json2ttb User Manual

For version 1.0.0

Contents

Introduction

json2ttb is a complementary program for Railway Operation Simulator to help in the creation of timetables.

Features

What json2ttb can do.	What json2ttb cannot do. (yet)
Creates timetables for ROS from user data.	No current support for joins/splits or shuttles.
Allows for "odd" repeats of services.	Does not simplify writing individual services.
Create templates of service data and use them for several services.	No current support for the ROS repeats (e.g. R;30;2;2)
Change the reference and description of single repeats.	
In most cases, quicker than writing timetables in ROS.	

Using the Program

Prerequisites

1. Ensure you have Java installed

json2ttb uses Java. There is a good chance you already have this installed, if not, you can find more details about how to install here.

2. A text editor for writing JSON files.

Notepad (or similar) will work fine for this. However, you may find it useful to have a text editor with *code* support which should insert closing brackets and such. I use Atom by GitHub, but there's many out there.

3. The latest jar of json2ttb.

You can download the latest version from the json2ttb GitHub repository here. In the latest release, select the assets dropdown, then choose the .jar file.

Running the Program

In a command prompt window (or PowerShell), enter the following command and run it with relevant details filled in.

```
java -jar <path to jar> <path to json>
```

The ttb file will be created in the same directory as the json file.

Guide to the JSON Structure

Read this section carefully as the program will not work if the format is incorrect in your .json file.

Key Terminology

- Service: The list of instructions and data that a train will be associated with.
- Instance [of a service]: The actual train which will follow a service (e.g. each repeat of a service.)
- Timetable: The final product, containing all services which can be used with ROS.

Example JSON files

Below are a few example files which match the json format, feel free to use these to get a hang of the structure.

- json2ttb Testing Example
- South London and Thameslink
- Llandudno Junction
- Leeds and Bradford (WIP)

Starting the File

To start, the file must contain curly brackets (an object) at the top and bottom.

```
{

}
```

To be contained in here will be the start time of the timetable, and an array which will contain all of the services.

```
{
  "startTime":"07:00",
  "services": [

  ]
}
```

Service Information

To create a new service, we start with an object inside the `services` array. This object **requires** the following information:

```
{
  "startTime":"07:00",
  "services": [
    {
      "ref": "1A00",
      "description": "Z to C",
      "startSpeed": 20,
      "maxSpeed": 121,
      "mass": 200,
      "maxBrake": 149,
      "power": 1500,
      "increment": 1
    }
  ]
}
```

We can replace `maxSpeed`, `mass`, `maxBrake` and `power` with a `dataTemplate`. Find more information here.

Within the service object, we also need to include `events` and `times` arrays:

```
{
  "startTime":"07:00",
  "services": [
    {
      "ref": "1A00",
      "description": "A to B",
      "startSpeed": 20,
      "maxSpeed": 121,
      "mass": 200,
      "maxBrake": 149,
      "power": 1500,
      "increment": 1,
      "events": [

      ],
      "times": [

      ]
    }
  ]
}
```

Adding Events and Times

Once we have a service, we can add any number of events within the `events` array. The services that we create use the exact same syntax as those in ROS timetables. They must be as strings separated by commas.

There are two ways to achieve timings for instances, both will work and can be used interchangeably within a single timetable.

Method A: `events` dictate the *actual* times.

This is probably considered the simpler method. Here we use the actual times for the events.

```
      ...
      "events": [
```

```
      "07:00;Snt;6-2 5-2",
      "07:02;07:02;A",
      "07:05;07:05;B",
      "07:07;Fer;15-2"
    ],
    ...
```

The `times` will contain the time in relation to `00:00` that these services will occur. In the below example, we want our service to occur every 30 mins for 4 instances.

```
    ...
    "times": [
      "00:00", "00:30", "01:00", "01:30"
    ]
    ...
```

This will mean that, in our final timetable we will have 4 instances.
(Remember that `"ref":"1A00"` and `"increment":"1"`)

- 1A00, which enters at 07:00
- 1A01, which enters at 07:30
- 1A02, which enters at 08:00
- 1A03, which enters at 08:30

We are not required to pick times every x minutes. We can mix up the times and have some very odd service patterns.

```
    ...
    "times": [
      "00:00", "00:24", "00:38", "01:00"
    ]
    ...
```

In this example, we will have 4 instances:

- 1A00, which enters at 07:00
- 1A01, which enters at 07:24
- 1A02, which enters at 07:38
- 1A03, which enters at 08:00

Method B: `times` dictate the *actual* times.

