TD

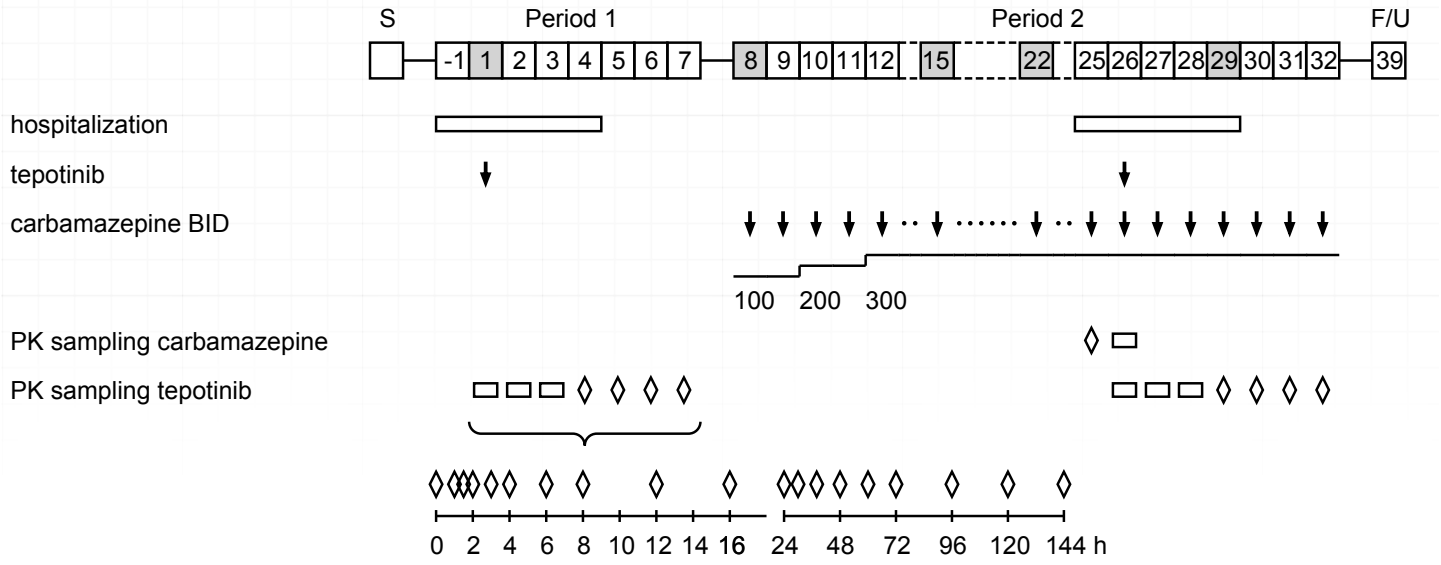
None

Table of contents

1. Trial design visualization 5
2. About 6
3. Input file format 7
   - 3.1 Periods 7
   - 3.2 Period elements 9
   - 3.3 Advanced notation 13
4. Using TD 21
   - 4.1 Requirements 21
   - 4.2 Installation 21
   - 4.3 Running TD 21
5. Functions 25
   - 5.1 `activity_days(period)` 25
   - 5.2 `add_output(old, new)` 25
   - 5.3 `assert_interval_format(interval)` 26
   - 5.4 `assert_period_format(period)` 26
   - 5.5 `assert_procedure_format(procedure)` 26
   - 5.6 `day_index(period, day)` 26
   - 5.7 `decode_daylist(daylist)` 27
   - 5.8 `ensure_list(period, key)` 27
   - 5.9 `extract_footnotes(period, caption)` 28
   - 5.10 `extract_procedure(period, caption)` 28
   - 5.11 `extract_start_end(daylist)` 29
   - 5.12 `has_timescale(period, caption)` 29
   - 5.13 `item_names(periods, item_class)` 30
   - 5.14 `iterate_over_procedures(period, caption, out, function)` 30
   - 5.15 `leading_edge(x)` 31
   - 5.16 `main(file=<typer.models.ArgumentInfo object at 0x1038c10a0>, output=<typer.models.OptionInfo object at 0x1038c10d0>, font=<typer.models.OptionInfo object at 0x1038c1100>, fontsize=<typer.models.OptionInfo object at 0x1038c1130>, padding=<typer.models.OptionInfo object at 0x1038c1160>, condensed=<typer.models.OptionInfo object at 0x1038c1190>, ellipsis=<typer.models.OptionInfo object at 0x1038c11c0>, timescale=<typer.models.OptionInfo object at 0x1038c11f0>, graph=<typer.models.OptionInfo object at 0x1038c1220>, footnotes=<typer.models.OptionInfo object at 0x1038c1250>, all=<typer.models.OptionInfo object at 0x1038c1280>, autocompress=<typer.models.OptionInfo object at 0x1038c12b0>, version=<typer.models.OptionInfo object at 0x1038c12e0>)` 32
   - 5.17 `make_dayrange(start, duration)` 34
   - 5.18 `normalize_procedure(procedure)` 34
   - 5.19 `period_day_centers(period, xoffset, daywidth_function)` 34
   - 5.20 `period_day_ends(period, xoffset, daywidth_function)` 34

5.21 period_day_starts(period, xoffset, daywidth_function) 35
5.22 render_daygrid(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True) 35
5.23 render_dose_graph(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True) 36
5.24 render_dummy(period, xoffset, yoffset, lineheight, metrics) 37
5.25 render_interval(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True) 37
5.26 render_periodcaption(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True) 38
5.27 render_periods(periods, x, y, caption, height, render_function, metrics, style, dashes=False, footnotes=False, **kwargs) 39
5.28 render_procedure(period, caption, xoffset, yoffset, lineheight, metrics, style, default_symbol='diamond', first_pass=True) 40
5.29 render_times(period, caption, xoffset, yoffset, lineheight, metrics, style, maxwidth=100) 41
5.30 trailing_edge(x) 44
5.31 unnormalize_procedure(procedure) 45

1. Trial design visualization



TD is a command line tool written in python to create overview figures for clinical study designs, and to summarize relevant elements of the schedule of assessments.

The input describing the study design is expected as a json-formatted text file.

The output is provided in vector graphic form (as a .svg file) that can be easily included in Office documents or websites.

1.0.1 Getting started

- Go to Use for guidance how to install and run the tool.
- The Input page describes the structure and syntax of the input required to generate

1.0.2 Code documentation

- A detailed documentation of the python code is provided in the Code documentation section.

2. About

This is a command line tool to create high-quality study design overview figures for clinical studies.

TD uses a json-formatted input file to specify the study design elements and has a number of options to further specify the graphical output.

TD is written in functional python by Rainer Strotmann (@auchkunscht).

3. Input file format

The TD tool expects a json-formatted input file (see Use) that specifies the study elements to be rendered.

In general, the json format has specific syntactic requirements in order to be read correctly by TD:

- All json elements are enclosed in curly brackets, including the complete input file
- Fields within elements have a name that must be enclosed in quotes, e.g., "caption", followed by a colon and the respective value
- Values can be numerical, character (i.e., enclosed in quotes), other json elements, or lists (enclosed in square brackets) of any of the previous
- Fields must be separated by commas, but there is no comma after the respective last element

The below overview specifies the specific expected format of the json-formatted input file to describe the trial design elements.

3.1 Periods

On the highest level, clinical study elements within the input file are expected to be structured in *periods*. In oncology studies that are structured in cycles, each cycle can be encoded as a period.

As a minimum, each *period* element needs to have *caption*, *start* and *duration* fields. A minimum period "Period 1" that includes days 1 through 7 is described like this:

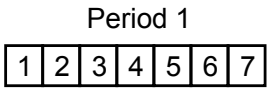
```
{
    "caption": "Period 1",
    "start": 1,
    "duration": 7
}
```

Note that the period definition is enclosed in curly brackets. In the json format, this is used to structure elements. You will see below that objects can themselves contain objects. In additino, some elements are grouped together in a list. Lists are delimited in square brackets.

