Description of the JSON Data Format for the `tbischeduling` Application

H. Simonis

January 22, 2025

Abstract

This is a semi-generated report which describes the JSON format for input and output of the `tbischeduling` application.

1 Introduction

This report shows the different data types that are contained in the JSON input and output files for the `tbischeduling` application. The report is semi-generated, the textual description is hand-written. while the images are generated from a sample data file via the *plantuml* tool, based on the actual fields being written into an output file.

The `tbischeduling` application at startup produces a small `sampler.txt` file with the `GenerateJSONDoc` class, based on a sample dataset. It creates a dataset with `CreateData()`, using the smallest number of values that provide a full application data set. It will also produce a set of files of the form *class.txt*, which describe one instance of a class at a time. In the directory `site/jsondoc` we find a `Makefile` which will produce this report from the textual description and a set of `.png` files which are generated from the `.txt` files by *plantuml*.

1.1 Design Rules

Each object (except the *root*) described in the file has a `String` field called *name*, which is used to identify the object. Null values, empty names, or duplicated names within the same class are not acceptable for this field.

If a structure refers to another object, it uses the name of that object as a reference, instead of repeating the content of the structure in multiple locations. So an *Order* will refer to a *Product* by name, which in turn refers to a *Process* in the attribute `defaultProcess`.

2 Overview

Figure 1 shows the complete format of the input data file for the application. There is a top-level structure which contains global parameters and a version number, and one JSONArray for each data type that is stored in the application. The image is generated from a produced sample datafile using the *plantuml* drawing application.

The structures marked in green are used as foreign keys in other structures.

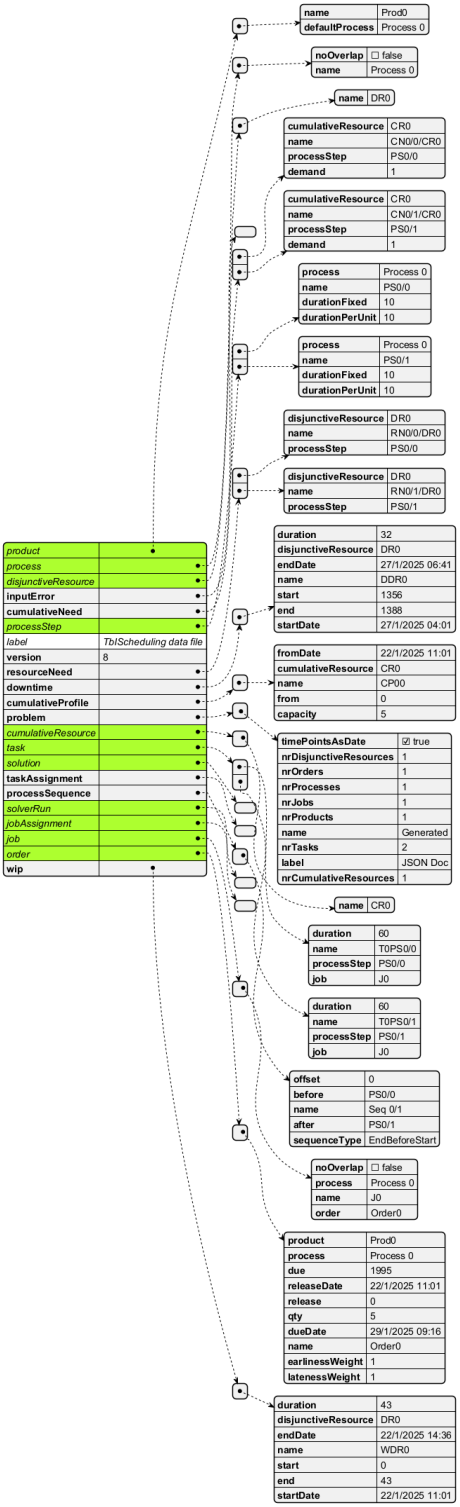

Figure 1: JSON Input Format Overview

Table 1: Color Legend

Style	Description
name	The `name` field is the primary key of a structure. The names within one class must be unique.
key	A key is a foreign key reference. It is the name of a structure of another type.
date	A date expresses time in the external representation, e.g. day, month, year, hour, minute.
elapsed	A time period measured in elapsed seconds. Used for time-out and runtime reporting.
enum	An eumerated type can only take values from a restricted vocabulary. Lower/Upper case is significant.
internaltime	Many fields use the solver internal time, which is expressed as an integer, based on the distance to a reference start date, divided by the time resolution.
optional	Some input data fields are optional, if they are not given, the value from another field is used.