This method may be more useful if you're copying directly from an external timetable. Here we can create events which all have times before and after some *pivot* time, which could be a significant departure time. You could achieve this like so:

```
    ...
    "events": [
      "23:58;Snt;6-2 5-2",
      "00:00;00:00;A",
      "00:03;00:03;B",
      "00:05;Fer;15-2"
    ],
    ...
```

This example has the same time differences as the previous examples, however, we have now chosen the *pivot* to be the departure time from station A. Now in `times` , we can specify the departure times from station A, and all other times will be changed accordingly.

```
...
"times": [
  "07:02", "07:26", "07:40", "08:02"
]
...
```

You could, for example, extract departure times from a single principle station from an external timetable, and import these into your json file. This is especially helpful if the services do not have a regular departure time pattern.

Change Information per Instance

For example, we may have a one-off service that extends beyond the usual destination station. For this, we can change the `description` for an individual instance in the `times` array.

To do this, we change one element of the array to be an object, which has a `time` key which contains the time as before, as well as a `description` key with the updated description.

```
...
"times": [
  "07:02", {"time":"07:26", "description":"Z to D"}, "07:40", "08:02"
]
...
```

We can also do a similar thing with the service references. Please note, however, the `increment` will still apply the changed ref, so the next instance will have an increment twice over.

```
...
"times": [
  "07:02", {"time":"07:26", "ref":"2D01"}, "07:40", "08:02"
]
...
```

This extract from the South London and Thameslink timetable shows how these can be used together to create instances with data from RealTime Trains.

```
"times": [
  {"time":"06:11","ref":"1P02","description":"Ramsgate to London Victoria"},
  {"time":"06:26","ref":"2A06","description":"Swanley to London Victoria"},
  {"time":"06:41","ref":"1P06","description":"Dover Priory to London Victoria"},
  {"time":"06:43","ref":"2A08","description":"Ashford International to London Victoria"},
  ...
]
```

Data Templates

We can generalise `maxSpeed`, `mass`, `maxBrake` and `power` for each service using a data template, which hold all of this information. This is done by using a `dataTemplate` instead of the above keys. This will contain a keyword that refers to a given template.

```
{
  "startTime":"07:00",
  "services": [
    {
      "ref": "1A00",
      "description": "A to B",
```

```
      "startSpeed": 20,
      "dataTemplate": "TestData",
      "increment": 1,
      "events": [
        ...
      ],
      "times": [
        ...
      ]
    }
  ]
}
```

We can create a **custom template** by adding a new array between `startTime` and `services`, which contains an object with keys for `keyword`, `maxSpeed`, `mass`, `maxBrake` and `power`.

```
{
    "startTime": "07:00",
    "dataTemplates": [
        {
            "keyword":"TestData",
            "maxSpeed": 150,
            "mass": 100,
            "maxBrake": 20,
            "power": 25
        }
    ],
    "services": [
      ...
    ]
}
```

There are also a large number of pre-defined templates for a large number of UK trains from Mark's great stock data spreadsheet. When using them as a data template, the `keyword` is generally of the form:

`C<class>_<carriages/cars>` or `C<class>_<subclass>_<carriages/cars>`

For example, a 2-car Class 150/1 is `C150_1_2`.

A list of pre-defined templates can be found in Appendix A.

Get Help

If you would like to get help when using this program, please get in touch on the ROS Discord Server or contact me (DanG#4669) directly on Discord.

You can also leave an issue on the json2ttb GitHub repository here, and I will get back to you.

Appendix A: List of Pre-Defined Data Templates

EMUs are listed first, then DMU/DEMUs.

Classes	Keyword
Class 313/0	C313_0_3
Class 313/1	C313_1_3
Class 313/2	C313_2_3

Classes	Keyword
Class 314/2	C314_2_3
Class 315/8	C315_8_3
Class 317/1	C317_1_4
Class 317/5	C317_5_4
Class 317/6	C317_6_4
Class 317/7	C317_7_4
Class 317/8	C317_8_4
Class 318	C318_3
Class 319	C319_4
Class 320/3	C320_3_3
Class 321/3	C321_3_4
Class 321/4	C321_4_4
Class 322	C322_4
Class 323	C323_3
Class 325	C325_4
Class 331/0	C331_0_3
Class 331/1	C331_1_4
Class 332	C332_4
Class 333	C333_4
Class 334	C334_3
Class 345	C345_9
Class 350/1	C350_1_4
Class 350/2	C350_2_4
Class 350/3	C350_3_4
Class 350/4	C350_4_4
Class 357	C357_4
Class 360/1	C360_1_4
Class 360/5	C360_5_5
Class 365	C365_4
Class 373	C373_16
Class 373	C373_20