In the input file, periods, even if only one is defined, are expected as a *periods* list. A minimal viable input file could look like this:

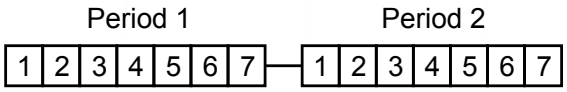
```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": 1,
            "duration": 7
        }
    ]
}
```

The respective output is:



Further periods can be added as more members to the *periods* list, e.g.:

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": 1,
            "duration": 7
        },
        {
            "caption": "Period 2",
            "start": 1,
            "duration": 7
        }
    ]
}
```



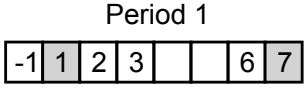
3.1.1 Period formatting

Additional fields can be added to the *period* elements to specify their visual appearance. Each *period* can have the following fields:

- *daylabels*, a list of days for which daynumbers will be printed. This can be used to precisely control the visual appearance of the period. Days that have no daylabel are visually compressed when the `td -c` option is used for the rendering. This can help focusing the figure on relevant days by omitting unnecessary detail. See Condensed in the Use section for details on this. If *daylabels* is not specified, all day numbers are printed.

- *dayshading*, a list of days for which the day box will have a grey background. This can be used to highlight relevant days, e.g., weekend days.

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, 1, 2, 3, 6, 7],
            "dayshading": [1, 7]
        }
    ]
}
```



Note

Wherever lists of days are expected as the input to a field, there are different options. In the simplest case, a list of day numbers can be provided:

```
[-1, 1, 2, 3, 5, 7]
```

If subsequent days are to be specified, it may be more convenient to use a range format (e.g, *"1-3"*). Note that ranges need to be provided with enclosing double quotes. Individual days and ranges can be combined:

```
[-1, "1-3", 5, 7]
```

3.2 Period elements

To every *period*, trial procedure elements can be appended. There are three classes of elements: *intervals*, *administrations* and *procedures*. Individual elements of these classes must be given as a list of the respective types (see examples below). This results in a hierarchical structure of the input file.

The three element classes have different visual representations and may have different property fields. The following gives an overview on this.

Note

To better visualize their hierarchy, successive indentation levels are used in the json blocks shown in this documentation. Note that this is not a strict requirement for the json-formatted input file, however it is considered good practice.

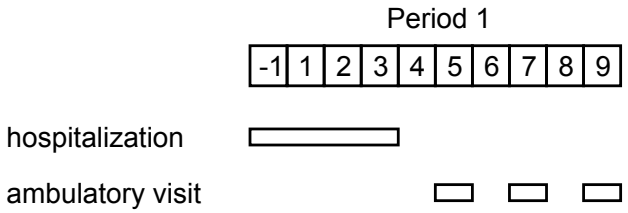
3.2.1 Intervals

Intervals are typically used to denote hospitalization phases or ambulatory visits, and are represented in the output as boxes spanning the respective days. There are two different ways of specifying intervals:

- For intervals that span multiple days, a *start* and *duration* must be given.
- For single-day intervals, a list of *days* can be given instead. The duration is then assumed to be one day.

A valid example may look like this:

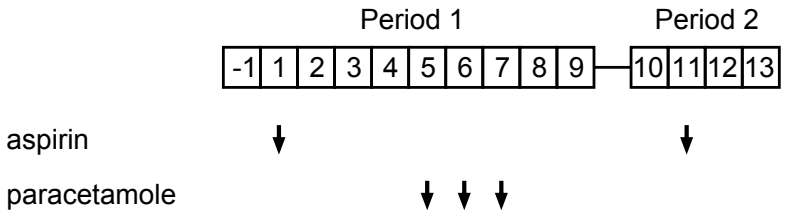
```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 10,
            "daylabels": [-1, "1-9"],
            "intervals": [
                {
                    "caption": "hospitalization",
                    "start": -1,
                    "duration": 4
                },
                {
                    "caption": "ambulatory visit",
                    "days": [5, 7, 9]
                }
            ]
        }
    ]
}
```



3.2.2 Administrations

IMP administrations are shown in the output using arrow symbols. The minimally required fields are *caption* and *days*. As mentioned above, *days* is expected to be a list of numbers or (double-quoted) day ranges. Both are used in the below example:

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 10,
            "daylabels": [-1, "1-9"],
            "administrations": [
                {
                    "caption": "aspirin",
                    "days": [1]
                },
                {
                    "caption": "paracetamole",
                    "days": ["5-7"]
                }
            ]
        },
        {
            "caption": "Period 2",
            "start": 10,
            "duration": 4,
            "daylabels": ["10-14"],
            "administrations": [
                {
                    "caption": "aspirin",
                    "days": [11]
                }
            ]
        }
    ]
}
```



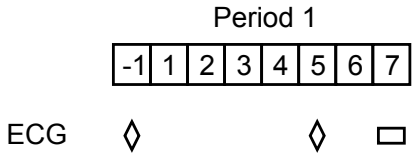
3.2.3 Procedures

All other study assessments (e.g., blood sampling, ECG, etc.) are specified as *procedures*.

In general, study procedures may be conducted once per day or multiple times per day. In the graphical output, the symbols are diamonds and boxes, respectively. The frequency of a *procedure* can be specified in the input file using the *freq* field: "QD" indicates once daily, while "rich" indicates multiple daily time points. The *freq* field can also be omitted to indicate once daily scheduling (e.g., for the ECG on day 5, below).

Alternatively to the *freq* field, more granular daily schedules can be defined using the *times* field (see exact procedure times, below).

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, "1-7"],
            "procedures": [
                {
                    "caption": "ECG",
                    "days": [-1],
                    "freq": "QD"
                },
                {
                    "caption": "ECG",
                    "days": [5]
                },
                {
                    "caption": "ECG",
                    "days": [7],
                    "freq": "rich"
                }
            ]
        }
    ]
}
```



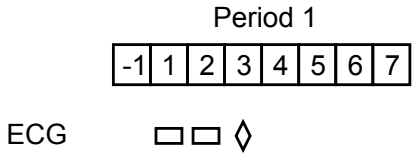
3.3 Advanced notation

3.3.1 Exact procedure times

As a more granular alternative to the rather coarse definition of the procedure frequency using the *freq* field, precise preocedure times can be noted for procedures, e.g., for PK samplings. The times (in hours) is to be provided as a list to the *times* field. In addition, a *relative* field must be provided to clearly indicate to which day the times refer:

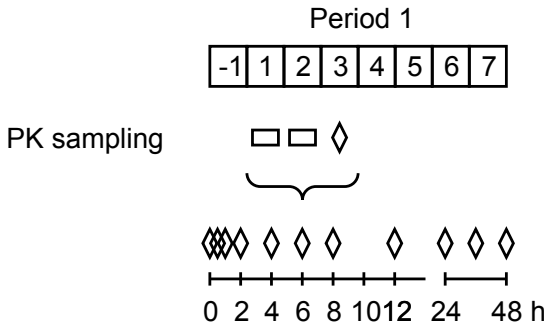
```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, "1-7"],
            "procedures": [
                {
                    "caption": "ECG",
                    "days": [1],
                    "times": [0, 0.5, 1, 2, 4, 6, 8, 12, 24, 36, 48],
                    "relative": 1
                }
            ]
        }
    ]
}
```

This notation automatically assigns the right symbols for the days onto which the time points fall. In the above example, the time point list indicates that there are multiple time points on days 1 and 2 (leading to a box symbol for these days) but a single time point on day 2 (diamond symbol):



Sometimes, it is desirable to print a more detailed timeline for a procedure (e.g., to show the PK sampling schedule). This can be achieved by adding a *timescale* field with the value "show". In addition, as described on the Use page, the commandline parameter `--timescale` (or short `-t`) must be set to achieve this.

