TimeFlies – A Tool for Time Tracking

Jörg Bullmann <jb@heilancoo.net>

November 17, 2012

User Manual and Reference
TimeFlies 0.6

Contents

1 What is *TimeFlies* About?

Do you want to account for the time you spend at work? What project or work package have you been working on? Do you want to keep track of the hours you work? How much leave have you got left for this year?

Do you keep a daily work journal containing things you did, problems you solved, some kind of to do list?

Do you want to make an estimate of effort for a project or work package? Would you like to break down those things into smaller items and possibly break down those again too?

TimeFlies can help you with this – and this is how: you keep a journal of what you do using *TimeFlies*' hopefully not too overbearing markup syntax. *TimeFlies* can later distill from your journal files what you did when, how much you worked and on which projects.

TimeFlies assumes you can tell what you are doing on a daily basis and it relies on you entering sound data for what you do during the day. It offers a light weight syntax for this. It does not, however, have fine grained time keeping functionality such as stop watch like features some other time keepers have. In that sense *TimeFlies* is targeted at a slightly higher level of time tracking. Maybe call it *macro* level time tracking instead of *micro* level time tracking?

2 Tutorial by Example

In this section we will look at a number of use cases. All *TimeFlies* data is kept in plain text files. So all your data is always easily accessible to you and the format itself is quite human-readable. Moreover it can easily be version controlled.

2.1 Recording Time

To record your work time keep a work log file with `day` lines specifying the dates and in and out times telling when you arrived at work and when you left. The times can be given in decimals or in hours and minutes:

```
day 2012-09-19 8:30 17:15
day 2012-09-18 8.75 17.75
day 2012-09-17 8 18
```

Do you need to account for breaks you are taking? Use `off` instructions to state periods of time in a day of work during which you were not actually working:

```
day 2012-09-19 8.5 17.25, off 0.5
day 2012-09-18 8.75 17.75, off 0.75
day 2012-09-17 8 18, off 0.5, off 0.25
```

This last file is equivalent to the following:

```
day 2012-09-19 8.5 17.25
off 0.5
day 2012-09-18 8.75 17.75
off 0.75
day 2012-09-17 8 18
off 0.5
off 0.25
```

Above example illustrates the notion of a `day`-*block*: a `day`-*block* extends from one `day`-keyword to the next and everything inside this day block is part of that day.

The days in the file do not need to be listed chronologically. You could e.g. list the days in reverse order so that the present is always at the top of the file.

If you want to mask out part of your log temporarily you can use the *# source comment* marker. *TimeFlies* ignores the # and everything following it until the end of line. It works just the same as e.g. a Python comment.

```
day 2012-09-19 8.5 17.25
off 0.5

# day 2012-09-18 8.75 17.75
# off 0.75

day 2012-09-17 8 18
off 0.5
off 0.25
```

2.2 Keeping Notes in the Log

Do you want to keep notes about your work in the same place as you keep the time information? Use *log comment* lines like in this file:

```
day 2012-09-19 8.5 17.25
; updated regression tests
off 0.5
; fixed build scripts

day 2012-09-18 8.75 17.75
; wrote unit test to reproduce problem report 2012-0098
; fixed problem report 2012-0098
off 0.75
; added HTML output option to object dumper
; discussed implications of Java 1.7 rollout

day 2012-09-17 8 18
; weekly team meeting
off 0.5
; monthly quality task force
off 0.25
; code review: server side includes
```

A log comment line starts with a semicolon and one or more space characters. All text following these characters until the end of the line (or until a # source comment marker) with trailing spaces removed constitute the recorded log comment.

Now what can you do with such a file? Assume the above work log file's name is `work-log.fly`, then option `-t` tells *TimeFlies* to calculate your work times.

```
> timeflies -t work-log.fly
Time at work overview (all):
     when        worked   leave    sick balance
2012-09-17 Mon:    9.25 ----.-- ----.--    1.25
2012-09-18 Tue:    8.25 ----.-- ----.--    0.25
2012-09-19 Wed:    8.25 ----.-- ----.--    0.25
  week 2012-38:   25.75 ----.-- ----.--    1.75
 month 2012-09:   25.75 ----.-- ----.--    1.75
         total:   25.75 ----.-- ----.--    1.75
     when        worked   leave    sick balance
```

