

---

# TreeTime User Manual

*Release 2024.3*

Jacob Kanev

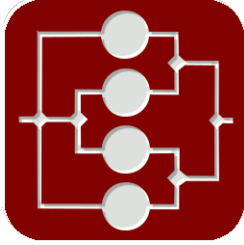
Jul 02, 2024



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Concept . . . . .	3
1.2	Basic Use . . . . .	5
1.3	Data Files . . . . .	7
1.4	Installation . . . . .	7
<b>2</b>	<b>Data Format</b>	<b>9</b>
2.1	Global Structure . . . . .	9
2.2	Tree Definition . . . . .	10
2.3	Data Item Definition . . . . .	11
2.4	The Data Pool . . . . .	11
<b>3</b>	<b>Data Fields</b>	<b>13</b>
3.1	string . . . . .	13
3.2	text . . . . .	13
3.3	longtext . . . . .	13
3.4	url . . . . .	14
3.5	integer . . . . .	14
3.6	timer . . . . .	14
<b>4</b>	<b>Tree Fields</b>	<b>15</b>
4.1	General Syntax . . . . .	15
4.2	string . . . . .	16
4.3	url . . . . .	16
4.4	text . . . . .	16
4.5	sum . . . . .	16
4.6	set . . . . .	16
4.7	sum-time . . . . .	17
4.8	difference . . . . .	17
4.9	difference-time . . . . .	17
4.10	mean . . . . .	17
4.11	mean-percent . . . . .	18
4.12	min . . . . .	18
4.13	max . . . . .	18
4.14	min-string . . . . .	18
4.15	max-string . . . . .	18
4.16	ratio . . . . .	18
4.17	ratio-percent . . . . .	19
4.18	node-name . . . . .	19
4.19	node-path . . . . .	19

<b>5</b>	<b>History and Road Map</b>	<b>21</b>
5.1	Past . . . . .	21
5.2	2024 . . . . .	23
5.3	Present . . . . .	24
5.4	Future . . . . .	24



*TreeTime* is a general data organisation, management and analysis tool using linked trees instead of flat lists of tables. A tree is a hierarchical structure that arranges your data into units and sub-units. Mathematical functions (sum, difference, mean, ratio) can be calculated recursively. Linked trees are distinct trees that share data between them. In *TreeTime*, a data object is part of several trees at the same time. *TreeTime* is a time planner, a to-do list manager, a test report tool, a project planner, a family ancestry editor, a mind-mapping tool, and similar.



## INTRODUCTION

### 1.1 Concept

#### 1.1.1 What is a Tree?

A “tree” is a data structure, much like a table or a list. A tree sorts information hierarchically into boxes and sub-boxes and sub-sub-boxes.

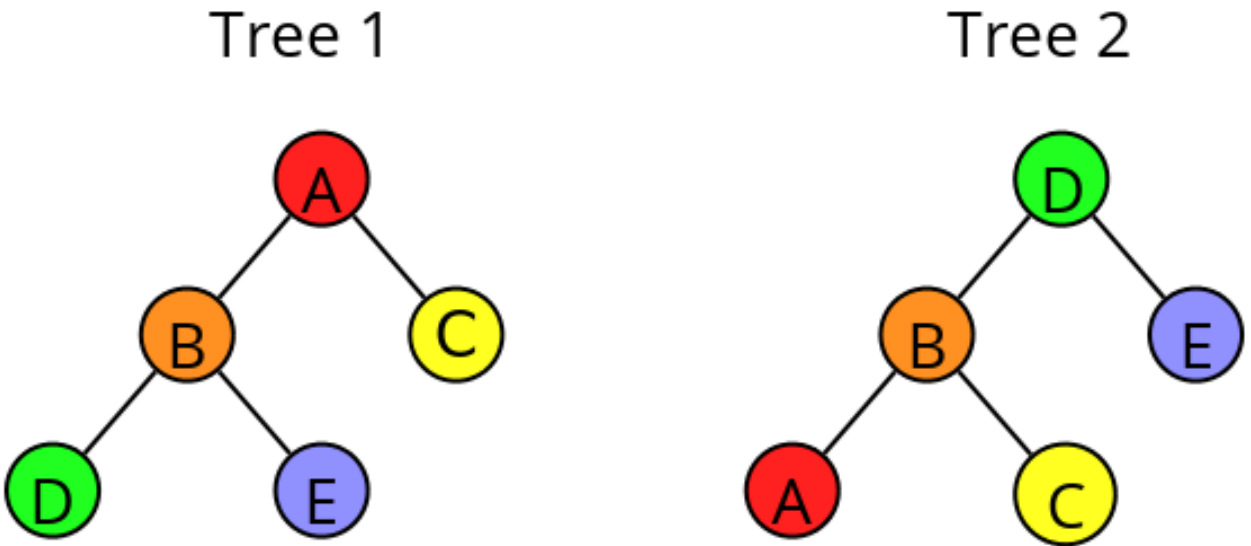
If you want to organise your work tasks you could sort them into work packages, that are part of projects, that are part of products. If you plan a larger project, you can sort all tasks by responsible persons, who are part of teams, that are part of departments, that are part of branches. You can also make a time plan, where a year consists of quarters, that consist of weeks, that contain a number of tasks. You can have an address book where you have a hierarchy of friends / colleagues / acquaintances, or you can sort knowledge about animals into kingdom / class / family / species.