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, "1-7"],
            "procedures": [
                {
                    "caption": "PK sampling",
                    "days": [1],
                    "times": [0, 0.5, 1, 2, 4, 6, 8, 12, 24, 36, 48],
                    "relative": 1,
                    "timescale": "show"
                }
            ]
        }
    ]
}
```



(Created by invoking: `td -t sample.jpg`)

3.3.2 Exact dose information

In some cases, the dose for an IMP changes over time in a scheduled way, e.g. to phase in or out a sensitive drug. In this cases, the respective administration element can include specific dosing information using a numerical *dose* field. The below example shows a dose escalation for carbamazepine:

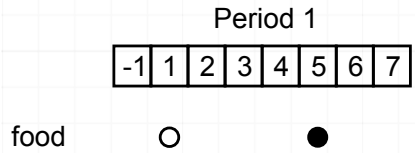
```
"administrations": [
    {
        "caption": "carbamazepine BID",
        "days": ["8-9"],
        "dose": 100
    },
    {
        "caption": "carbamazepine BID",
        "days": ["10-11"],
        "dose": 200
    },
    {
        "caption": "carbamazepine BID",
        "days": ["12-32"],
        "dose": 300
    }
]
```

As detailed on the Use page, a dose graph can then be displayed in the output to indicate the dose over time using the "--graph" (or "-g") option.

3.3.3 Other procedure symbols

In cases where prodecure symbols should indicate different conditions (e.g., fasted vs. fed), a *value* field can be included in the procedure element. Procedures with a *value* field are not shown as diamonds but as hollow circles if the value is zero, and filled circles for any value other than zero:

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, "1-7"],
            "procedures": [
                {
                    "caption": "food",
                    "days": [1],
                    "value": 0
                },
                {
                    "caption": "food",
                    "days": [5],
                    "value": 1
                }
            ]
        }
    ]
}
```

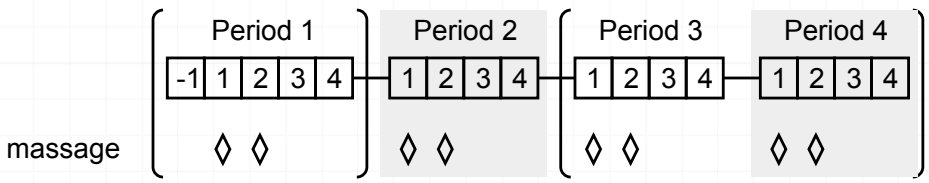


3.3.4 Period highlighting

To visually make a period stand out, the *period* element can include a *decoration* field. Possible values are "highlighted" for a shaded background or "bracketed" to indicate, e.g., optional periods. The *decoration* field may have a single value or a combination of multiple values. In the latter case, the values must be provided as a list, see, e.g., Period 4 in the below example.

Two adjacent periods that are both bracketed will not be rendered as individually bracketed but enclosed together in brackets, see, e.g., Periods 3 and 4 in the below example.

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 5,
            "daylabels": [-1, "1-4"],
            "decoration": "bracketed",
            "procedures": [
                {
                    "caption": "massage",
                    "days": ["1-2"]
                }
            ]
        },
        {
            "caption": "Period 2",
            "start": 1,
            "duration": 4,
            "daylabels": ["1-4"],
            "decoration": "highlighted",
            "procedures": [
                {
                    "caption": "massage",
                    "days": ["1-2"]
                }
            ]
        },
        {
            "caption": "Period 3",
            "start": 1,
            "duration": 4,
            "daylabels": ["1-4"],
            "decoration": "bracketed",
            "procedures": [
                {
                    "caption": "massage",
                    "days": ["1-2"]
                }
            ]
        },
        {
            "caption": "Period 4",
            "start": 1,
            "duration": 4,
            "daylabels": ["1-4"],
            "decoration": ["bracketed", "highlighted"],
            "procedures": [
                {
                    "caption": "massage",
                    "days": ["1-2"]
                }
            ]
        }
    ]
}
```

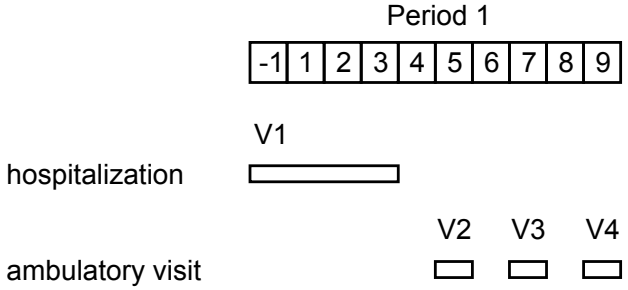


3.3.5 Procedure labels

To annotate individual procedure instances, e.g., to give visits a unique identifier (see below) or to add details to a procedure on a certain day, the *labels* field can be added to a procedure. Its value is expected to be a list of labels corresponding to the list of *days* of that procedure. Labels are rendered in the output above the respective days. Lables can be added to intervals, administrations or procedures.

Note: If the procedure is an interval that is defined using *start* and *duration*, that list is expected to contain only one label which is then displayed above the start day of the interval:

```
{
    "periods": [
        {
            "caption": "Period 1",
            "start": -1,
            "duration": 10,
            "daylabels": [-1, "1-9"],
            "intervals": [
                {
                    "caption": "hospitalization",
                    "start": -1,
                    "duration": 4,
                    "labels": ["V1"]
                },
                {
                    "caption": "ambulatory visit",
                    "days": [5, 7, 9],
                    "labels": ["V2", "V3", "V4"]
                }
            ]
        }
    ]
}
```



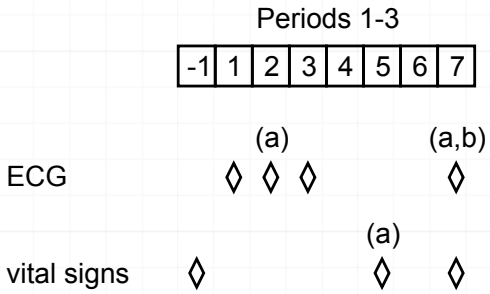
3.3.6 Footnotes

Footnotes can be added to any *interval*, *administration* or *procedure* field, defining an entry *footnotes* that has a list as its value. The elements of that list need to have *days*, *symbol* and *text* entires that define the days to be annotated, the footnote symbol to be rendered above it, and the full text that is rendered at the bottom of the output.

Footnote symbols can be re-used in other procedures. the *text* field in further copies is expected to be an empty string (i.e., "").

Note that in the output, footnotes are only rendered if specified explicitly with the "--footnote" (or "-n") option (see Use). The below figure was created by invoking `td -n sample.jpg`:

