Mnemonics

A **mnemonic** defines a single field of **numeric** data in a XINA model. A **datapoint** is a single record of a mnemonic, consisting of:

- time (Unix microseconds)
- mnemonic identifier
- value (numeric)

In other words, **the value of a single mnemonic at a moment in time**.

A model has one or more mnemonic databases, containing all of the datapoints associated with the model.

Mnemonic Definitions

All mnemonics are defined in a **mnemonic definitions** database. It is **strongly recommended** to use a single definitions database for an entire project to faciliate comparison of data between models.

A core challenge of working with mnemonics is synchronizing mnemonic definitions from XINA to the point of data collection. Especially in early test environments, fields may be frequently added or removed on the the fly and labels may change, but must be consistently associated with a single mnemonic definition. Broadly there are two approaches to manage this challenge.

The first is user maintained mnemonic definitions. This is recommended for environments without frequent changes, and ideally one data source. The end user is responsible for ensuring that imported data has matching `mn_id` values to mnemonics present in the definitions database. This will typically result in faster imports and support complex or custom data pipeline solutions.

The second solution is allowing XINA to manage mnemonic definitions. With this approach, data can be imported with plain text labels and automatically associated with mnemonic definitions if available, or new definitions can be created on the fly.

Both approaches can be accomplished with the `model_mn_import` API action, documented here. The details of the required approach will depend on project requirements.

Fields

Mnemonic ID

Unique numeric ID, assigned by XINA.

Name

Mnemonic name. Conforms to structs naming conventions. Must be unique in combination with subname and unit.

Subname

Essentially a name suffix.

Description

Optional plain text mnemonic description.

Unit

Optional (but strongly recommended) measurement unit (for example, `V`, `mA`, etc).

Standard Fields

field	type	description
`state`	`model_mn_state`	current state of mnemonic (`active`, `inactive`, `archived`, `deprecated`)
`origins`	`jsonobject`	map of model(s) to associated origin(s)
`full`	`asciivstring(32)`	the primary database for the mnemonic, default `f8` (may be `null`)
`bin`	`set(asciivstring(32))`	the opt-in bin database(s) to include the mnemonic in
`format`	`asciivstring(32)`	printf-style format to render values
`enum`	`jsonobject`	mapping of permitted text values to numeric values
`labels`	`list(jsonobject)`	mapping of numeric values or ranges to labels
`aliases`	`set(asciivstring(128))`	set of additional names associated with the mnemonic
`meta`	`jsonobject`	additional metadata as needed
`query`	`asciivstring(32)`	query name for meta-mnemonics (may be `null`)
`conf`	`jsonobject`	configuration for meta-mnemonics (may be `null`)

Although the mnemonic name is intended to be unique, insertion of a mnemonic with the same name but different unit will create a new mnemonic definition. This is intended to avoid interruption of data flow, but should be corrected with the Mnemonic Management tool when possible. The `model` and `origin` are populated automatically for auto-generated mnemonic definitions.

The mnemonic `state` affects how the mnemonic will be displayed and populated. An `inactive` mnemonic indicates data is no longer relevant or actively populated and will be hidden by default. A `deprecated` mnemonic extends this concept but will throw errors if additional data points for the mnemonic are imported.

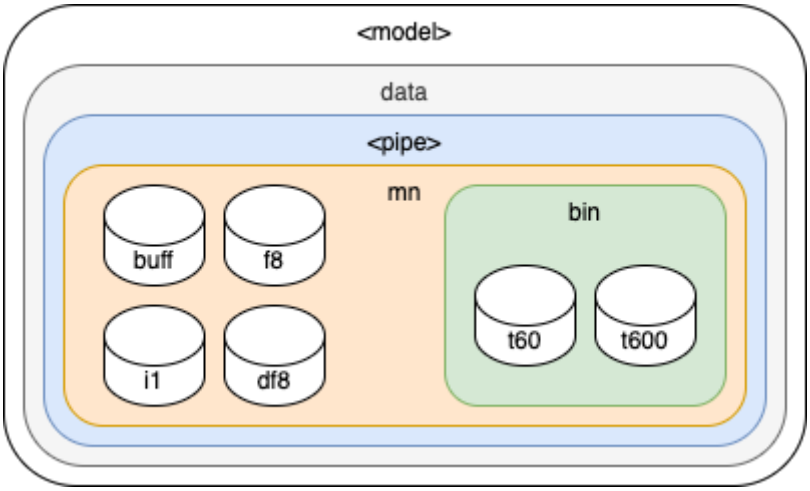
If `enum` is provided a mnemonic will apply labels to enumerated numeric values, as provided in `values`. For example, a 0|1 on|off state could be represented by `{"0":"OFF", "1":"ON"}`. Values in this map may also be used to parse imported data.