The nice thing about trees is that you can define mathematical functions on them. Planned hours can be summed up per work package and project, or per person and team, or per week and month. A mean priority can be shown per work package and project.

The concept of hierarchical categorisation can be applied to all sorts of data and will feel a lot more natural and easier to use than organising the same data in spread sheets.

#### 1.1.2 What are linked Trees?

The core concept of *TreeTime* are linked trees. Linked trees are separate trees that share the same data. One piece of information (a *node*) can be in several trees at the same time, but in different place of the tree. As a single tree is a way of sorting information, different linked trees sort the same data in different ways.



In *Tree 1*, Node *E* is right at the bottom, as a child of *B* and a grandchild of *A*. In *Tree 2* it is a child of *D*.

In *TreeTime*, a *node* or *item* can hold different information like text, numbers, dates, internet links. These are saved in the *item's fields*.

Here we have a field we call “value”. Each node in all trees has a value field that can hold a number (like a cell in a spread sheet). The node *A* has the value=1, *B*=2, etc. In addition we have a field we call “Sum”. Its content is calculated automatically, summing up the item’s own value plus the values of all children. In *TreeTime*, looking at item *E* and *Tree 1* this looks like this:

E

Tree 1

A

B

>

Tree 2

D

>

value 5

Tree 1

Tree 2

	Value	Sum
√ A	1	15
√ B	2	11
D	4	4
E	5	5
C	3	3

Clicking on the other tab shows the second tree while the same items stays selected:

E

Tree 1

A

B

>

Tree 2

D

>

value 5

Tree 1

Tree 2

	Value	Sum
√ D	4	15
√ B	2	6
A	1	1
C	3	3
E	5	5



Note how the values are summed up the branches. Apart from sums, *TreeTime* also offers means, ratios, or differences, using different combinations of parent, child, or sibling fields.

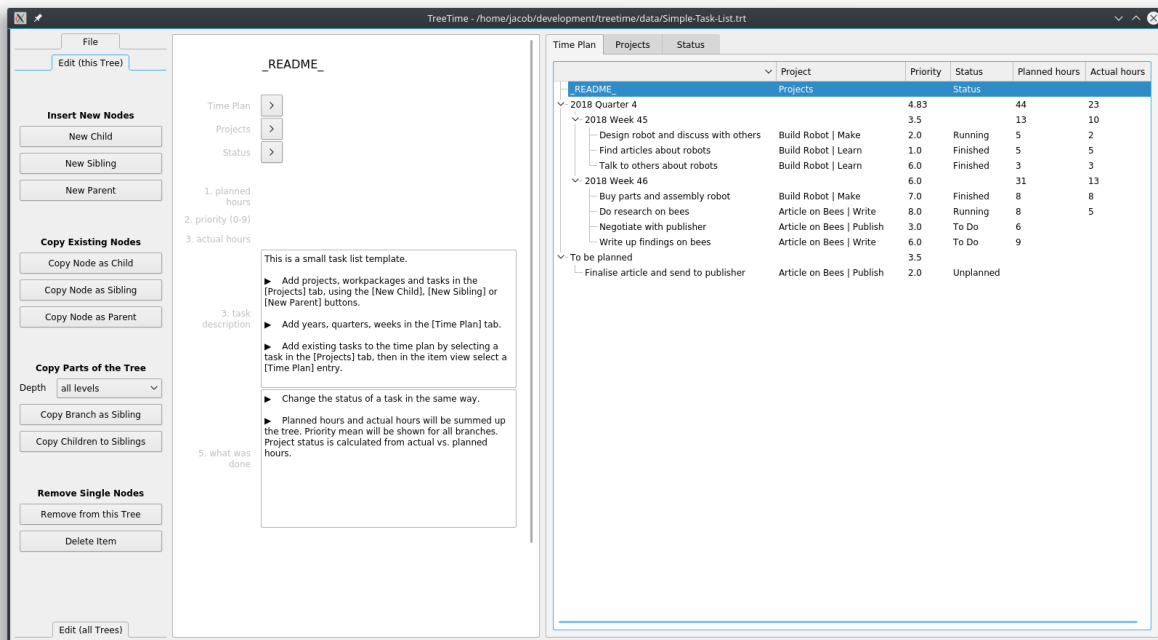
Linked trees are a natural and powerful way to structure data. If you, for instance, organise information about animals, you might want to see the animal's taxonomy (kingdom/class/family/species), but also their habitat (continent/country/area), and switch between both views. If you organise tasks, you could switch between a year/quarter/week/day breakdown, a company/department/team/person tree, and a product/project/package/task overview.

In *TreeTime*, the structure of your data (whether you store priority, hours, and a description for a task, or expected life span, habitat and number of legs for an animal), the trees themselves, and the calculated values within the trees are completely user defined. Data is stored in text files, changes are saved on the fly, and when opening *TreeTime*, the software is automatically connected to the last used file.