```
{
    "periods": [
        {
            "caption": "Periods 1–3",
            "start": -1,
            "duration": 8,
            "daylabels": [-1, "1–7"],
            "procedures": [
                {
                    "caption": "ECG",
                    "days": [ 1, 2, 3, 7],
                    "footnotes": [
                        {
                            "days": [2, 7],
                            "symbol": "a",
                            "text": "Period 1 only"
                        },
                        {
                            "days": 7,
                            "symbol": "b",
                            "text": "Before discharge"
                        }
                    ]
                },
                {
                    "caption": "vital signs",
                    "days": [-1, 5, 7],
                    "footnotes": [
                        {
                            "days": 5,
                            "symbol": "a",
                            "text": ""
                        }
                    ]
                }
            ]
        }
    ]
}
```



(a) Period 1 only
(b) Before discharge

4. Using TD

This gives an overview on how to run the td tool from the command line.

4.1 Requirements

TD is written in python 3. In order to run the tool on your system, python 3 must be installed. You can find out which version of python is installed (or is the default) on your system with `python --version`.

4.2 Installation

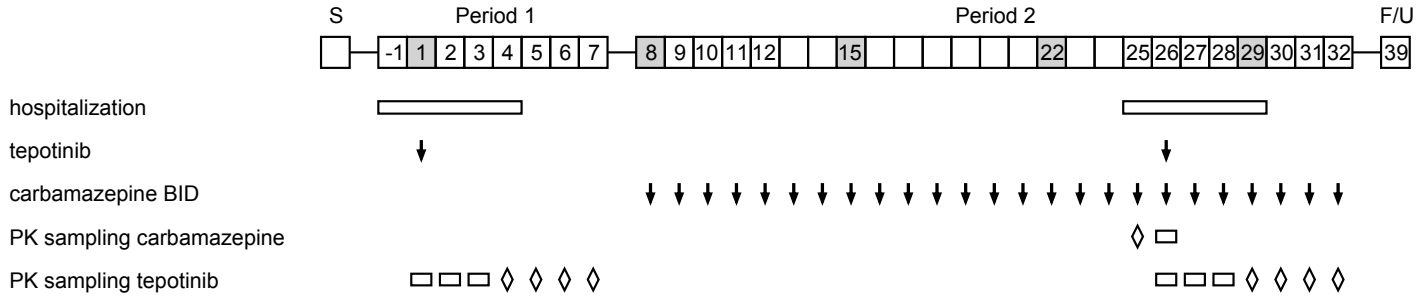
TD is provided as a .whl file that can be installed using the python packet manager, pip.

To install the package on your system, run `pip install NAME.whl` from the command line where *NAME* is the exact name of the provided file (e.g., "td-2.1-py3-none-any.whl"). The filename begins with *td-*... but the full name is dependent on the release of the specific version, please check your version.

4.3 Running TD

TD is a command line script. Open up a terminal window and enter the TD command in the form `td [OPTIONS] FILE` where *FILE* corresponds to the json-formatted input file (see Input for details).

As a reference example, the following figure (based on this input file) was rendered running the basic command `td test.json`), i.e., without further OPTIONS:



The following section summarizes the available rendering options (*OPTIONS*) that can be used to further specify the visual output.

The available options can also be shown with `td --help`. In the current version of TD, they include:

Option	Alternative	Description
--output TEXT	-o	Output file name. Default: INPUT.svg
--fontsize INTEGER	-s	Output font size (default 11)
--font TEXT	-f	Output font type (default: Arial)
--padding FLOAT	-p	Y-axis padding factor (default 1)
--condensed	-c	Show condensed daygrid
--timescale	-t	Show time scale
--graph	-g	Show dose graph
--ellipsis	-e	Reduce symbols in condensed output
--footnotes	-n	Show footnotes
--all	-A	All options, equivalent to -ctge
--version		Show version and exit
--help		Show this message and exit.

4.3.1 Output file

The default output file name is the input file name (e.g., "test.json") with the .svg extension (i.e., "test.svg"). This can be overridden with the *--output* or *-o* option.

4.3.2 Font size and family

The default font is Arial 11 point. Both font and size can be overridden using the *--font* (*-f*) and *--fontsize* (*-s*) options. The available font families depend on the fonts installed on the target system.

In addition, the generated svg file can be scaled in the target application (e.g., MS PowerPoint), and any element can be reformatted separately.

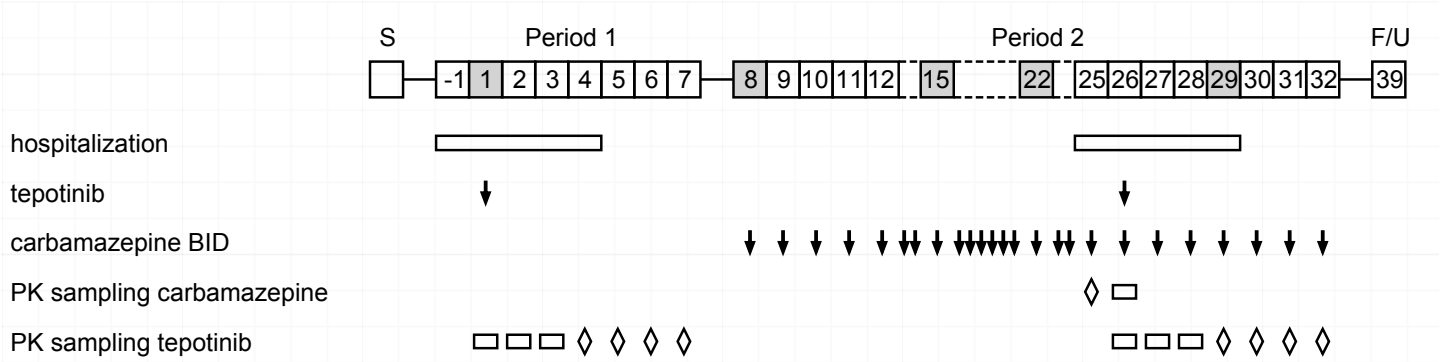
4.3.3 Padding

The parameter *--padding* (*-p*) increases or decreases the vertical space between period elements. The default of 1 should work in most cases.

4.3.4 Condensed

This option can be used to compress the visual output horizontally. Period days for which no daylabel has been specified in the input file (see Period formatting in the Input section) will be rendered narrower.

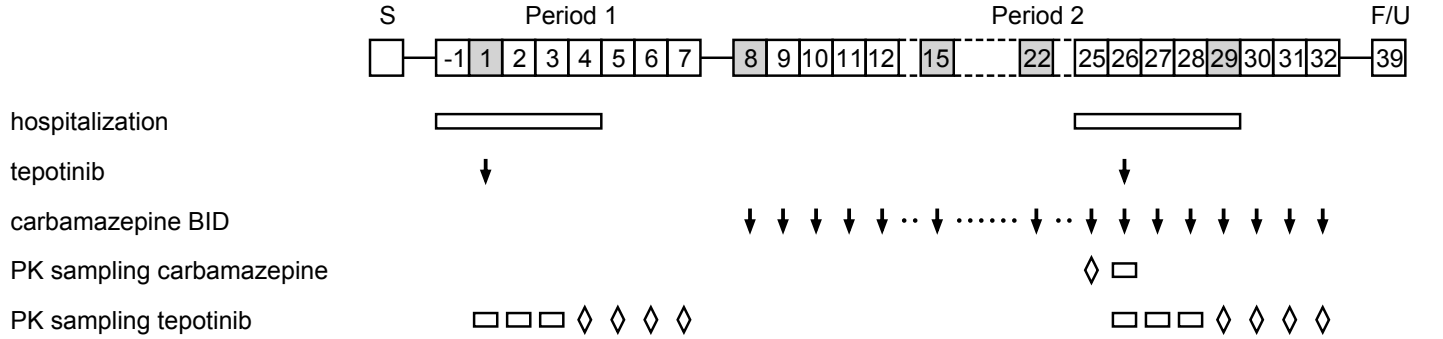
The below version of the above example was rendered using the same input file but with the *--condensed* option, i.e., running `td --condensed test.json` (or, alternatively, `td -c test.json`):



4.3.5 Ellipsis

In addition to condensed output, daily recurring procedure symbols that visually clutter the output can be reduced using the *--ellipsis* or *-e* option.

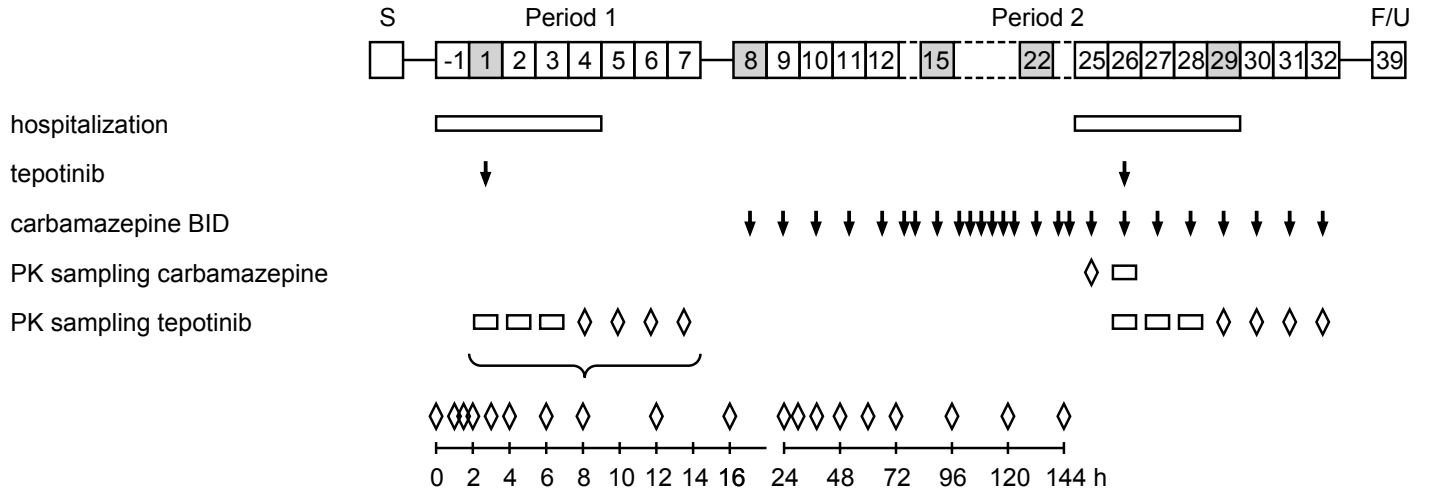
The below version was rendered running `td -ce test.json` for a combination of condensed and ellipsis output:



4.3.6 Timescale

Procedures that have exact time information included in the input file, e.g., PK samplings (see "exact procedure times"), can be displayed with an inset figure underneath that shows the timescale detail.

The following output was generated using the `td --condensed --timescale test.json` (or, alternatively, `td -ct test.json`) command:



Note that that there must be the `"timescale": "show"` line included in the json input file for this to take effect on a procedure (see "exact procedure times").

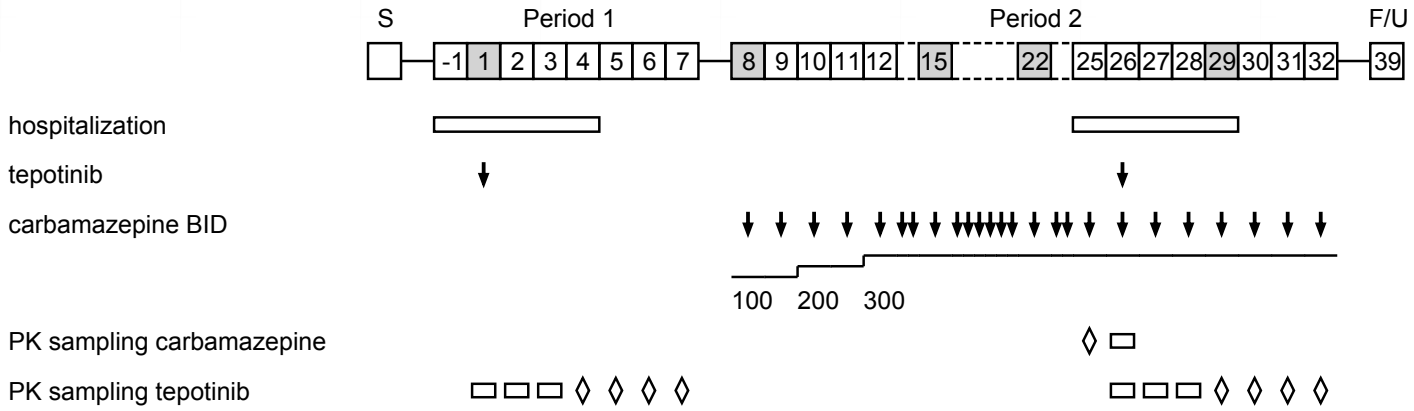
For visual clarity, it is recommended to limit display of timescales to the last element (the one at the bottom) of a trial design visualization.

4.3.7 Dose graph

In cases where intraindividual dose escalation occurs, e.g., to phase in a drug, a dose graph can be shown underneath the administration symbols to indicate the dose over time.

A prerequisite is that the respective dosing information is included per day in the input file (see Exact dose information in the Input section).

The following output was generated running the `td --condensed --graph test.json` (or, alternatively, `td -cg test.json`) command to include a dose graph for carbamazepine:



4.3.8 Footnotes

If footnotes have been defined in the input file (see Footnotes in the Input section), they can be rendered in the output using the "--footnotes" (or "-n") option.

5. Functions

5.1 activity_days(period)

returns a list of boolean values per day to indicate whether there are procedures on the day

• Source code in `td/td.py`