A mnemonic may specify one or more `aliases` to indicate additional names that should be included in the single mnemonic definition. If present, the aliases are referenced at a **higher priority** than the mnemonic name during import lookup. For example, a given mnemonic `a` is erroneously labeled `b` in some imported data, which creates a new separate mnemonic definition for `b`. To correct this, `b` could be added as an alias for `a`, and the `b` mnemonic could be deprecated. All `a` and `b` data from the source telemetry would then correctly be merged into the `a` mnemonic.

`name`, `unit`, `state`, `enum`, `models`, and `aliases` may be used during the data import process to validate and interpret data. Full details of how each field is used is documented with the associated API action.

Mnemonic Databases

Within a model, each data source must have a set of one or more mnemonic databases. Each set should be contained by a group, which can be configured to define any relationships between the databases. This will typically include a **full** database, containing all or **delta** optimized data (see below for additional information), and one or more types of **bin** databases, depending on requirements.



While each data source must have its own mnemonic database(s), it may be beneficial for a single data source to further subdivide mnemonics into different types of databases for optimization purposes. For example, a model with a large number of mnemonics that only require single byte precision would see significant performance gains from separate databases using the `int(1)` type. In practice this could look like:

Full Database

In most cases, there will be a single primary database containing **full mnemonic** data (all points from original telemetry), **delta mnemonic** data (an optimization option, see below), or a mix of both. Data is stored with a single data point per row.

A value of `null` may be used for `v` to indicate a gap in data, otherwise data will appear visually connected by default in XINA charts. `null` may also be appropriate to represent `NaN` or `Inf` values, as these cannot be stored in the database, but the preference to include these as `null` or omit them altogether may depend on an individual mnemonic.

For large data sets with infrequent value changes, it may be beneficial to employ a **delta mnemonic** optimization. This requires the `n` field listed above. In this case, a point is only included in the database at the moment the value for a given mnemonic changes, and the number of points is stored in `n`. For example, given the set of points:

t	v
0	0
1	0
2	0
3	1
4	1

t	v
5	1
6	1
7	2
8	2
9	2

Delta optimization would condense the data to:

t	v	n
0	0	2
2	0	1
3	1	3
6	1	1
7	2	2
9	2	1

Note the final data point of a data set is always included.

Bin Database(s)

The most common data optimization employed with mnemonics is **binning**, combining multiple data points over a fixed time range into a single data point with a min, max, avg, and standard deviation. A model may define one or more bin databases depending on performance requirements, but four types are supported by default. The time range of bins is interpretted as `[start, end)`.

Time Binning

Bins are applied on a **fixed time interval** for all points in the database (for example, 1 minute or 1 hour).

Standard Fields

field	type	description	required
`t`	`instant` (matching model standard)	start time of the bin	yes
`t_min`	`instant` (matching model standard)	time of first data point in bin	yes
`t_max`	`instant` (matching model standard)	time of last data point in bin	yes
`mn_id`	`int(4)`	unique mnemonic ID	yes
`n`	`int(4)`	number of data points in bin	yes
`avg`	`float(8)`	average of points in bin	yes
`min`	`float(8)`	min of points in bin	yes
`max`	`float(8)`	max of points in bin	yes

field	type	description	required
med	float(8)	median of points in bin	no
var	float(8)	variance of points in bin	no
std	float(8)	standard deviation of points in bin	no

Interval Binning

Bins are based on explicitly defined **intervals**.

Standard Fields

field	type	description	required
t_start	instant(us)	start time of the bin	yes
t_end	instant(us)	end time of the bin	yes
dur	duration(us)	duration	yes
t_min	instant(us)	time of first data point in bin	yes
t_max	instant(us)	time of last data point in bin	yes
u_id	UUID	UUID of associated interval	yes
p_id	int(8)	primary ID of associated interval	yes
s_id	int(4)	secondary ID of associated interval	yes
mn_id	int(4)	unique mnemonic ID	yes
n	int(4)	number of data points in bin	yes
avg	float(8)	average of points in bin	yes
min	float(8)	min of points in bin	yes
max	float(8)	max of points in bin	yes
med	float(8)	median of points in bin	no
var	float(8)	variance of points in bin	no
std	float(8)	standard deviation of points in bin	no

Revision #20
Created 27 July 2022 18:51:52 by Nick Dobson
Updated 17 July 2024 17:09:52 by Nick Dobson