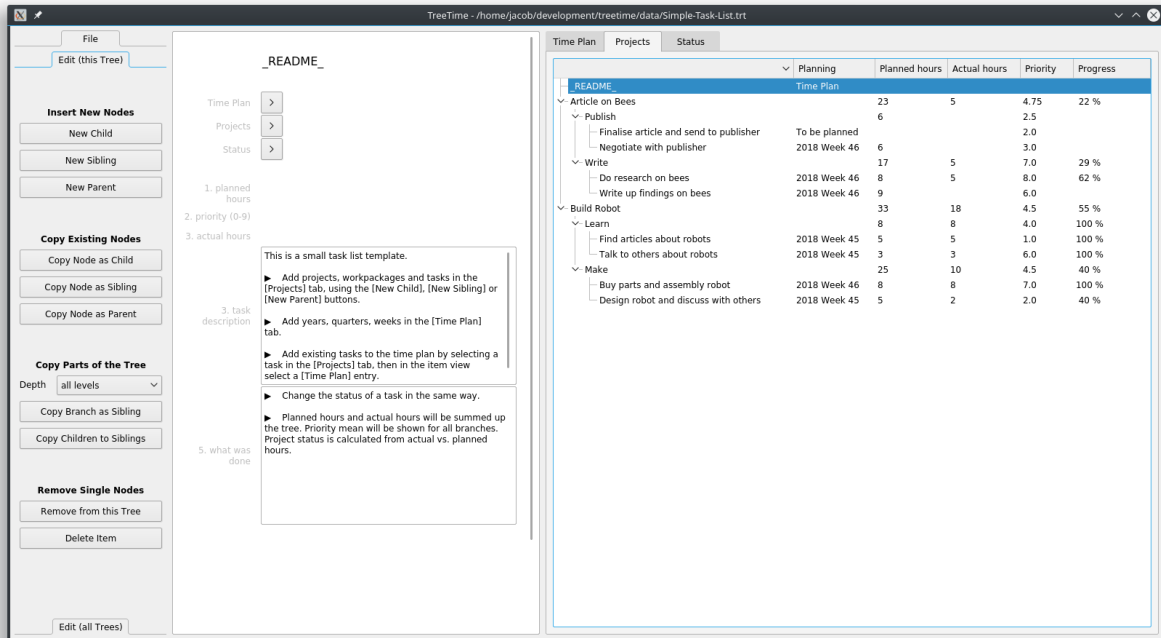
## 1.2 Basic Use

Start the software (see 'Execute' from the section [Installation](#id-installation)). In the main dialog, go to "File" / "New from Template", select "Simple-Task-List.trt" and in the next dialog give a file name for the new file. An example file with a simple project task list structure will open.

The GUI consists of three parts: - A button box on the left. Execute tree structure operations from here. - An editing grid in the middle, showing the contents of the selected data item. Edit single data items here. - A tab view with trees spanning the center-right. View and analyse your data here.



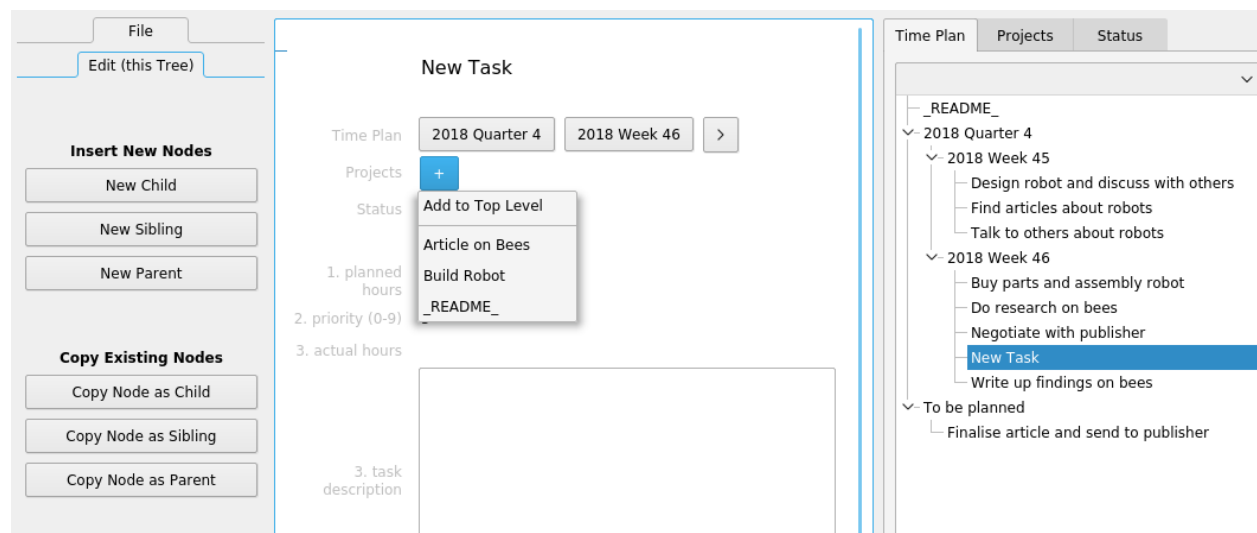
Access each single tree by clicking a tab on the main tree-view widget (the picture above shows the tree *Time Plan*, the picture below the tree *Projects*).