```
def activity_days(period):
	"""returns a list of boolean values per day to indicate whether there are procedures on the day"""
	start = period["start"]
	duration = period["duration"]
	if start <0 and start+duration >0:
		duration+=1

	# start and end of period, start and end of trains of procedure days
	out = [start, start+duration-1]
	for x in ["administrations", "procedures"]:
		if x in period.keys():
			for i in period[x]:
				if "days" in i.keys():
					temp = decode_daylist(i["days"])
					out += extract_start_end(temp)

	# all PK days
	if "procedures" in period.keys():
		for i in period["procedures"]:
			if "times" in i:
				out += [d for (d, t, r) in normalize_procedure(extract_procedure(period, i["captior

	if "intervals" in period.keys():
		for i in period["intervals"]:
			if "start" in i.keys() and "duration" in i.keys():
				start = i["start"]
				duration = i["duration"]
				if start < 0 and start + duration > 0:
					duration += 1
				out += extract_start_end(make_dayrange(start, duration))
	out.sort()
	temp = [False] * period['duration']
	for i in list(dict.fromkeys(out)):
		temp[day_index(period, i)] = True
	return(temp)
```

5.2 add_output(old, new)

add output of render functions

• Source code in `td/td.py`

```
def add_output(old, new):
	"""add output of render functions"""
	return([o+n for o, n in zip(old, new)])
```

5.3 assert_interval_format(interval)

assert minimum required fields are present in interval

• Source code in `td/td.py`

```
def assert_interval_format(interval):
        """assert minimum required fields are present in interval
        """
        try:
                assert "caption" in interval.keys()
                assert ("start" in interval.keys() and "duration" in interval.keys()) or "days" in interval.keys()
        except AssertionError as err:
                raise AssertionError(f'missing required fields (caption", and either start and duration, or days) i
        return
```

5.4 assert_period_format(period)

assert minimum required fields are present in period

• Source code in `td/td.py`

```
def assert_period_format(period):
        """assert minimum required fields are present in period
        """
        try:
                assert "caption" in period.keys()
                assert "duration" in period.keys() and type(period["duration"])==int
        except AssertionError as err:
                raise AssertionError(f'missing required fields (caption and duration) in period {period}')
        return
```

5.5 assert_procedure_format(procedure)

assert minimum required fields are present in procedure

• Source code in `td/td.py`

```
def assert_procedure_format(procedure):
        """assert minimum required fields are present in procedure
        """
        try:
                assert "caption" in procedure.keys()
                assert "days" in procedure.keys()
        except:
                raise AssertionError(f'missing required fields (caption and days) in procedure {procedure}')
        return
```

5.6 day_index(period, day)

convert day to index within daylist

• Source code in `td/td.py`

```
def day_index(period, day):
        """convert day to index within daylist"""
        temp = day - period['start']
        if period['start'] < 0 and day > 0:
                temp -= 1 # correct for absent day 0
        if temp <0 or temp>period["duration"]-1:
                raise IndexError(f'day index {day} out of range ({period["start"]} to {period["start"]+period["dura
        return(temp)
```

5.7 decode_daylist(daylist)

convert 'days' field (including day ranges) to list of individual days

Convert list of period days given in a flxible format into a well-formed into a list of days. The input can contain either days in numerical format (e.g., -1, 1, 2), or as strings that may represent single days (e.g., "-1", "1") or day ranges (e.g., "1-3") . Day ranges can also include multiple segments (e.g., "1-3, 5-7", "1-3, 4, 5").

:param daylist: input list of numbers and/or strings representing individual days or day ranges (see above) :type daylist: list :rtype: list :return: list of days in strict numerical form

• Source code in `td/td.py`

```
def decode_daylist(daylist: list) -> list:
        """convert 'days' field (including day ranges) to list of individual days

        Convert list of period days given in a flxible format into a well-formed into a list of days. The input car

        :param daylist: input list of numbers and/or strings representing individual days or day ranges (see above)
        :type daylist:  list
        :rtype:                 list
        :return:        list of days in strict numerical form
        """
        days = []
        if not isinstance(daylist, list):
                daylist = [daylist]
        for i in daylist:
                if isinstance(i, int):
                        days.append(i)
                elif isinstance(i, str):
                        pat_element = r'(\d+)(-(\d+))?'
                        pat = f'({pat_element}(, )*)'
                        m = re.findall(pat, i)
                        if m:
                                for mm in m:
                                        if mm[3] == "":
                                                days.append(int(mm[1]))
                                        else:
                                                for i in range(int(mm[1]), int(mm[3])+1):
                                                        days.append(i)
        return(days)
```

5.8 ensure_list(period, key)

ensure to return a list for a key, even if value is not a list or the key is not used

• Source code in `td/td.py`

```
def ensure_list(period, key):
	"""ensure to return a list for a key, even if value is not a list or the key is not used"""
	if key in period.keys():
		if not isinstance(period[key], list):
			return([period[key]])
		else:
			return(period[key])
	else:
		return([])
```

5.9 extract_footnotes(period, caption)

extract footnotes for procedures by day, if applicable

• Source code in `td/td.py`

