undefined in PEtab, but reserved to avoid implementation issues.
- PEtab math function names (section 15.3.2)

17 Extensions

An elaborate, monolithic format would make it difficult to understand and implement support for PEtab, leading to a steep learning curve and discouraging support in new toolboxes. To address this issue, the PEtab format is modular and permits modifications through extensions that complement the core standard. This modular specification evens the learning curve and provides toolbox developers with more guidance on which features to implement to maximize support for real world applications. Moreover, such modular extensions facilitate and promote the use of specialized tools for specific, non-parameter estimation tasks such as visualization.

Rules for extensions:

- Specifications in PEtab extensions take precedence over PEtab core, i.e., they may ease or refine format restrictions imposed by PEtab core.
- PEtab extensions should extend PEtab core with new orthogonal features or tasks, i.e., they should not make trivial changes to PEtab core.
- PEtab extensions must be named according to the following regular expression: `^[a-zA-Z][a-zA-Z0-9_\-]*$`.
- PEtab extensions must be versioned using semantic versioning.
- Extensions are specified in the `problem configuration file` (section 12) under the key `extensions`. If an extension changes the mathematical interpretation of a PEtab problem, its `required` attribute must be set to `true`.
- Toolboxes may ignore extensions with `required: false` that they do not support.

- Toolboxes must reject PEtab problems that use extensions with `required: true` that they do not support.

18 Acknowledgments

The authors thank the PEtab community for their valuable feedback that went into the development of PEtab 2.0. M.K. acknowledges support by the BMBF-funded de.NBI Cloud within the German Network for Bioinformatics Infrastructure (de.NBI) (031A537B, 031A533A, 031A538A, 031A533B, 031A535A, 031A537C, 031A534A, 031A532B).

19 Author contributions

All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

20 Research funding

D.W. received funding from the German Federal Ministry of Education and Research (BMBF) within the e:Med funding scheme (junior research alliance PeriNAA, grant no. 01ZX1916A). M.K. received funding from the German Federal Ministry of Research, Technology and Space (BMFTR, Germany) within ATLAS by grant number 031L0304B and by the German Research Foundation (DFG) by the DFG grant number 436883643 and 465194077. F.T.B. was supported by LIBIS (Baden-Württemberg Institute for Bioinformatics Infrastructure), and de.NBI, the German Network for Bioinformatics Infrastructure (W-de.NBI-016). This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy (EXC 2047—390685813, EXC 2151—390873048), by the European Union via ERC grant INTEGRATE (grant no 101126146), and by the University of Bonn (via the Schlegel Professorship of J.H.). This work was funded by the European Union (ERC, DeepMechanism, grant no 101163005) and the Francis Crick Institute, which receives its core funding from Cancer Research UK (CC2242), the UK Medical Research Council (CC2242), and the Wellcome Trust (CC2242).

21 Competing interests

The authors state no conflict of interest.

References

[1] YAML Ain't Markup Language (YAML™) version 1.2, revision 1.2.2. https://yaml.org/spec/1.2.2/, 2021. Accessed 2026-01-16.

[2] Michael Clerx, Michael T. Cooling, Jonathan Cooper, Alan Garny, Keri Moyle, David P. Nickerson, Poul M. F. Nielsen, and Hugh Sorby. CellML 2.0. *Journal of Integrative Bioinformatics*, 17(2-3):20200021, 2020. URL: https://doi.org/10.1515/jib-2020-0021 [cited 2026-01-05], doi:doi:10.1515/jib-2020-0021.

[3] James R. Faeder, Michael L. Blinov, and William S. Hlavacek. *Rule-Based Modeling of Biochemical Systems with BioNetGen*, pages 113–167. Humana Press, Totowa, NJ, 2009. doi:10.1007/978-1-59745-525-1_5.

[4] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003. doi:10.1093/bioinformatics/btg015.