Branches and children can be sorted, branches can be folded and unfolded. Data content is shown via analytic fields that are defined per tree. In the example project you will see a sum, a percentage, and text display.

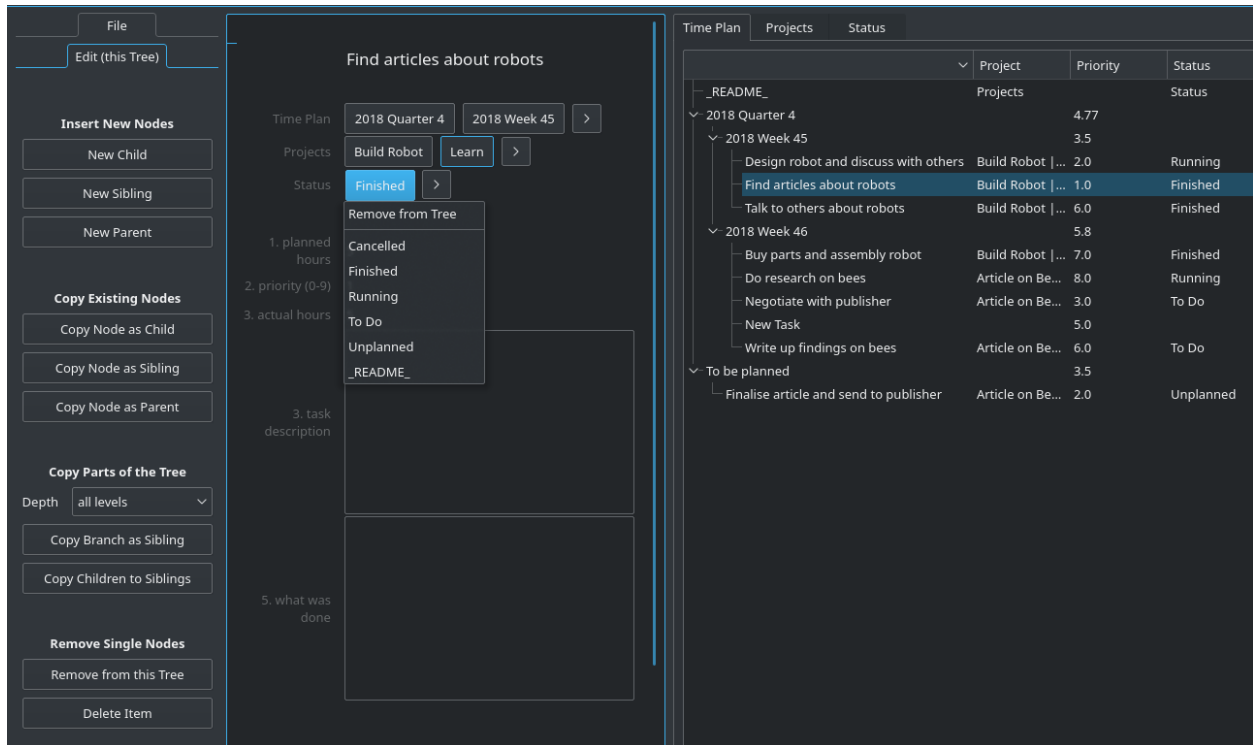
Add, move and remove single nodes and complete branches by using the buttons on the left. Change the name of a node by selecting the node and editing the name in the top of the edit grid in the middle. Change all other values (numbers or text) by clicking into the field and start typing.

The parents of an item are listed underneath the item name. Each tree has a separate line. Change the position of a node within a tree by clicking on any of the parent buttons.



In this example a new node in the tree *Time Plan* has just been created, and is now added to the tree *Projects*.

*TreeTime* lets you select different themes and will try to use the default colours that are defined with your operating system.



## 1.3 Data Files

The data in TreeTime is stored in a plain text file, marked with a 'trt' file ending ('trt' for 'TreeTime').

The button 'Load File' will open an existing 'trt' file. After this, all changes are written to that file. There is no 'Save' button, changes are written to the file immediately.

The button 'New From Template' opens an existing data file, creates a copy, and saves this copy. A data file can be created by copying the currently open file. All write operations will be performed on that copy. This is to create a new file from a basically empty 'trt' file that has a pre-defined data structure.

A data file can be created by saving the currently open file to a copy. The button 'Save As' saves the current state. All write operations will be on the new file.

## 1.4 Installation

### 1.4.1 Using pre-compiled Binaries

- Windows, Linux: Go to [\[github.com/jkanev/treetime/releases/tag/2024.3\]](https://github.com/jkanev/treetime/releases/tag/2024.3) and download a zipped package for Windows 10, 64 bit, or for Linux 64 bit from there. Unzip it into your program directory and run *TreeTime* or *TreeTime.exe* from the new folder. Unzip the data package too. Add the program folder to your path.

Executable bundles have been created with pyinstaller ([www.pyinstaller.org](http://www.pyinstaller.org)) (<http://www.pyinstaller.org>)).