```
def extract_footnotes(period, caption):
	"""extract footnotes for procedures by day, if applicable"""
	out = [[False] * period['duration'], [''] * period['duration'], []]
	def temp(proc, out):
		if 'footnotes' in proc.keys():
			for f in proc["footnotes"]:
				if not "days" in f.keys():
					raise KeyError(f'no "days" in footnote "{f["text"]}"')
				else:
					if not isinstance(f["days"], list):
						daylist = [f["days"]]
					else:
						daylist = f["days"]
					for d in decode_daylist(daylist):
						i = day_index(period, d)
						out[0][i] = True
						if out[1][i]:
							out[1][i] += ","
						out[1][i] += str(f['symbol'])
						out[2].append([f['symbol'], f['text']])
		return(out)
	return(iterate_over_procedures(period, caption, out, temp))
```

5.10 extract_procedure(period, caption)

get specified administration/procedure as list of tuples (day, [times], relative) for individual days

• Source code in `td/td.py`

```
def extract_procedure(period, caption):
	"""get specified administration/procedure as list of tuples (day, [times], relative) for individual days
	"""
	out = []
	def temp(proc, out):
		if 'times' in proc.keys():
			t = proc['times']
		elif 'freq' in proc.keys() and proc['freq'] == 'rich':
			t = [0, 0]
		else:
			t = [0]
		if 'relative' in proc.keys():
			rel = proc['relative']
		else:
			rel = 1
		out += [(d, t, rel) for d in decode_daylist(proc['days'])]
		return(out)
	return(iterate_over_procedures(period, caption, out, temp))
```

5.11 extract_start_end(daylist)

from day list, extract start and end days of trains of days

• Source code in `td/td.py`

```
def extract_start_end(daylist):
	"""from day list, extract start and end days of trains of days"""
	last_day = 0
	out = []
	if daylist:
		daylist.sort()
		for i in daylist:
			if i == daylist[0] or i != last_day+1 and not (last_day == -1 and i == 1):
				out += [last_day, i]
			last_day = i
		out += [i]
	out = list(dict.fromkeys(out))
	if 0 in out:
		out.remove(0)
	return(out)
```

5.12 has_timescale(period, caption)

test if procedure has timescale in the respective period

• Source code in `td/td.py`

```
def has_timescale(period, caption):
	"""test if procedure has timescale in the respective period"""
	out = []
	def temp(proc, out):
		if 'timescale' in proc.keys():
			if proc['timescale'] == 'show':
				out.append(True)
		return(out)
	return(True in iterate_over_procedures(period, caption, out, temp))
```

5.13 item_names(periods, item_class)

return a list of interval/administration/procedure names included in a list of periods

:param periods: periods to search :type periods: list of period dictionaries :param item_class: class of items to include, can be 'intervals', 'administrations' or 'procedures' :type item_class: string :rtype: list :return: names of items of the respective type included in the list periods

- Source code in `td/td.py`

```
def item_names(periods, item_class):
        """return a list of interval/administration/procedure names included in a list of periods

        :param periods: periods to search
        :type periods:          list of period dictionaries
        :param item_class:      class of items to include, can be 'intervals', 'administrations' or 'procedures'
        :type item_class:       string
        :rtype:                         list
        :return:                names of items of the respective type included in the list periods
        """
        out = []
        for p in periods:
                if item_class in p.keys():
                        for proc in p[item_class]:
                                try:
                                        temp = proc['caption']
                                        if not temp in out:
                                                out.append(temp)
                                except:
                                        raise KeyError(f'no caption field in item {proc}')
        return(out)
```

5.14 iterate_over_procedures(period, caption, out, function)

apply a reduce function to all procedures with a given caption

:param period: period :type period: dictionary :param caption: procedure caption to select :type caption: string :param out: accumulator start value, mostly a list with length of the period :type out: flexible :rtype: flexible :return: accumulator out

• Source code in `td/td.py`

```
def iterate_over_procedures(period, caption, out, function):
        """apply a reduce function to all procedures with a given caption

        :param period:  period
        :type period:   dictionary
        :param caption: procedure caption to select
        :type caption:  string
        :param out:             accumulator start value, mostly a list with length of the period
        :type out:              flexible
        :rtype:                 flexible
        :return:                accumulator out
        """
        for x in ['intervals', 'administrations', 'procedures']:
                if x in period.keys():
                        for proc in period[x]:
                                if proc['caption'] == caption:
                                        function(proc, out)
        return(out)
```

5.15 `leading_edge(x)`

return leading True values in list of booleans

• Source code in `td/td.py`

```
def leading_edge(x):
        """return leading True values in list of booleans"""
        out = []
        status = False
        for i in x:
                out.append(i and i!=status)
                if i != status:
                        status = i
        return(out)
```

```
5.16 main(file=<typer.models.ArgumentInfo objec
t at 0x1038c10a0>, output=<typer.models.OptionI
nfo object at 0x1038c10d0>,
font=<typer.models.OptionInfo object at 0x1038c
1100>, fontsize=<typer.models.OptionInfo object
at 0x1038c1130>, padding=<typer.models.OptionIn
fo object at 0x1038c1160>, condensed=<typer.mod
els.OptionInfo object at 0x1038c1190>,
ellipsis=<typer.models.OptionInfo object at 0x1
038c11c0>, timescale=<typer.models.OptionInfo o
bject at 0x1038c11f0>, graph=<typer.models.Opti
onInfo object at 0x1038c1220>,
footnotes=<typer.models.OptionInfo object at 0x
1038c1250>, all=<typer.models.OptionInfo object
at 0x1038c1280>, autocompress=<typer.models.Opt
ionInfo object at 0x1038c12b0>,
version=<typer.models.OptionInfo object at 0x10
38c12e0>)
```

Clinical trial design visualization

Generates a 'schedule of assessments' overview for clinical trials, based on a json-formatted input FILE. Graphical output is provided in svg vector format that can be rendered by any webbrowser or directly imported into Office applications. Use below OPTIONS to manage the output style.

Version 2.1, proudly written in functional Python (Rainer Strotmann, Jan-2022)

• Source code in `td/td.py`

```
@app.command()
def main(
        file: str = typer.Argument(...),
        #debug: bool = typer.Option(False, "--debug", "-d", help="Debug output"),
        output: str = typer.Option("", "--output", "-o", help="Output file name"),
        font: str = typer.Option("Arial", "--font", "-f", help="Font type"),
        fontsize: int = typer.Option(14, "--fontsize", "-s", help="Font size"),
        padding: float = typer.Option(1, "--padding", "-p", help="Y-axis padding factor"),
        condensed: bool = typer.Option(False, "--condensed", "-c", help="Show condensed daygrid"),
        ellipsis: bool = typer.Option(False, "--ellipsis", "-e", help="Reduce symbols in condensed output"),
        timescale: bool = typer.Option(False, "--timescale", "-t", help="Show time scale"),
        graph: bool = typer.Option(False, "--graph", "-g", help="Show dose graph"),
        footnotes: bool = typer.Option(False, "--footnotes", "-n", help="Show footnotes"),
        all: bool = typer.Option(False, "--all", "-A", help="All options, equivalent to -ctgen"),
        autocompress: bool = typer.Option(False, "--autocompress", "-a", help="Automatically compress daygrid"),
        version: bool = typer.Option(False, "--version", help="Show version and exit", callback=version_callback)
        ):
        """Clinical trial design visualization


        Generates a 'schedule of assessments' overview for clinical trials, based on a json-formatted input FILE. (


        Version 2.1, proudly written in functional Python (Rainer Strotmann, Jan-2022)
        """
        if version:
                sys.exit(__version__)

        if all:
                condensed=True
                timescale=True
                graph=True
                ellipsis=True
                footnotes=True

        # read input file
        infile = pathlib.Path(file)
        inpath = pathlib.Path(file).resolve().parent
        if output:
                outfile = pathlib.Path.cwd().joinpath(output)
        else:
                outfile = inpath.joinpath(infile.stem + ".svg")
        try:
                with open(infile) as f:
                        td = json.load(f)
        except json.decoder.JSONDecodeError as err:
                sys.exit(f'Json syntax error in input file {infile}:\n{err}')
        except:
                sys.exit("Error loading input file")

        try:
                svg_out = render_td(td, title=infile.stem, debug=debug, fontsize=fontsize, font=font, condensed=cor
        except Exception as err:
                sys.exit(err)

        with open(outfile, "w") as f:
                f.write(svg_out)
        return
```

5.17 make_dayrange(start, duration)

return a range of days defined by start and duration. Day 0 does not exist and is deleted, if necessary

- Source code in `td/td.py`