3 Input Data

3.1 Problem Description

The problem description part of the data defines how the factory operates, which products it can make, and how the product will be produced described in a process. We also define available resources and the resources needed to perform some process step. This description is quite static, it will only change if new products are introduced, or processes are changed.

In the current version, we assume that we know which process to use for which product. In a future version we can allow to pick the best process to make the product for an order as part of the solution process.

Our current version allows multiple identical parallel machines, but not machine preferences or production times that vary with the machine assigned.

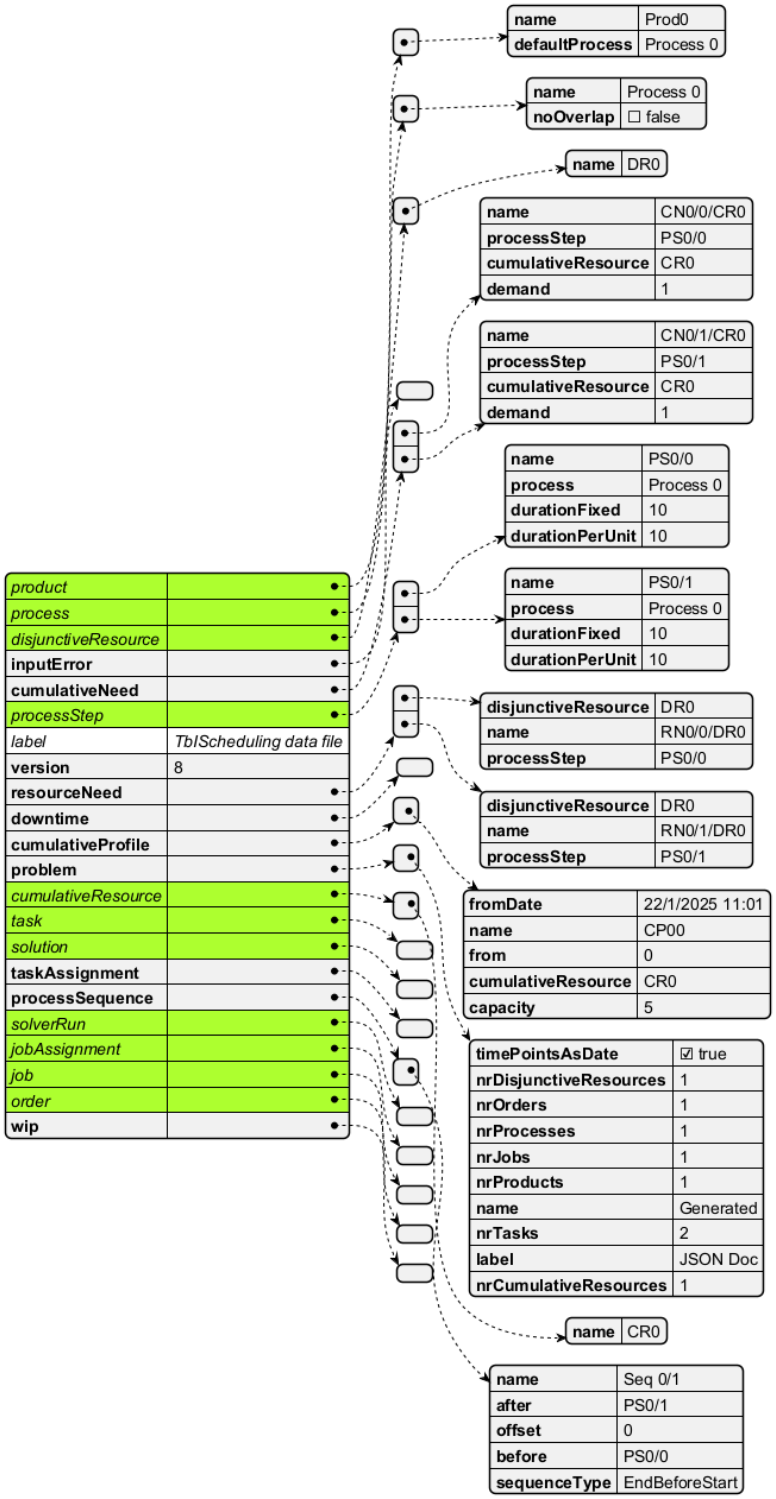

Figure 2: Base Data Defining Problem Structure

3.1.1 Problem

There can be only one *Problem* instance in each data file, which summarizes the input data.

timePointsAsDate	☑ true
nrDisjunctiveResources	1
nrOrders	1
nrProcesses	1
nrJobs	1
nrProducts	1
name	*Generated*
nrTasks	2
label	*JSON Doc*
nrCumulativeResources	1

3.1.2 Product

A *Product* describes the end product that is made for an order. There is a default process which describes how the product is made.

name	*Prod0*
defaultProcess	*Process 0*

3.1.3 Process

The *Process* describes how a product is being made. The process typically contains multiple process steps, temporal relations between process steps, and resource requirements for each step.

noOverlap	☐ false
name	*Process 0*

3.1.4 ProcessStep

A *ProcessStep* describes describes one step of a manufacturing process. The duration values are used to compute how long a tasks will take, depending on the quantity produced for one order.

process	*Process 0*
name	*PS0/0*
durationFixed	*10*
durationPerUnit	*10*

3.1.5 ProcessSequence