- Mac: Mac users please use the Python code (see below). There is no executable for Mac. (If anybody can help building an executable for other platforms I'd be delighted.)

### 1.4.2 Using a PyPi package in Python

1. If you don't have it yet, install python3
2. **Install PyQt6** – on an elevated command prompt (Windows), or on the standard command line (Mac, Linux), type:  
*pip install PyQt6*
3. **Install TreeTime** – on an elevated command prompt (Windows), or on the standard command line (Mac, Linux), type:  
*pip install treetime*

### 1.4.3 Using script code with Python

1. If you don't have it yet, install python3
2. Install PyQt6 – on an elevated command prompt (Windows), or on the standard command line (Mac, Linux), type: *pip install PyQt6*
3. Download this project from GitHub as a zip file (<https://github.com/jkanev/treetime/archive/master.zip>) and unzip
4. **Install TreeTime: in the command line, cd into the main directory, then type:**
  - **Linux:**
    - *python3 setup.py build*
    - *sudo python3 setup.py install*
  - **Windows:**
    - *py setup.py build*
    - *py setup.py install*

### 1.4.4 Execute

- Windows: Hit the Windows key and type “TreeTime”, then click the “run command treetime” that comes up.
- Linux, Mac: On the command line, type “TreeTime”. You can also start this any other way your operating system supports. Plus, there's a .desktop file (for KDE and Gnome) in the data directory to create desktop or menu link.

## DATA FORMAT

### 2.1 Global Structure

TreeTime data files are plain text (Unicode/UTF8) and can be edited with any text editor. The global structure consists of three parts: The tree definition, the item definition, and the item pool.

- The tree definition is preceded with the marker `--trees--` followed by a newline. This defines the number and data structure of the trees in the file.
- The data item definition is preceded by the marker `--item-types--` followed by a newline. This defines the data fields of each data item.
- The item pool is preceded by the marker `--item-pool--` followed by a newline. This section contains the actual data.

The file content of the simple example file in the Introduction chapter looks like this:

```
--trees--

tree "Tree 1"
  field "Value"
    field-type "sum"
    own-fields ["value"]
    child-fields []
    sibling-fields []
    parent-fields []
  field "Sum"
    field-type "sum"
    own-fields ["value"]
    child-fields ["Sum"]
    sibling-fields []
    parent-fields []

tree "Tree 2"
  field "Value"
    field-type "sum"
    own-fields ["value"]
    child-fields []
    sibling-fields []
    parent-fields []
  field "Sum"
    field-type "sum"
```

(continues on next page)

(continued from previous page)

```

    own-fields ["value"]
    child-fields ["Sum"]
    sibling-fields []
    parent-fields []

--item-types--

item Node
  fields {"value": {"content": 0, "type": "integer"}}
  trees [[], []]

--item-pool--

item A
  fields {"value": {"content": 1, "type": "integer"}}
  trees [[0], [0, 0, 0]]

item B
  fields {"value": {"content": 2, "type": "integer"}}
  trees [[0, 0], [0, 0]]

item C
  fields {"value": {"content": 3, "type": "integer"}}
  trees [[0, 1], [0, 0, 1]]

item D
  fields {"value": {"content": 4, "type": "integer"}}
  trees [[0, 0, 0], [0]]

item E
  fields {"value": {"content": 5, "type": "integer"}}
  trees [[0, 0, 1], [0, 1]]

```

## 2.2 Tree Definition

A single tree is defined by the name of the tree and a list of tree fields. A node's tree field values are calculated from data fields or tree fields of the node itself, its siblings, parent and children. Each of these are mentioned in the field definition. There are various different field types, some use values in the current tree, some use values from other trees. You can for example display the name of a node's parent in a different tree. Trees are numbered starting with 0. Look at the first tree in the example:

```

tree "Tree 1"
  field "Value"
    field-type "sum"
    own-fields ["value"]
    child-fields []
    sibling-fields []
    parent-fields []
  field "Sum"
    field-type "sum"

```

(continues on next page)

(continued from previous page)

```
own-fields ["value"]
child-fields ["Sum"]
sibling-fields []
parent-fields []
```

The tree itself is called “Tree 1”. It has two tree fields, “Value” and “Sum”. The tree field “Value” is of type “sum”, and it displays anything that is found in the data item field “value”. The tree field “Sum” is also of type “Sum” and for each node it adds everything in the node’s item field “value”, plus all values in the tree field “Sum” of its children.

More about how to define tree fields in the next chapter.

## 2.3 Data Item Definition

Each node in a tree is stored as a “data item”. In the data file a “data item” is stored like this:

```
item A
  fields {"value": {"content": 1, "type": "integer"}}
  trees [[], []]
```

After four spaces indent, there’s the keyword “item” and the name (in this case “A”). This is the name that’s displayed in the heading of the data item pane in the GUI, and as the node name in the tree pane of the GUI. The next line, after an indent of 8 spaces, has the keyword “fields” followed by a json dictionary:

```
"field name 1": {"content": 1, "type": "integer"}, "field name 2": ...
```

In this dictionary, each data field has a sub-dictionary listing its default content, the field type, and possibly some other values (timers have a running/stopped flag and a last-started flag). When a new item/node is created, this default content will be in all data fields. In the example above, a new node will contain one single field called “value” with the content “1”.