```
def make_dayrange(start, duration):
        """return a range of days defined by start and duration. Day 0 does not exist and is deleted, if necessary"
        out = list(range(start, start + duration))
        if 0 in out:
                out.remove(0)
        return(out)
```

5.18 normalize_procedure(procedure)

break down procedure times to subsequent days, if longer than 24 h

- Source code in `td/td.py`

```
def normalize_procedure(procedure):
        """break down procedure times to subsequent days, if longer than 24 h"""
        out = []
        for (d, t, rel) in procedure:
                dd = 0
                while t:
                        temp = [i for i in t if i<24]
                        if temp:
                                out.append((d+dd, temp, rel))
                        t = [i-24 for i in t if i>=24]
                        dd += 1
        return(out)
```

5.19 period_day_centers(period, xoffset, daywidth_function)

return list of x-coordinates for day centers

- Source code in `td/td.py`

```
def period_day_centers(period, xoffset, daywidth_function):
        """return list of x-coordinates for day centers"""
        return([start + width / 2 for start, width in zip(period_day_starts(period, xoffset, daywidth_function), da
```

5.20 period_day_ends(period, xoffset, daywidth_function)

return list of x-coordinates for day ends

• Source code in `td/td.py`

```
def period_day_ends(period, xoffset, daywidth_function):
        """return list of x-coordinates for day ends"""
        starts = period_day_starts(period, xoffset, daywidth_function)
        widths = daywidth_function(period)
        return([s+w for s, w in zip(starts, widths)])
```

5.21 period_day_starts(period, xoffset, daywidth_function)

return list of x-coordinates for day starts

• Source code in `td/td.py`

```
def period_day_starts(period, xoffset, daywidth_function):
        """return list of x-coordinates for day starts"""
        out=[xoffset]
        acc = xoffset
        for i in daywidth_function(period):
                acc += i
                out.append(acc)
        return out[:-1]
```

5.22 render_daygrid(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True)

render svg output for the day grid for a period. Output is [svg_output, height]

• Source code in `td/td.py`

```
def render_daygrid(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True):
        """render svg output for the day grid for a period. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        y = yoffset

        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, height, metrics)

        for start, width, center, label, shading in zip(period_day_starts(period, xoffset, daywidth_function), dayw
                if shading:
                        svg_out += svg_rect(start, y, width, height, lwd=0, fill_color="lightgray")
                if width > textwidth_function("XX")/3:
                        svg_out += svg_rect(start, y, width, height, lwd=lwd)
                else:
                        svg_out += svg_line(start, y, start+width, y, lwd=lwd, dashed=True)
                        svg_out += svg_line(start, y+height, start+width, y+height, lwd=lwd, dashed=True)
                label = str(label)
                delta = textwidth_function("1")*.5 if label and label[0] == "1" else 0
                if width>textwidth_function(str(label)):
                        svg_out += svg_text(center - textwidth_function(str(label)) / 2-delta, yoffset + height - (
        return([svg_out, height+ypadding*2])
```

5.23 render_dose_graph(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True)

render dose over time for administration. Output is [svg_output, height]

• Source code in `td/td.py`

```
def render_dose_graph(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True):
        """render dose over time for administration. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight+ textheight_function("X"), metrics)

        startx = period_day_starts(period, xoffset, daywidth_function)
        endx = period_day_ends(period, xoffset, daywidth_function)
        doses = [i for i in extract_field(period, caption, "dose")]
        doses_num = [i for i in doses if isinstance(i, int) or isinstance(i, float)]
        if len(doses_num):
                maxdose, mindose = max(doses_num), min(doses_num)
                def dosey(dose):
                        return(yoffset + lineheight*0.6 - (dose-mindose)/(maxdose-mindose)*lineheight*0.6)
                # if doses:
                lastx, lasty, lastdose = 0, 0, 0
                lastend = 0
                for (s, e, d) in zip(startx, endx, doses):
                        if type(d)==int or type(d)==float:
                                svg_out += svg_line(s, dosey(d), e, dosey(d), lwd=lwd)
                                if lasty:
                                        svg_out += svg_line(lastx, lasty, s, dosey(d), lwd=lwd)
                                lastx, lasty = e, dosey(d)
                                if d != lastdose:
                                        if lastend + textwidth_function("n") < s:
                                                svg_out += svg_text(s, yoffset + lineheight + textheight_function('
                                                lastend = s + textwidth_function(str(d))
                                        lastdose = d
        return([svg_out, lineheight+textheight_function("X")+ypadding])
```

5.24 render_dummy(period, xoffset, yoffset, lineheight, metrics)

render bounding box for visual debugging purposes. Output is svg code only.

• Source code in `td/td.py`

```
def render_dummy(period, xoffset, yoffset, lineheight, metrics):
        """render bounding box for visual debugging purposes. Output is svg code only."""
        daywidth_function = metrics[0]
        return(svg_rect(xoffset, yoffset, period_width(period, daywidth_function), lineheight, lwd=0, fill_color="c
```

5.25 render_interval(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True)

render interval for procedure. Output is [svg_output, height]

• Source code in `td/td.py`

```
def render_interval(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True):
        """render interval for procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        y = yoffset + lineheight/2
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight, metrics)

        if first_pass:
                svg_out += svg_text(5, y + textheight_function(caption) * (1/2 - 0.1), caption)

        # render interval box
        starts = period_day_starts(period, xoffset, daywidth_function)
        ends = period_day_ends(period, xoffset, daywidth_function)
        widths = daywidth_function(period)

        height = 0.4 * lineheight
        if 'intervals' in period.keys():
                for intv in period['intervals']:
                        if intv['caption'] == caption:
                                if "start" in intv.keys() and "duration" in intv.keys():
                                        start_list, duration_list = [intv['start']], [intv['duration']]
                                elif "days" in intv.keys() and isinstance(intv["days"], list):
                                        start_list = decode_daylist(intv["days"])
                                        duration_list = [1 for i in decode_daylist(intv["days"])]
                                else:
                                        raise TypeError(f'{period["caption"]}, interval "{intv["caption"]}"')

                                for start, duration in zip(start_list, duration_list):
                                        startx = starts[day_index(period, start)]
                                        end = start + duration -1

                                        if start <0 and end >0:
                                                end += 1
                                        endx = ends[day_index(period, end)]
                                        if "decoration" in intv.keys():
                                                if intv["decoration"] == "bracketed":
                                                        wo = widths[day_index(period, start)]
                                                        wc = widths[day_index(period, end)]
                                                        svg_out += svg_open_bracket(startx, y, lineheight, wo*.6, x
                                                        svg_out += svg_close_bracket(endx, y, lineheight, wc*.6, xp
                                        svg_out += svg_rect(startx, y-height/2, endx-startx, height, lwd=lwd)
        return([svg_out, lineheight+ypadding])
```

5.26 render_periodcaption(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True)

render caption for period. The 'caption' input is ignored and the caption field of the input period is used. Output is [svg_output, height]

• Source code in `td/td.py`

```
def render_periodcaption(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True):
        """render caption for period. The 'caption' input is ignored and the caption field of the input period is u
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, height, metrics)
        xcenter = xoffset + period_width(period, daywidth_function)/2
        svg_out += svg_text(xcenter - textwidth_function(str(period['caption']))/2, yoffset+ height - (height-texth
        return([svg_out, height+ypadding/2])
```

5.27 render_periods(periods, x, y, caption, height, render_function, metrics, style, dashes=False, footnotes=False, **kwargs)

applies rendering function to all periods

• Source code in `td/td.py`

```
def render_periods(periods, x, y, caption, height, render_function, metrics, style, dashes=False, footnotes=False,
        """applies rendering function to all periods"""
        daywidth_function= metrics[0]
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        w = [period_width(i, daywidth_function) for i in periods]
        first = True
        last = False
        h = 0
        out = ""

        # render labels, if applicable
        has_labels = len([i for ii in [extract_labels(p, caption) for p in periods] for i in ii if i != '']) != 0
        has_footnotes = True in [i for ii in [extract_footnotes(p, caption)[0] for p in periods] for i in ii]
        if not footnotes:
                has_footnotes = False

        if has_labels or has_footnotes:
                xx = x
                for p in periods:
                        [svg_out, y_out] = render_labels_footnotes(p, caption, xx, y, height, metrics, style, footr
                        out += svg_out
                        xx += period_width(p, daywidth_function) + periodspacing
                h += lineheight
                y += h

