

# 0006 — Report: updatator.py — Model Update After Validation

## 1. Purpose and role

The module `dtwinpy/updatator.py` implements **model update** when validation indicators fall **below** the allowed threshold. It can update either:

- **Logic** — Intended for future “model generation” (not implemented yet).
- **Input** — **Distribution fitting** on real process times: for each machine, fit a distribution (e.g. normal, exponential) to the real log and **write the new parameters** (or mean, for deterministic treatment) into the **model JSON** (contemp per node).

So the updatator **adjusts the digital model’s process-time parameters** using real data, so that the next validation run can perform better.

---

## 2. Dependencies and imports

```
from .helper import Helper
from .interfaceDB import Database
import numpy as np
import scipy.stats
import warnings
import matplotlib.pyplot as plt
import json
```

- **Helper** — Logging and kill.
  - **Database** — Read real log with time window and `model_update=True`.
  - **scipy.stats** — Distribution fitting (`fit`, `kstest`, `probplot`).
  - **json** — Read/write model file in `aligner()`.
- 

## 3. Updatator class — overview

Single class **Updatator**. Main flow:

- **Constructor**: Store update type (logic/input), digital model, real DB path and time window, **model\_last\_sync** (path to the JSON file to update), and create the real Database with `model_update=True`.
- **run()**: If **logic** → **update\_logic()** (stub). If **input** → **generate\_qTDS\_traces()** (machine process times from real log), **update\_input()** per machine (fit distribution, choose best by KS p-value), then **aligner()** to write **contemp** into the JSON.

The JSON that is **written** is **model\_last\_sync**, which may differ from the path used by the digital model at run time (e.g. a post-sync copy).

---

## 4. Constructor and main attributes

```
def __init__(self, update_type, digital_model, real_database_path, start_t
              model_last_sync, plot=False):
```



Parameter / attribute	Description
<b>update_type</b>	'logic' or 'input'.
<b>digital_model</b>	Model instance; used to get machines and to resolve machine by name in <b>run()</b> .
<b>real_database_path</b>	Path to DB containing <b>real_log</b> .
<b>start_time, end_time</b>	Time/event window for reading real log.
<b>model_last_sync</b>	<b>Path to the JSON file to update</b> (e.g. last synced model). All writes go to this file.
<b>plot</b>	If True, <b>update_input()</b> can draw Q-Q plots.
<b>feature_usingDB</b>	'valid_logic' (logic) or 'valid_input' (input), passed to Database.
<b>real_database</b>	Database with event_table='real_log', time window, <b>model_update=True</b> .
<b>machines_vector, queues_vector</b>	From <code>digital_model.get_model_components()</code> .

## 5. Logic update (stub)

### 5.1 update\_logic()

- Prints a warning that “Model Generation” is still in progress.
- **pass** — no actual logic update implemented.

Planned use: when validation fails on **logic** (e.g. LCSS), some future logic could alter the model structure or routing; currently only the **input** path is implemented.

## 6. Input update — distribution fitting

### 6.1 update\_input(input\_data, probable\_distribution=None)

Fits a set of distributions to **input\_data** (e.g. one machine’s process times from the real log), selects the best by **Kolmogorov–Smirnov (KS) test** p-value, and returns the distribution name, parameters, and mean.

#### Steps:

1. **Distribution list:** Only ['norm', 'expon'] are used (others are commented).
2. **Fit:** For each distribution, `scipy.stats.<dist>.fit(input_data) → parameters`.
3. **KS test:** `scipy.stats.kstest(input_data, distribution, args=parameters) → (D, p-value)`. Warnings suppressed during fit/kstest.
4. **Best:** Index with **maximum p-value** (prefer p-value close to 1); `ks = [best_dist_name, best_pvalue]`.
5. **Optional plots (if self.plot):**
  - Q-Q plot with best-fit distribution and its parameters.
  - If **probable\_distribution** is given, Q-Q plot with that distribution and its fitted parameters.
6. **Mean:**
  - If best is **norm**: `mean = parameters[0] (loc)`.
  - If best is **expon**: `mean = parameters[0] + 1/parameters[1] (loc + 1/scale)`.
7. **Return:** `(dist_name, parameters_list, mean)`.

So the caller gets both the chosen distribution and parameters (for writing ["norm",  $\mu$ ,  $\sigma$ ] or similar) and the mean (for writing a scalar when treating the system as deterministic).