For a description of all possible data field types, see the Data Fields chapter.

The last line, “trees ...”, must contain an array of  $N$  empty arrays, where  $N$  is the number of trees in your file. If you have four trees in your tree field, that line must read:

```
trees [[], [], [], []]
```

This makes your field definition available in all trees (and creates an error otherwise).

## 2.4 The Data Pool

The Data Pool is the last of the three sections of the tree file, and in most cases the largest. This is where the actual data is stored. It consists of a list of items in the tree, with a syntax like in the data item definition section:

```
item D
  fields {"value": {"content": 4, "type": "integer"}}
  trees [[0, 0, 0], [0]]

item E
  fields {"value": {"content": 5, "type": "integer"}}
  trees [[0, 0, 1], [0, 1]]
```

The content here is the actual content in the field. The tree structure is stored in the last line:

```
trees [[0, 0, 1], [0, 1]]
```

This is an array of arrays, each of which is a path in the tree. In the example above the node can be found following the path 0-0-1 in the first tree starting at the root node, and 0-1 in the second tree. Children are numbered using fixed indexes, starting at 0. A path of 0-0-1 means: My node is the second child (-1) of the first child (-0-1) of the first child (0-0-1) of the root node in the (first) tree. And in the second tree, the path 0-1 says the node is the second child of the first child of the root.



## DATA FIELDS

Data fields are defined by a dictionary {...} where the field names are the dictionary keys {"name1": ..., "name2": ..., "name3": ...}, and their type as well as their default value are the dictionary values {"name1": {"content": "", "type": "longtext"}, "name2": {"content": "", "type": "url"}, "name3": ...}. The possible types are “string”, “text”, “longtext”, “url”, “integer”, and “timer”.

### 3.1 string

One line of text.:

```
"Name": {"content": "Maria Sibylla Merian", "type": "string"}
```

The field contains strings (small texts). In the GUI, the field will span one line. Text can be entered.

### 3.2 text

Longer text.:

```
"details": {"content": "Please do the following: ", "type": "text"}
```

The field can contain longer text. In the GUI, there are 10 lines and there's a scrollbar for entering longer texts.

### 3.3 longtext

Quite long text.:

```
"details": {"content": "We discussed the following...", "type": "longtext"}
```

Identical to the text field, but in the GUI there are 25 lines and a scrollbar.

## 3.4 url

A URI of any type (file, http, ...):

```
"external link": {"content": "https://tree-time.info", "type": "url"}
```

In the GUI there's a text field and a button saying "Open". Clicking the button will use the the content of the text field and call the open method defined in the operating system (e.g. a content of "<https://tree-time.info>" or "<file:///home/myself/downloads/pass-word.info.html>" would be opened with your default web browser).

## 3.5 integer

A number.:

```
"hours planned": {"content": 4, "type": "integer"}
```

A simple number, can be a floating point number such as -1.23456.

## 3.6 timer

A stop watch counting hours/minutes/seconds.:

```
"hours spent": {"content": 0, "running_since": false, "type": "timer"}
```

In the GUI there will be a "Start" button and the field will contain a number.

Hitting the "Start" button will change the text in the field to "stop watch running", and the text on the button changes to "Stop". The stored item in the fiel changes to: "hours spent": {"content": 1.2000021166666666, "running\_since": "2024-04-17 10:25:03", "type": "timer"}. The actual tree field values will get updated once a second, including all branches and parents, updating all values like ratios and sums. It makes sense to use tree fields like *ratio-time* and *sum-time* to see the value in hh:mm:ss format instead of floating point numbers. The stop watch keeps running even when the file is closed or the computer is shut down.

Hitting the "Stop" button will display the currently summed up value in the field, and the text on the button changes to "Start" again. In the file, the "running" flag is removed: "hours spent": {"content": 1.3, "running\_since": false, "type": "timer"}

Subsequent start/stops on the button will add to the total value.

## TREE FIELDS

### 4.1 General Syntax

Each tree field is a function with a list of input fields. These fields can be either tree fields or data fields. To avoid ambiguities it is good practice to name tree fields starting with a capital letter and data fields with a lower case letter. A tree field is always defined as part of a tree (see previous chapter). The definition states the name, the field type, and the input parameters:

```
field "Name"  
  field-type "type"  
  own-fields [...]  
  child-fields [...]  
  sibling-fields [...]  
  parent-fields [...]
```

The field is started by the line `field "Name"` where “Name” is the name of the field. This will appear as the column heading in the tree list. After this, indented with four spaces, is the field type: `field-type "type"`, where “type” is the type (see next for an overview). After this, the lines `own-fields [...]`, `child-fields [...]`, `sibling-fields [...]`, and `parent-fields [...]` each define a list of field names. These are the input parameters for the function. They are evaluated in the order they are mentioned. A real-world example:

```
field "Progress"  
  field-type "ratio-percent"  
  own-fields ["Spent Hours", "Planned Hours"]  
  child-fields []  
  sibling-fields []  
  parent-fields []
```

The tree field “Progress” is a ratio, defined as  $\text{parameter1} / (\text{parameter2} + \text{parameter3} + \dots)$ . In the tree view it will be displayed as a percentage. It shows the ratio of the tree fields “Spent Hours” / “Planned Hours”.