Classes	Keyword
Class 374	C374_16
Class 375	C375_3
Class 375	C375_4
Class 376	C376_5
Class 377/1	C377_1_4
Class 377/2	C377_2_4
Class 377/3	C377_3_3
Class 377/4	C377_4_4
Class 377/5	C377_5_4
Class 377/6	C377_6_5
Class 377/7	C377_7_5
Class 378/0	C378_0_3
Class 378/0	C378_0_4
Class 378/1	C378_1_5
Class 378/2	C378_2_7
Class 379	C379_4
Class 380/0	C380_0_3
Class 380/1	C380_1_4
Class 385	C385_3
Class 385	C385_4
Class 387/1	C387_1_4
Class 387/2	C387_2_4
Class 387/3	C387_3_4
Class 390/0	C390_0_9
Class 390/1	C390_1_11
Class 395	C395_6
Class 399	C399_3
Class 442	C442_5
Class 444	C444_5
Class 450	C450_4
Class 455	C455_4

Classes	Keyword
Class 456	C456_2
Class 458/0	C458_0_4
Class 458/5	C458_5_5
Class 460	C460_8
Class 465	C465_4
Class 466	C466_2
Class 482	C482_2
Class 483	C483_3
Class 507	C507_3
Class 508	C508_3
Class 700	C700_8
Class 707	C707_5
Class 710/1	C710_1_4
Class 710/2	C710_2_4
Class 717	C717_6
Class 801	C801_5
Class 801	C801_9
Class 139	C139_1
Class 142	C142_2
Class 143/0	C143_0_2
Class 143/3	C143_3_2
Class 143/6	C143_6_2
Class 144	C144_2
Class 144	C144_3
Class 150/0	C150_0_3
Class 150/1	C150_1_2
Class 150/2	C150_2_2
Class 150/9	C150_9_3
Class 153	C153_1
Class 155	C155_2

Classes	Keyword
Class 156	C156_2
Class 158	C158_2
Class 158	C158_3
Class 159	C159_3
Class 165/0	C165_0_2
Class 165/0	C165_0_3
Class 165/1	C165_1_2
Class 165/1	C165_1_3
Class 166	C166_3
Class 168	C168_3
Class 168	C168_4
Class 170/1	C170_1_2
Class 170/1	C170_1_3
Class 170/2	C170_2_2
Class 170/2	C170_2_3
Class 170/3	C170_3_3
Class 170/4	C170_4_3
Class 170/5	C170_5_2
Class 170/6	C170_6_3
Class 170/6	C170_6_3
Class 171/7	C171_7_2
Class 171/8	C171_8_4
Class 172/0	C172_0_2
Class 172/1	C172_1_2
Class 172/2	C172_2_2
Class 172/3	C172_3_3
Class 175/0	C175_0_2
Class 175/1	C175_1_3
Class 180 *Zephyr*	C180_5
Class 185 *Pennine*	C185_3
Class 220 *Voyager*	C220_4

Classes	Keyword
Class 220 *Voyager*	C220_4
Class 221 *Super Voyager*	C221_4
Class 221 *Super Voyager*	C221_5
Class 222/0 *Meridian*	C222_0_5
Class 222/0 *Meridian*	C222_0_7
Class 222/0 *Meridian*	C222_0_4
Class 195/0	C195_0_2
Class 195/1	C195_1_3
Class 230	C230_2
Class 230	C230_3
Class 755/3	C755_3_3
Class 755/4	C755_4_4
Class 756/3	C756_3_3
Class 756/4	C756_4_4
Class 800/1	C800_1_5
Class 800/2	C800_2_9
Class 800/1	C800_1_9
Class 802	C802_5
Class 68 +MK5+DVT *Nova*	C68_5
Class 67 +MK3+DVT	C67_4
Class 67 +MK3+DVT	C67_6
Class 43 HST *Castle*	C43_4
Class 43 HST (2+6, B)	C43_6
Class 43 HST (2+7)	C43_7
Class 43 HST (2+8)	C43_8
Class 43 HST (2+9)	C43_9