To include the log comments in this output, use the `-C` option:

```
> timeflies -t -C work-log.fly
Time at work overview (all):
     when        worked   leave    sick balance
2012-09-17 Mon:    9.25 ----.-- ----.--    1.25
               ; weekly team meeting
               ; monthly quality task force
               ; code review: server side includes
2012-09-18 Tue:    8.25 ----.-- ----.--    0.25
               ; wrote unit test to reproduce problem report 2012-0098
               ; fixed problem report 2012-0098
               ; added HTML output option to object dumper
               ; discussed implications of Java 1.7 rollout
2012-09-19 Wed:    8.25 ----.-- ----.--    0.25
               ; updated regression tests
               ; fixed build scripts
  week 2012-38:   25.75 ----.-- ----.--    1.75
 month 2012-09:   25.75 ----.-- ----.--    1.75
         total:   25.75 ----.-- ----.--    1.75
     when        worked   leave    sick balance
```

2.3 Time Summaries

Here's a longer example where you can see the use of weekly and montly summaries. Also a few days of annual leave and sickness are inserted using the `leave-days` and `sick` instructions.

```
day 2012-08-23 8.5 17.25, off 0.5
day 2012-08-24 8.75 17.75, off 0.75
day 2012-08-27 8.5 17.25, off 0.5
day 2012-08-28 8.75 17.75, off 0.75
day 2012-08-29 8 18, off 0.5, off 0.25
```

```
leave-days 2012-08-30 2012-09-04; Hiking
day 2012-09-05 8.75 17.75, off 0.75
day 2012-09-06 8 18, off 0.5, off 0.25
day 2012-09-07 8.5 17.25, off 0.5
day 2012-09-10 8.75 17.75, off 0.75
day 2012-09-11, sick 8; Broken arm
day 2012-09-12, sick 8; Broken leg
day 2012-09-13 8.5 17.25, off 0.5
day 2012-09-14 8 18, off 0.5, off 0.25
day 2012-09-17 8 18, off 0.5, off 0.25
day 2012-09-18 8.5 17.25, off 0.5
```

Use option `-f week` to get an overview of weekly work time balances.

```
> timeflies -t -f week work-log.fly
Time at work overview (week):
     when        worked   leave    sick balance
  week 2012-34:   16.50 ----.-- ----.--    0.50
  week 2012-35:   25.75   16.00 ----.--    1.75
  week 2012-36:   25.75   16.00 ----.--    1.75
  week 2012-37:   25.75 ----.--   16.00    1.75
  week 2012-38:   17.50 ----.-- ----.--    1.50
         total:  111.25   32.00   16.00    7.25
     when        worked   leave    sick balance
```

Or have both weekly and monthly balances shown.

```
> timeflies -t -f week,month work-log.fly
Time at work overview (week, month):
     when        worked   leave    sick balance
  week 2012-34:   16.50 ----.-- ----.--    0.50
 month 2012-08:   42.25   16.00 ----.--    2.25
  week 2012-35:   25.75   16.00 ----.--    1.75
  week 2012-36:   25.75   16.00 ----.--    1.75
  week 2012-37:   25.75 ----.--   16.00    1.75
  week 2012-38:   17.50 ----.-- ----.--    1.50
 month 2012-09:   69.00   16.00   16.00    5.00
         total:  111.25   32.00   16.00    7.25
     when        worked   leave    sick balance
```

Maybe you only want to look at one month with daily details? Note the comments that have been associated with the leave and the sick days in the input file show up in the respective daily output lines.

```
> timeflies -t -f 2012-09 work-log.fly
Time at work overview (2012-09):
     when        worked   leave    sick balance
  week 2012-35: ----.-- ----.-- ----.-- ----.--
2012-09-03 Mon: ----.--    8.00 ----.-- ----.-- Hiking
2012-09-04 Tue: ----.--    8.00 ----.-- ----.-- Hiking
2012-09-05 Wed:    8.25 ----.-- ----.--    0.25
2012-09-06 Thu:    9.25 ----.-- ----.--    1.25
2012-09-07 Fri:    8.25 ----.-- ----.--    0.25
```

```
  week 2012-36:   25.75   16.00 ----.--    1.75
2012-09-10 Mon:    8.25 ----.-- ----.--    0.25
2012-09-11 Tue: ----.-- ----.--    8.00 ----.-- Broken arm
2012-09-12 Wed: ----.-- ----.--    8.00 ----.-- Broken leg
2012-09-13 Thu:    8.25 ----.-- ----.--    0.25
2012-09-14 Fri:    9.25 ----.-- ----.--    1.25
  week 2012-37:   25.75 ----.--   16.00    1.75
2012-09-17 Mon:    9.25 ----.-- ----.--    1.25
2012-09-18 Tue:    8.25 ----.-- ----.--    0.25
  week 2012-38:   17.50 ----.-- ----.--    1.50
 month 2012-09:   69.00   16.00   16.00    5.00
         total:   69.00   16.00   16.00    5.00
    when        worked   leave    sick balance
```