## 4.2 string

The simple display of the content of one or multiple data fields or tree fields. Syntax:

```
field "Name"
  field-type "string"
  own-fields ["field1"]
  child-fields []
  sibling-fields []
  parent-fields []
```

Result: The values or strings found in the fields *field1*, *field2*, *field3*, ..., put together, in the order they are mentioned.

## 4.3 url

Same as “string”, but in an html export the field is formatted as url link (clickable).

## 4.4 text

Same as “string”, but the exported field has a larger width and can span several lines.

## 4.5 sum

The sum of all input fields. Syntax:

```
field "Name"
  field-type "sum"
  own-fields ["field1", "field2", ...]
  child-fields ["field3", ...]
  sibling-fields [...]
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields. Fields must be integer fields, the result for string fields is not defined.

Result: The value  $field1 + field2 + field3 + \dots$

## 4.6 set

A list of unique occurrences of values of all input fields. Syntax:

```
field "Name"
  field-type "set"
  own-fields ["field1", "field2", ...]
  child-fields ["field3", ...]
  sibling-fields [...]
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields.

Result: A list like *value1*, *value2*, *value3*, *value4*, where each value is the value of at least on input field and each value is listed only once, sorted alphabetically.

## 4.7 sum-time

Same as “sum”, but will show the result as hour format, e.g. the value *1.5* will be displayed and exported as *1:30:00*.

## 4.8 difference

Difference of numbers. Syntax:

```
field "Name"
  field-type "difference"
  own-fields ["field1", "field2", ...]
  child-fields ["field3", ...]
  sibling-fields [...]
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields.

Result: The value *field1* - (*field2* + *field3* + ...), in the order they are mentioned.

## 4.9 difference-time

Same as “difference”, but will show the result as hour format, e.g. the value *1.5* will be displayed and exported as *1:30:00*.

## 4.10 mean

The statistical mean of all input fields. Syntax:

```
field "Name"
  field-type "mean"
  own-fields ["field1", "field2", ...]
  child-fields ["field3", ...]
  sibling-fields [...]
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields.

Result: The value  $(field1 + field2 + field3 + \dots) / N$ , where *N* is the number of fields.

## 4.11 mean-percent

Same as “mean”, but will show the result as a percentage, e.g. the value *0.75* will show as *75 %*.

## 4.12 min

The minimum. Syntax:

```
field "Name"  
  field-type "min"  
  own-fields ["field1", "field2", ...]  
  child-fields ["field3", ...]  
  sibling-fields [...]  
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields.

Result: The minimum value *min(field1, field2, field3, ...)*. This can only be for numbers. If you want to find the minimum of texts, use *min-string*.

## 4.13 max

The Maximum. Same as *min*, but displays the maximum.

## 4.14 min-string

The smallest of a list of strings. Same as *min*, but can be used for text, e.g., names of branches collected by a *node-name* field (see below). Comparison is alphabetically, “aaaab” is smaller than “bc”.

## 4.15 max-string

The largest of a list of strings. Same as *min-string*, but shows the maximum.

## 4.16 ratio

The ratio between the first and the sum of all following input fields. Syntax:

```
field "Name"  
  field-type "ratio"  
  own-fields ["field1", "field2", ...]  
  child-fields ["field3", ...]  
  sibling-fields [...]  
  parent-fields [...]
```

were “field1”, “field2”, “field3”, ..., are the names of data or tree fields.

Result: The value *field1 / (field2 + field3 + ...)*, where *N* is the number of fields.

## 4.17 ratio-percent

Same as “ratio”, but displayed as percentage (e.g., 0.75 is displayed as 75 %).

## 4.18 node-name

The name of the node’s parent in another tree. Syntax:

```
field "Name"
  field-type "node-name"
  own-fields []
  child-fields []
  sibling-fields []
  parent-fields [N]
```

where  $N$  is an integer number.

Result: Displays the name of the node’s parent in tree  $N$ . Trees are counted starting with 0.

Example: This field is called “Project” and is defined in a tree “Time”, which is the first tree (i.e. Tree 0). There is another tree called “Projects”, which is the third tree (i.e. Tree 2):

```
tree "Time"
  field "Project"
    field-type "node-name"
    own-fields []
    child-fields []
    sibling-fields []
    parent-fields [2]

tree "Tasks"
  ...