## 7. Machine traces and determinism

### 7.1 generate\_qTDS\_traces()

- Same idea as in **Validator**: build process times **per machine** from the real log.
- Uses **get\_machines\_with\_completed\_traces()** (only machines with full traces).
- For each machine: **get\_time\_activity\_of\_column(column="machine\_id", table="real\_log", column\_id=machine\_id[0])** → list of (time, activity\_type).
- For each **Started–Finished** pair: `processed_time = finished_time - started_time`; append to that machine's trace.
- **Difference from validator**: If the first event is **Finished** (no preceding Started), the code **does not** add a process time; it resets and continues (see ISSUE #263: no worked\_time in update, so that finish is not carried as a process time).
- Returns **dict**: `machine_name` → [`process_time_1`, `process_time_2`, ...].
- Prints the sequence considered for update.

### 7.2 is\_deterministic()

- Loops over **machines\_vector**.
- If **any** machine has **non-list** `process_time` (i.e. a scalar), sets **flag\_is\_deterministic = True**.
- Returns that flag.

So “deterministic” here means “at least one machine uses a scalar process time”. In **run()**, when **flag\_deterministic** is True, **every** updated machine gets a **scalar** (rounded mean). When False, every updated machine gets a **list** (distribution name + parameters).

---

## 8. Writing the model JSON: aligner()

### 8.1 aligner(machine\_id, new\_process\_time)

- Opens **model\_last\_sync** (path set in constructor) in read-write mode.
- Loads JSON; finds the node with **node['activity'] == machine\_id**.
- Replaces **node["contemp"]** with **new\_process\_time** (either a number or a list, e.g. [`"norm"`, 17, 2]).
- Writes the JSON back (`seek(0)`, `dump`, `truncate`).

So the **only** field updated in the JSON for input update is **contemp** of the corresponding node. The file updated is always **model\_last\_sync**, not necessarily `digital_model.get_model_path()`.

---


## 9. run() — main entry point

1. **Logic**: If `update_type == 'logic'`, call **update\_logic()** (stub) and stop.
2. **Input**:
  - **generate\_qTDS\_traces()** → **matrix\_ptime\_qTDS** (`machine_name` → list of process times).
  - **is\_deterministic()** → **flag\_deterministic**.
  - For each **key** in **matrix\_ptime\_qTDS**:
    - Resolve **machine** = `digital_model.get_selected_machine(machine_name=key)`.
    - **machine\_trace** = `matrix_ptime_qTDS[machine_name]`.
    - **update\_result** = **update\_input(machine\_trace)** → (`dist_name`, `parameters`, `mean`).
    - **Deterministic (flag\_deterministic True)**:

- **new\_process\_time** = round(update\_result[-1]) (rounded mean).
- If negative, error and kill.
- **Non-deterministic (flag\_deterministic False):**
  - **new\_process\_time** = [dist\_name, round(param\_a), round(param\_b)]; if param\_b is 0, set to 1.
  - **aligner(machine\_id, new\_process\_time)** — write to **model\_last\_sync**.
- Print “System Updated”.

So in one **run()**, all machines that appear in **matrix\_ptime\_qTDS** get their **contemp** updated in the same JSON file, either all as scalars (deterministic) or all as distribution lists (non-deterministic).

## 10. High-level flow (Mermaid)

  
Mermaid diagram 0

## 11. Relation to other modules

Module / concept	Relation to updatator
<b>digital_model</b>	Provides machines (get_model_components, get_selected_machine); the <b>path</b> written is <b>model_last_sync</b> , which may be the same or a copy of the model path.
<b>interfaceDB</b>	<b>Database</b> with real log, time window, <b>model_update=True</b> ; <b>get_machines_with_completed_traces()</b> , <b>get_time_activity_of_column()</b> for building machine traces.
<b>validator</b>	Validator produces the <b>indicators</b> that trigger an update; updatator uses the <b>same real log</b> (and same kind of machine traces as qTDS) to refit and write the model.
<b>Model JSON</b>	Only <b>nodes[].contemp</b> is updated (per machine); file is <b>model_last_sync</b> .

## 12. Summary

- **updatator.py** updates the digital model when validation is below threshold: **logic** update is a stub; **input** update refits process-time distributions from the real log and writes them to the JSON.
- **update\_input()** fits *norm* and *expon* to a machine’s process times, picks the best by KS p-value, and returns distribution name, parameters, and mean.
- **generate\_qTDS\_traces()** builds per-machine process times from the real log (Started–Finished), matching the validator’s qTDS trace generation but without using “first event Finished” as a process time.
- **aligner()** writes the new process time (scalar or list) into **model\_last\_sync** for the given machine.
- **run()** for input: generate traces → fit per machine → write all updates to **model\_last\_sync**; if any machine is deterministic, all updated machines get a scalar (mean), otherwise all get distribution parameters.