The *ProcessSequence* describes the temporal relations between two process steps. We currently only allow the most basic *SequenceType* `EndBeforeStart`.

offset	0
before	PS0/0
name	Seq 0/1
after	PS0/1
sequenceType	EndBeforeStart

The possible values for the `sequenceType` field are given in Table 2.

Table 2: Possible Values for SequenceType

Value
EndBeforeStart
StartBeforeStart
NoWait
Blocking

3.1.6 DisjunctiveResource

A *DisjunctiveResource* can work on one task at a time. Task run without interruption, the resource is *non-preemptive*.

name	DR0

3.1.7 ResourceNeed

Depending on the process step, we may be able to use only one or one of multiple possible machines. There is one entry for each possible machine. We will add machine preferences and machine dependent times later on. Every process step needs to be compatible with at least one disjunctive resource.

disjunctiveResource	DR0
name	RN0/0/DR0
processStep	PS0/0

3.1.8 Cumulative Resource

A *CumulativeResource* can perform multiple tasks at the same time, as long as a given resource profile is not exceeded. A task may consume more than one unit of resource at a time.

name	CR0

3.1.9 Cumulative Need

CumulativeNeed describes how many values of a cumulative resource a process step requires. Not all process steps will use a specific cumulative resource, while come tasks will consume resources on more than one cumulative resource.

cumulativeResource	*CR0*
name	*CN0/0/CR0*
processStep	*PS0/0*
demand	1

3.1.10 Cumulative Profile

CumulativeProfile describes how the available resources of a cumulative resource change over time. The value is valid from the date from to the next profile date, or until the end of the project, if there is no further profile entry.

fromDate	*22/1/2025 11:01*
cumulativeResource	*CR0*
name	*CP00*
from	*0*
capacity	5

3.2 Schedule

The Schedule data describe the activities that need to be performed, without without assigning resources or start and end times.