[5] Sarah M Keating, Dagmar Waltemath, Matthias König, Fengkai Zhang, Andreas Dräger, Claudine Chaouiya, Frank T Bergmann, Andrew Finney, Colin S Gillespie, Tomáš Helikar, Stefan Hoops, Rahuman S Malik-Sheriff, Stuart L Moodie, Ion I Moraru, Chris J Myers, Aurélien Naldi, Brett G Olivier, Sven Sahle, James C Schaff, Lucian P Smith, Maciej J Swat, Denis Thieffry, Leandro Watanabe, Darren J Wilkinson, Michael L Blinov, Kimberly Begley, James R Faeder, Harold F Gómez, Thomas M Hamm, Yuichiro Inagaki, Wolfram Liebermeister, Allyson L Lister, Daniel Lucio, Eric Mjolsness, Carole J Proctor, Karthik Raman, Nicolas Rodriguez, Clifford A Shaffer, Bruce E Shapiro, Joerg Stelling, Neil Swainston, Naoki Tanimura, John Wagner, Martin Meier-Schellersheim, Herbert M Sauro, Bernhard Palsson, Hamid Bolouri, Hiroaki Kitano, Akira Funahashi, Henning Hermjakob, John C Doyle, Michael Hucka, Richard R Adams, Nicholas A Allen, Bastian R Angermann, Marco Antoniotti, Gary D Bader, Jan Červený, Mélanie Courtot, Chris D Cox, Piero Dalle Pezze, Emek Demir, William S Denney, Harish Dharuri, Julien Dorier, Dirk Drasdo, Ali Ebrahim, Johannes Eichner, Johan Elf, Lukas Endler, Chris T Evelo, Christoph Flamm, Ronan MT Fleming, Martina Fröhlich, Mihai Glont, Emanuel Gonçalves, Martin Golebiewski, Hovakim Grabski, Alex Gutteridge, Damon Hachmeister, Leonard A Harris, Benjamin D Heavner, Ron Henkel, William S Hlavacek, Bin Hu, Daniel R Hyduke, Hidde de Jong, Nick Juty, Peter D Karp, Jonathan R Karr, Douglas B Kell, Roland Keller, Ilya Kiselev, Steffen Klamt, Edda Klipp, Christian Knüpfer, Fedor Kolpakov, Falko Krause, Martina Kutmon, Camille Laibe, Conor Lawless, Lu Li, Leslie M Loew, Rainer Machne, Yukiko Matsuoka, Pedro Mendes, Huaiyu Mi, Florian Mittag, Pedro T Monteiro, Kedar Nath Natarajan, Poul MF Nielsen, Tramy Nguyen, Alida Palmisano, Jean-Baptiste Pettit, Thomas Pfau, Robert D Phair, Tomas Radivoyevitch, Johann M Rohwer, Oliver A Ruebenacker, Julio Saez-Rodriguez, Martin Scharm, Henning Schmidt, Falk Schreiber, Michael Schubert, Roman Schulte, Stuart C Sealfon, Kieran Smallbone, Sylvain Soliman, Melanie I Stefan, Devin P Sullivan, Koichi Takahashi, Bas Teusink,

David Tolnay, Ibrahim Vazirabad, Axel von Kamp, Ulrike Wittig, Clemens Wrzodek, Finja Wrzodek, Ioannis Xenarios, Anna Zhukova, and Jeremy Zucker. SBML level 3: an extensible format for the exchange and reuse of biological models. *Molecular Systems Biology*, 16(8), August 2020. doi:10.15252/msb.20199110.

[6] Paul Lindner. text/tab-separated-values. https://www.iana.org/assignments/media-types/text/tab-separated-values, 1993. Retrieved 5 January 2026.

[7] Leonard Schmiester, Yannik Schälte, Frank T. Bergmann, Tacio Camba, Erika Dudkin, Janine Egert, Fabian Fröhlich, Lara Fuhrmann, Adrian L. Hauber, Svenja Kemmer, Polina Lakrisenko, Carolin Loos, Simon Merkt, Wolfgang Müller, Dilan Pathirana, Elba Raimúndez, Lukas Refisch, Marcus Rosenblatt, Paul L. Stapor, Philipp Städter, Dantong Wang, Franz-Georg Wieland, Julio R. Banga, Jens Timmer, Alejandro F. Villaverde, Sven Sahle, Clemens Kreutz, Jan Hasenauer, and Daniel Weindl. PEtab—interoperable specification of parameter estimation problems in systems biology. *PLOS Computational Biology*, 17(1):1–10, 01 2021. doi:10.1371/journal.pcbi.1008646.

[8] Alejandro F Villaverde, Dilan Pathirana, Fabian Fröhlich, Jan Hasenauer, and Julio R Banga. A protocol for dynamic model calibration. *Briefings in Bioinformatics*, 23(1):bbab387, 10 2021. doi:10.1093/bib/bbab387.

[9] Austin Wright, Henry Andrews, Ben Hutton, and Greg Dennis. JSON schema: A media type for describing JSON documents (draft 2020-12). https://json-schema.org/draft/2020-12/json-schema-core.html, 2022.