You only need weekly totals in that one month?

```
> timeflies -t -f week,2012-09 work-log.fly
Time at work overview (week, 2012-09):
     when        worked   leave    sick balance
  week 2012-35: ----.-- ----.-- ----.-- ----.--
  week 2012-36:   25.75   16.00 ----.--    1.75
  week 2012-37:   25.75 ----.--   16.00    1.75
  week 2012-38:   17.50 ----.-- ----.--    1.50
         total:   69.00   16.00   16.00    5.00
    when        worked   leave    sick balance
```

2.4 Logging Activities

Log comments are a good way to keep track of things you don't want to forget and have accessible and also aligned with your work time line. Log comments have no work effort assigned to them, though. So you cannot use them in any way for calculations of effort spent.

You use *work packages* and *activities* to connect the time you work with the work packages you work on: first, you define your work packages, then you use *activity* lines in the day blocks instead of log comment lines.

A work package definition is a line starting with the keyword `work-package` (or its abbreviation `wp`) followed by a work package name.

An activity line starts with a single dash character `-` followed by one or more spaces. This is followed by a work package id and a duration. This is optionally followed by a semicolon and some activity comment.

See below the converted example work log file.

```
wp regression-tests
wp meetings
wp quality-task-force
wp problem-reports
wp development
wp other

```

```
day 2012-09-19 8.5 17.25
- regression-tests 4; updated
off 0.5
- other 3.5; fixed build scripts

day 2012-09-18 8.75 17.75
- problem-reports 2; wrote unit test to reproduce problem report 2012-0098
- problem-reports 2.5; fixed problem report 2012-0098
off 0.75
- development 3; added HTML output option to object dumper
- other 1; discussed implications of Java 1.7 rollout

day 2012-09-17 8 18
- meetings 2.0; weekly team meeting
off 0.5
- quality-task-force 6
off 0.25
- other 1.25; code review: server side includes
```

Option `-w` tells *TimeFlies* to calculate the times you have been working on the different work packages:

```
> timeflies -w work-log.fly
Work package summary (all):
  25.25 : ALL
       4.00 : regression-tests
       2.00 : meetings
       6.00 : quality-task-force
       4.50 : problem-reports
       3.00 : development
       5.75 : other
```

To also show the activities contributing to the different work packages, use option `-a`:

```
> timeflies -w -a work-log.fly
Work package summary (all):
  25.25 : ALL
       4.00 : regression-tests
              - 2012-09-19 4.0; updated
       2.00 : meetings
              - 2012-09-17 2.0; weekly team meeting
       6.00 : quality-task-force
              - 2012-09-17 6.0
       4.50 : problem-reports
              - 2012-09-18 2.0; wrote unit test to reproduce problem report 2012-0098
              - 2012-09-18 2.5; fixed problem report 2012-0098
       3.00 : development
              - 2012-09-18 3.0; added HTML output option to object dumper
       5.75 : other
              - 2012-09-17 1.25; code review: server side includes
              - 2012-09-18 1.0; discussed implications of Java 1.7 rollout
              - 2012-09-19 3.5; fixed build scripts
```

To check whether you have allocated all your working time to work packages, use option `-c`:

```
> timeflies -c work-log.fly
Day check (all):
2012-09-18 Tue: worked  8.25, allocated  8.50, delta  0.25
2012-09-19 Wed: worked  8.25, allocated  7.50, delta -0.75
2 problems detected.
```

This shows that on two days the time at work and the time worked on work packages are differing.