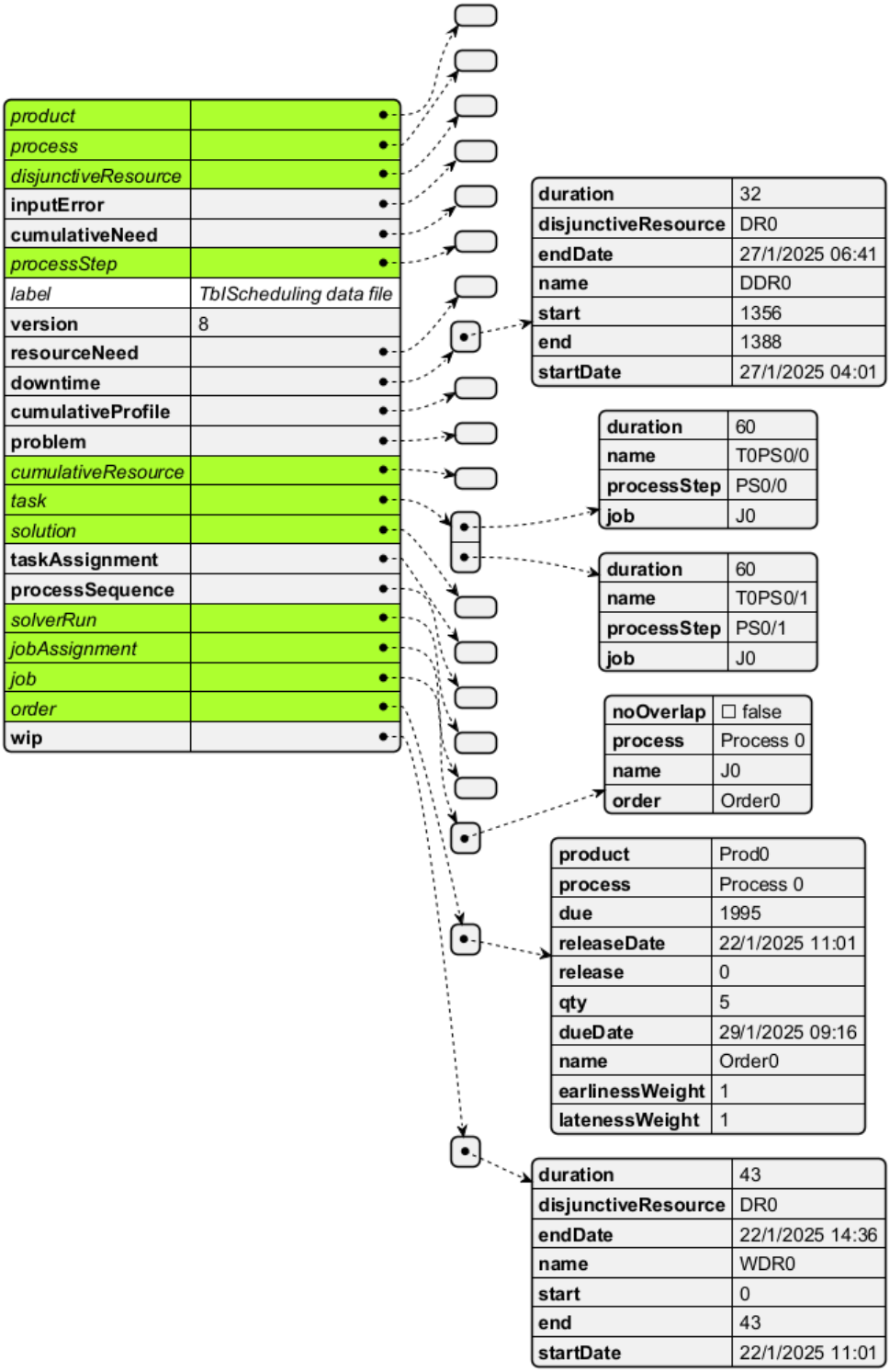

Figure 3: Schedule Specific Data Overview

3.2.1 Order

An *Order* describes which products we are going to make in which quantity, and when the products needs to be ready. We can also express optional weights which indicate the importance of the due date of the order. The order also defines which process we are using for production, in case we specific alternative processes for a product.

product	*Prod0*
process	*Process 0*
due	*1995*
releaseDate	*22/1/2025 11:01*
release	*0*
qty	5
dueDate	*29/1/2025 09:16*
name	*Order0*
earlinessWeight	*1*
latenessWeight	*1*

Jobs and tasks can be reconstructed from the orders and the general problem description.