tree "Projects"
  ...
```

This would create the column “Project” in the tree view of the “Time” tree. The line `parent-fields[2]` means each entry shows the respective node’s parent in the “Project” tree (e.g. “TreeTime”).

## 4.19 node-path

Same as “node-name”, but instead of the parent’s name, the entire path is shown, using “|” as delimiter (e.g. “Coding | Open Source | TreeTime”).





## HISTORY AND ROAD MAP

### 5.1 Past

#### 5.1.1 2015

- November: First implementation, simple data types, simple GUI

#### 5.1.2 2016

- February: Implemented selection (the same item gets selected in all trees, changing a tab shows the same item)
- March: Implemented remaining local functionality (Copy Branch as Sibling, Copy Children to Siblings, Remove from this Tree, Delete Item)
- August: Created installable python package

#### 5.1.3 2017

- May: Implemented new field type *text*
- June: Create deployable packages for Linux and Windows
- June: Made **pre-release v0.0** available
- October: Implemented new field type *node-path*, re-wrote the way nodes move to new parents
- November: Uploaded package to pypi.python.org, *TreeTime* can now be installed using pip

#### 5.1.4 2018

- October: Re-implemented the parent selection mechanism. The old cascaded menus have been replaced with single drop down lists.
- October: Re-furbished the GUI and removed a couple of bugs. Slightly changed the data file format. Implemented theme selection. Tested pyqtdeploy for deployment instead of pyinstaller. Updated the description.
- November: Released **version 2018-10**

### 5.1.5 2019

- January: Implemented new field type “URL”

### 5.1.6 2020

- June: Fixed problem with protected cells (typing into a cell without data could cause a crash), and fixed file selection dialog (now only offers .trt files).
- July: Implemented text export - single branches or complete trees can now be exported to txt files.
- August: Implemented time counters - nodes can record the time using a special field of type “timer” (experimental). GUI buttons can start and stop the stopwatch function.
- September: Added move-to-top-level option for first level nodes
- October: Added a dark and a light palette for GUI colours, selectable in addition to the theme selection.
- November: Fixed too slow editing in text fields when tree files are big (>1.5 MB).

### 5.1.7 2021

- January: Released **version 2021.01**.
- January: Bugfixing (timer crash)
- February: Released **version 2021.2**.
- March: New functions “Delete node” and “Remove node from tree” now move descendants one level up. “Remove branch” removes the respective branch in all trees, “Delete branch” deletes a branch, all child branches and inter-connections in all trees.
- March: If a file with running timers is saved, those timers will be running when the file is loaded.
- March: Added tooltips for main buttons
- March: Implemented HTML export of branches and complete trees
- March: Added auto-delete for orphans
- March: Released **version 2021.3**
- April: Added file option
- April: Implemented four-column layout and rainbow colours for html export
- April: Released **version 2021.4**
- May: Improvement to html and txt export (changed colours, headings have no different sizes)
- May: On export of both html and txt, user can now decide how many tree levels (depth) should be exported.
- May: Released **version 2021.5**
- July: Fixed broken application logo
- July: Implemented CSV export
- August: Released **version 2021.8**
- September: Added new export option “Text to Clipboard”
- November: Added new export option “Html (List) to File”
- December: Added two primitive template files (a text-only single tree and dual tree mindmap)

- December: Released **version 2021.9**

### 5.1.8 2022

- March: Fixed crash bug on non-export
- March 2022: Improved sorting and grouping in html export, changed to five columns
- June 2022: Added a tutorial file
- June 2022: Added first-use dialog when no file is loaded, instead of the file-open dialog
- June 2022: Released **version 2022.1**

### 5.1.9 2023

- February 2023: Added new tree field types “concatenation” and “set”.
- February 2023: Implemented adjustable width for the data item and the tree table main view.
- February 2023: Release **version 2023.1**
- April 2023: Removed deprecated tree field (“concatenation”), fixed missing logo.
- May 2023: Ported to PyQt 6.0
- May 2023: Implemented auto-adjusting name column
- June 2023: Created new default theme “Organic”, a mix between Fusion and Breeze
- June 2023: Implemented display of tree field definitions and of data field definitions
- July 2023: Release **version 2023.2**
- October 2023: Fixed crash when exporting text to clipboard.

## 5.2 2024

- January 2024: Changed node symbol to small circle in text eport (after asking users on social media).
- February 2024: Implemented min, max, min-string, max-string fields.
- March 2024: Implemented longtext data field.
- April 2024: Extended documentation on readthedocs.io. Release **version 2024.1**
- April 2024: Restructured export area, added name-only export. Made all export options (full tree / branch / node with context) (all fields / names only) available for all file formats and for both file and clipboard export.
- April 2024: Release **version 2024.2**
- Done March 2024: Implemented changeable font size (zoom) of data display
- Done May 2024: Implemented continuous text and html export
- July 2024: Release **version 2024.3**

## 5.3 Present

- Bugfixing
- Extend documentation on readthedocs.io
- Add more fields
- Add more examples and more template data files
- Structure editing/viewing in extra tab (editing the structure, number and definitions and trees and tree fields and data fields)

## 5.4 Future

### 5.4.1 Near Future

- Implement search function

### 5.4.2 Mid Future

- Implement global functions (Linearise Tree, Level-Swap, Merge identical Siblings, Merge Identical Parents/Children, Remove all Orphans, Insert all Orphans as Children)

### 5.4.3 Far Future

- Implement safe usage by multiple simultaneous users
- Implement a database backend instead of text file storage
- A whole lot of other fancy things that will probably never get done
- genindex
- modindex
- search