        # render procedure
        for p in periods:
                if p==periods[-1]:
                        last=True

                [svg_out, y_out] = render_function(p, caption, x, y, height, metrics, style, first_pass=first, **kw
                out += svg_out

                if dashes and not last:
                        out += svg_line(x+period_width(p, daywidth_function), y+height/2, x+period_width(p, daywidt
                x += period_width(p, daywidth_function) + periodspacing
                first=False
        return(add_output(["", h], [out, y_out]))
```

5.28

render_procedure(period, caption, xoffset, yoffset, lineheight, metrics, style, default_symbol='diamond', first_pass=True)

render procedure. Output is [svg_output, height]

• Source code in `td/td.py`

```
def render_procedure(period, caption, xoffset, yoffset, lineheight, metrics, style, default_symbol="diamond", first
        """render procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight, metrics)

        y = yoffset + lineheight/2 # center of the line
        if first_pass:
                svg_out += svg_text(5, y + textheight_function(caption) * (1/2 - 0.1), caption)

        centers = period_day_centers(period, xoffset, daywidth_function)
        widths = daywidth_function(period)
        brackets = extract_field(period, caption, "decoration")
        symbols = procedure_symbols(period, caption, default_symbol)
        dlabels = day_labels(period)
        values = extract_field(period, caption, "value")

        ellipses = [1 if (s!="" and l == "" and len(symbols)>3) else 0 for (s,l) in zip(symbols, dlabels)]

        for p, w, s, b, e, v in zip(centers, widths, symbols, brackets, ellipses, values):
                if s:
                        if e==1 and b=="" and ellipsis:
                                svg_out += svg_circle(p, y, lineheight/30, fill_color="black")
                        elif v != "":
                                if v == 0:
                                        svg_out += svg_symbol(p, y, w*.5, "circle", fill=False, fill_color="none",
                                else:
                                        svg_out += svg_symbol(p, y, w*.5, "circle", fill=True, fill_color="black")
                        else:
                                svg_out += svg_symbol(p, y, w, s, size=textheight_function("X"), lwd=lwd, title=cap
                                if b=="bracketed":
                                        svg_out += svg_open_bracket(p, y, lineheight, w*.8, xpadding=0, radius=line
                                        svg_out += svg_close_bracket(p, y, lineheight, w*.8, xpadding=0, radius=lir
        return([svg_out, lineheight+ypadding])
```

5.29 render_times(period, caption, xoffset, yoffset, lineheight, metrics, style, maxwidth=100)

render timescale for procedure. Output is [svg_output, height]

- Source code in `td/td.py`

```
def render_times(period, caption, xoffset, yoffset, lineheight, metrics, style, maxwidth=100):
        """render timescale for procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        out = ""
        proc = normalize_procedure(extract_procedure(period, caption))

        ts_days = []
        for x in ['procedures', 'administrations']:
                if x in period.keys():
                        for p in period[x]:
                                if p['caption'] == caption:
                                        if "timescale" in p.keys() and p["timescale"]=="show":
                                                if "relative" in p.keys():
                                                        rel = p["relative"]
                                                else:
                                                        rel = p["days"][0]
                                                # ts_days.append(p["relative"])
                                                ts_days.append(rel)

        ts_days = set(ts_days)

        y = yoffset
        bracketheight = lineheight * 2/3

        last_scale_end = 0
        if ts_days:
                ## curly brackets
                if debug:
                        out += render_dummy(period, xoffset, y, bracketheight, metrics)

                for ts_d in ts_days:
                        times = unnormalize_procedure([i for i in proc if i[2]==ts_d])[0][1]
                        startx = period_day_starts(period, xoffset, daywidth_function)[day_index(period, min([i for
                        endx = period_day_ends(period, xoffset, daywidth_function)[day_index(period, max([i for (i,
                        radius = bracketheight/2
                        if radius * 4 > endx-startx:
                                startx -= radius/2
                                endx += radius/2
                                radius = (endx-startx)/5
                        out += svg_curly_up(startx, endx, y, radius=radius, lwd=lwd)
                y += bracketheight + ypadding*1.5

                ## timescales
                if debug:
                        out += render_dummy(period, xoffset, y, lineheight*1.33 + ypadding*2 + textheight_function(
                for ts_d in ts_days:
                        times = unnormalize_procedure([i for i in proc if i[2]==ts_d])[0][1]

                        maxtime = max(times)
                        break_time = min(sorted(list([i for i in times if i<24]))[-1] + 2, 23)
                        times_below = len([i for i in times if i<=break_time])
                        times_above = len([i for i in times if i>break_time])

                        startx = period_day_starts(period, xoffset, daywidth_function)[day_index(period, min([i for

                        ### scale
                        scale_height = lineheight/3
```

```
                scale_width = min(len(times) * textwidth_function("XX"), maxwidth-xoffset)
                scale_break = scale_width * times_below/(times_below+times_above)
                scale_gap = textwidth_function("m")

                scale_startx = max(min(startx, xoffset + period_width(period, daywidth_function) - scale_wi
                if scale_startx < last_scale_end:
                        y += lineheight*1.33 + ypadding*3 + textheight_function("X")

                def render_scale(x, y, width, height, scale_min, scale_max, scale_labels, show_unit=False):
                        out = svg_line(x, y, x+width, y, lwd=lwd)
                        label_widths = [textwidth_function(str(i)) for i in scale_labels]
                        last_label_end = 0
                        final_label_begin = x + width - label_widths[-1]/2
                        min_delta = textwidth_function(".")

                        for i, wi in zip(scale_labels, label_widths):
                                xi = (i-scale_min) * width/(scale_max-scale_min) + x
                                out += svg_line(xi, y-height/2, xi, y+height/2, lwd=lwd)
                                dxi = wi/2
                                if xi-dxi > last_label_end and xi+dxi < final_label_begin - min_delta:
                                        out += svg_text(xi-dxi, y+height/2+textheight_function("X")+ypaddir
                                        last_label_end = xi+dxi+min_delta
                                if i == scale_labels[-1]:
                                        temp = str(i)
                                        if show_unit:
                                                temp += " h"
                                        out += svg_text(xi-dxi, y+height/2+textheight_function("X")+ypaddir
                        return(out)

                def render_points(x, y, width, scale_min, scale_max):
                        points = [t for t in times if t>=scale_min and t<=scale_max]
                        points_x = [(i-scale_min) * width/(scale_max-scale_min) + x for i in points]
                        out = ""
                        for p, xi in zip(points, points_x):
                                out += svg_symbol(xi, y + lineheight/2, 0, "diamond", size=textheight_funct
                        return(out)

                out += render_points(scale_startx, y, scale_break, 0, break_time)
                out += render_points(scale_startx+scale_break+scale_gap, y, scale_width - scale_gap - scale

                out += render_scale(scale_startx, y+lineheight+ypadding, scale_break, scale_height, 0, brea
                if maxtime >=24:
                        out += render_scale(scale_startx+scale_break+scale_gap, y+lineheight+ypadding, scal
                last_scale_end = scale_startx + scale_width

        return([out, y+lineheight*1.33 + ypadding*3 + textheight_function("X")-yoffset])
```

5.30 trailing_edge(x)

return trailing True values in list of booleans

• Source code in `td/td.py`

```
def trailing_edge(x):
        """return trailing True values in list of booleans"""
        out = []
        status = x[0]
        for i in x:
                out.append(not i and i!=status)
                if i != status:
                        status = i
        out.append(x[-1])
        return(out[1:])
```

5.31 unnormalize_procedure(procedure)

collate procedure times into single day, if relative to the same day

• Source code in `td/td.py`

```
def unnormalize_procedure(procedure):
        """ collate procedure times into single day, if relative to the same day"""
        out = []
        if procedure:
                rels = set([r for (d, ts, r) in procedure])
                for rel in rels:
                        current_times = []
                        for (d, ts, r) in procedure:
                                for t in ts:
                                        if r==rel:
                                                current_times.append(t+(d-r)*24)
                        out.append((rel, current_times, rel))
        return(out)
```