2.5 Work Package Breakdown

In the previous section, work packages have been defined as simple, atomic, named items. A work package can be subdivided and refined hierarchically. See the following example.

```
wp md; MightyDigester: digests inputs of all sorts
    in; read supported input formats
        xml
        json
        dottxt; dotted text format
        binary
    proc; processing modules
        stats; processing statistics
        phase-1; rough break-down
        phase-2; particle recombination
        phase-3; regrouping and amalgamation
    out; write supported output formats
        xml
        json
        text
        binary
    mmi
        gui
        cmdline
```

The items in this work package hierarchy can be referred to in activity lines as dot-delimited work package path names.

Following, a piece of work log for the above project.

```
day 2012-07-01 8 17, off 1
- md.in.xml 4; updated to new XSD
- md.out.xml 3; updated to new XSD
- md.mmi.cmdline 1; XML options
day 2012-07-02 8 17, off 1
- md.in.json 5; first minimal implementation
- md.proc.stats 1.5; line counting
- md.mmi.cmdline 1.5; statistics options
day 2012-07-03 8 17, off 1
```

```
- md.in.xml 4; adapted includes
- md.out.xml 3; normalised host node structure
- md.out.text 0.5; don't use TAB any more
- md.mmi.cmdline 0.5; text and xml options
```

Assume file `prj-mighty-digester.fly` contains the work package defintions and the work log itself is kept in `work-log.fly`. The work package summary can be calculated with option `-w` (which was also used in the previous example).

```
> timeflies -w prj-mighty-digester.fly work-log.fly
Work package summary (all):
  24.00 : ALL
      24.00 : md; MightyDigester: digests inputs of all sorts
          13.00 : in; read supported input formats
               8.00 : xml
               5.00 : json
           1.50 : proc; processing modules
               1.50 : stats; processing statistics
           6.50 : out; write supported output formats
               6.00 : xml
               0.50 : text
           3.00 : mmi
               3.00 : cmdline
```

And here the same with activities shown.

```
> timeflies -w -a prj-mighty-digester.fly work-log.fly
Work package summary (all):
  24.00 : ALL
      24.00 : md; MightyDigester: digests inputs of all sorts
          13.00 : in; read supported input formats
               8.00 : xml
                      - 2012-07-01 4.0; updated to new XSD
                      - 2012-07-03 4.0; adapted includes
               5.00 : json
                      - 2012-07-02 5.0; first minimal implementation
           1.50 : proc; processing modules
               1.50 : stats; processing statistics
                      - 2012-07-02 1.5; line counting
           6.50 : out; write supported output formats
               6.00 : xml
                      - 2012-07-01 3.0; updated to new XSD
                      - 2012-07-03 3.0; normalised host node structure
               0.50 : text
                      - 2012-07-03 0.5; don't use TAB any more
           3.00 : mmi
               3.00 : cmdline
                      - 2012-07-01 1.0; XML options
                      - 2012-07-02 1.5; statistics options
                      - 2012-07-03 0.5; text and xml options
```

3 Reference

3.1 Command Line Syntax

This section explains all of *TimeFlies*' command line options and arguments. Different combinations of these allow you to run consistency checks on the given input files or to generate the output reports from them.

3.1.1 Show Work Packages

...

3.1.2 Calculate Work Packages

...

3.1.3 Tally Days

...

3.1.4 Check Days

Use command line option `-c` or `--check` to check the plausibility of the entered work log data for each day in the input. The check will flag the following problems:

Activities vs. Work Time If you use work packages and activities to break down your daily work, for each day the sum of all your activities will be compared to your given time at work. There should be no difference.

Leave and Sick Times vs. Required Work Time If you have declared more `leave` or `sick` time for a day than the required working hours of that day, this will be flagged.

3.1.5 Filters

Filters are used with *TimeFlies* to restrict the processing or output generation. The command line option for filters is `-f` or `--filter` Multiple filters can be combined in form of a comma separated list.

Examples:

```
> timeflies -c -f 2012-08 work-log.fly # check August 2012
> timeflies -t -f 2012-08,week work-log.fly # tally days for August 2012
                                            # with weekly summaries
```

Time Filters Without any *time filter*, all given input is processed.