3.2.2 Job

A *Job* describes the work needed for one order. The job will consist of one more more tasks.

noOverlap	☐ false
process	*Process 0*
name	*J0*
order	*Order0*

3.2.3 Task

A *Task* describes the work needed to perform one process step for an order.

duration	*60*
name	*T0PS0/0*
processStep	*PS0/0*
job	*J0*

3.2.4 WiP

Work in progress (*WiP*) defines a period at the start of the schedule when a disjunctive resource is still busy with work assigned in a previous schedule. No task can be scheduled on the machine before the end of the work in progress. The values for the work in progress will normally come from the actual facility, not from the stored previous schedule.

duration	43
disjunctiveResource	DR0
endDate	22/1/2025 14:36
name	WDR0
start	0
end	43
startDate	22/1/2025 11:01

3.2.5 Downtime

A *Downtime* is a planned shutdown of a disjunctive resource with a defined `start` and `end` date. While the machine is shut down, no task can be performed on the resource.

duration	32
disjunctiveResource	DR0
endDate	27/1/2025 06:41
name	DDR0
start	1356
end	1388
startDate	27/1/2025 04:01

There are variations of a shutdown we might discuss later on, where an unfinished task may stretch over a downtime to complete once the machine becomes available again, but this is not considered here.

4 Results

This section describes the different result types that are stored in the application files. Note that not all object attributes are stored in the data files, some of them are only used when running the solver.

4.1 Solution Data

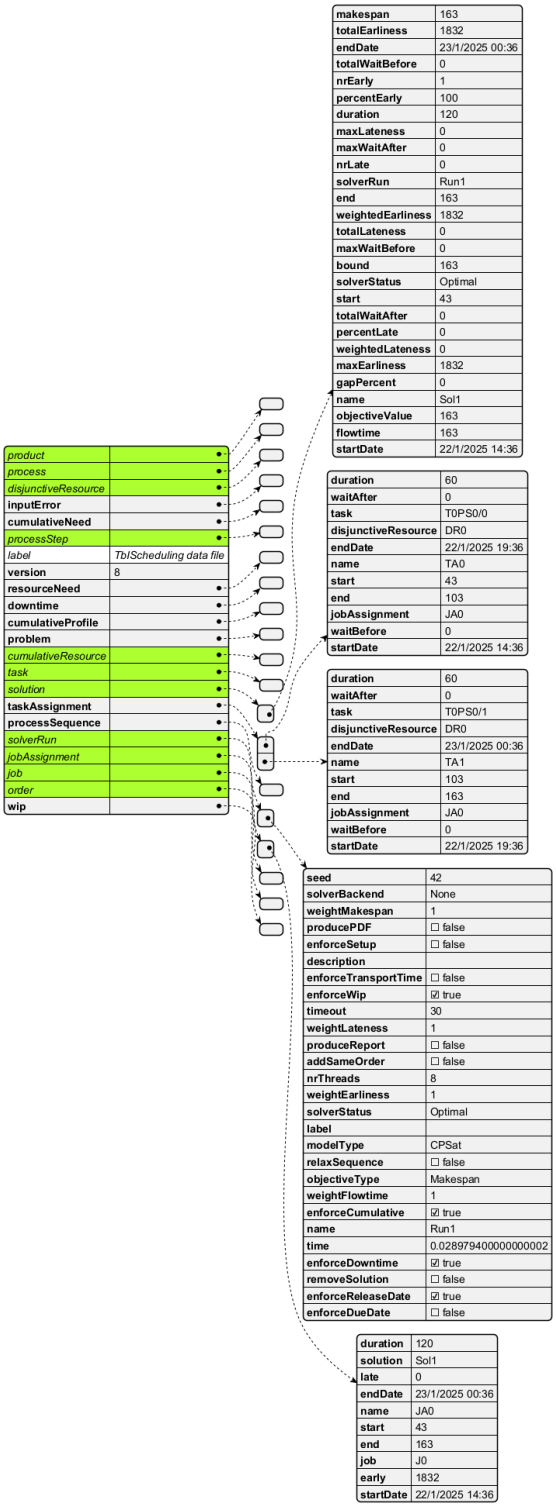

Figure 4: JSON Output Format Overview

4.1.1 SolverRun

A *SolverRun* object is created before the solver is run. It contains the settings which control which model and solver backend is executed together with other parameters. When asking for a solution, the *SolverStatus* field is set to `ToRun`, after running the solver it contains the solver status returned by the backend. The `time` value is also set after the solver has finished.

nrThreads	2
weightEarliness	1
seed	42
solverBackend	*None*
weightMakespan	1
producePDF	☐ false
solverStatus	*Optimal*
description	
label	
modelType	*CPO*
enforceWip	☑ true
objectiveType	*Makespan*
timeout	*30*
weightFlowtime	1
enforceCumulative	☑ true
name	*Run1*
weightLateness	1
time	*0.142*
produceReport	☐ false
enforceDowntime	☑ true
removeSolution	☐ false
enforceReleaseDate	☑ true
enforceDueDate	☐ false

The possible values for the `solverBackend` field are given in Table 3.

Table 3: Possible Values for SolverBackend

Value
None
Chuffed
Gecode
CPSat
Cplex

The possible values for the `solverStatus` field are given in Table 4.
The possible values for the `modelType` field are given in Table 5.
The possible values for the `objectiveType` field are given in Table 6.
The *SolverStatus* values `Error`, `Infeasible`, and `Unknown` indicate that the solver did not find a solution at all. Otherwise, the best solution found is returned as well. The value `ToRun` indicates that the solver was not run at all,

Table 4: Possible Values for SolverStatus

Value
ToRun
Optimal
Solution
Infeasible
Unknown
Error

Table 5: Possible Values for ModelType

Value
CPO
CPSat
MiniZincDiffn
MiniZincTask
REST
Batch

Table 6: Possible Values for ObjectiveType

Value
Makespan
Flowtime
TotalLateness
MaxLateness
WeightedLateness
TotalEarliness
MaxEarliness
WeightedEarliness
OnTime
Hybrid
Projection

due to major input errors.

4.1.2 Solution

There is a *Solution* object created every time the scheduling solver did find a solution. The object contains the objective value, other cost values and computed key performance indicators. The solver status indicate whether an optimal solution `Optimal` or only a feasible solution `Solution` was found.

makespan	*163*
totalEarliness	*1832*
endDate	*23/1/2025 00:36*
totalWaitBefore	*0*
nrEarly	1
percentEarly	100
duration	*120*
maxLateness	*0*
maxWaitAfter	*0*
nrLate	0
solverRun	*Run1*
end	*163*
weightedEarliness	1832
totalLateness	*0*
maxWaitBefore	*0*
bound	163
solverStatus	*Optimal*
start	*43*
totalWaitAfter	*0*
percentLate	0
weightedLateness	0
maxEarliness	*1832*
gapPercent	0
name	*Sol1*
objectiveValue	163
flowtime	*163*
startDate	*22/1/2025 14:36*

The possible values for the `solverStatus` field are given in Table 4.

4.1.3 JobAssignment

There is one *JobAssignment* object created for every *Job* and every *Solution* that is found. This object contains the resulting schedule for the job, i.e. its `start`, `end`, and `duration` as wells as `earliness` and `lateness` costs.

duration	120
solution	Sol1
late	0
endDate	23/1/2025 00:36
name	JA0
start	43
end	163
job	J0
early	1832
startDate	22/1/2025 14:36

4.1.4 TaskAssignment

There is one *TaskAssignment* object created for every *Task* and every *Solution* that is found. This object contains the result values for the task in the given solution. These are `start`, `end`, and `duration` values, as well as the `disjunctiveResource` on which the task is run.

duration	60
waitAfter	0
task	T0PS0/0
disjunctiveResource	DR0
endDate	22/1/2025 19:36
name	TA0
start	43
end	103
jobAssignment	JA0
waitBefore	0
startDate	22/1/2025 14:36

4.2 Error Information

4.2.1 Input Error

When loading a data file, the system checks the data for internal consistencies, and may produce *InputError* objects of different *Severity* levels. If major errors are found, the system will not run the solver, but return the input data with the description of the input errors added.

severity	Fatal
item	order1
field	release
name	Sample
description	Release date is after due date
classDesc	Order
value	75

The possible values for the `severity` field are given in Table 7.

Table 7: Possible Values for Severity

Value
Fatal
Critical
Major
Minor