To limit processing to certain time periods, use a time filter. A *month filter* is of the form `<YYYY>-<MM>` and limits processing to the specified month. A *time range filter* is of the form `<YYYY>-<MM>-<DD>..<YYYY>-<MM>-<DD>` and limits processing to the given time range. The time range extends from the beginning of the first given day to the end of the last given day.

Summary Filters Time at work overviews generated using the `-t` (or `--tally-days`) option can include daily, weekly and monthly summary records. If no *summary filter* keyword is given, then all three are assumed per default.

Use a summary filter of `day`, `week` or `month` (or any combination) to reduce the generated output. If at least one of those summary filters is given, *TimeFlies* will only generate output for the given filter or filters.

Examples:

```
> timeflies -t -f 2012-08,week work-log.fly # tally days for August 2012
                                            # with weekly summaries
> timeflies -t -f week,month work-log.fly # tally days for all input with
                                          # weekly and monthly summaries
```

3.1.6 Output Indentation

When generating output of work package hierarchies per default each hierarchy level is indented by 4 space characters. To change that indentation use option `-i <num>` or `--indent <num>`.

3.2 File Syntax

This section describs the *TimeFlies* input file syntax and explains of the associated semantics.

3.2.1 Source Comments and Persistent Comments

TimeFlies knows two kinds of comments: *source comments* and *persistent comments*.

Source Comments These are marked by a hash sign (#) and extend from it to the end of the line. *TimeFlies* ignores these comments and treats them as if they did not exist.

Persistent Comments These are marked by a semicolon (`;`) and extend to the end of the line (or a possibly following source comment in that line). A persistent comment is processed and will show up in the generated output. E.g. comments on leave or sick days will show up in the time at work output summaries. Comments on activities or work packages can show up in the work package outputs.

3.2.2 Time and Date Formats

Times and time durations can generally be given in *[h]h:mm* format or in decimals. Examples would be `8:30` or `8.5`. Times use the 24-h-system. So 5:15 p.m. would therefore have to be written as either `17:15` or `17.25`.

Dates must generally be written in *yyyy-mm-dd* format.

3.2.3 Work Packages

Syntax: `work-package <wpid> [; <comment>]`. The keyword `work-package` can be abbreviated as `wp`.

The fully qualified work package id `wpid` is a dot-delimited sequence of simple work package ids (tokens consisting of alphanumeric characters). It resembles a path from the root of the work package hierarchy. In an compound id *a.b* the id *b* appearing directly to the right of *a* means that work package *b* is an immediate sub work package of *a*. Work package hierarchies can therefore be given as sequences of work package definitions.

Alternatively (and more concise) a work package hierarchy can be given as hierarchically indented text (similar to Python indentation rules). In this case full work package ids are not necessary and simple ones suffice.

Illustrating this, the follwing example:

```
wp pro; project of some sort
wp pro.aaa; part aaa
wp pro.bbb; part bbb
wp pro.bbb.xxx; detail xxx
wp pro.bbb.yyy; detail yyy
wp pro.ccc; part ccc
```

is equivalent to:

```
wp pro; project of some sort
    aaa; part aaa
    bbb; part bbb
        xxx; detail xxx
        yyy; detail yyy
    ccc; part ccc
```

This concise form of work package definition can also be applied partially. So the following is another form, equivalent to the above two:

```
wp pro; project of some sort
    aaa; part aaa
    bbb; part bbb
    ccc; part ccc
wp pro.bbb
    xxx; detail xxx
    yyy; detail yyy
```

When defining a work package hierarchy using indentation, you must take care to not mix tab and space characters. I recommend using space characters exclusively. They are displayed the same way, no matter what tool you use to edit or view your files. If you really cannot stop yourself, *TimeFlies* will let you use tab characters. Never mix tabs and spaces, however, *TimeFlies* will not accept this.

To illustrate this tab/space subject matter, the following examples show these normally non-printable characters.

1. A good example, exclusively using space characters:

```
wp␣pro
␣␣␣␣aaa
␣␣␣␣bbb
␣␣␣␣␣␣␣␣xxx
␣␣␣␣␣␣␣␣yyy
␣␣␣␣ccc
```

2. An acceptable example, exclusively using tab characters:

```
wp␣pro
→aaa
→bbb
→   →xxx
→   →yyy
→ccc
```

3. A mildly annoying example where the first work package hierarchy uses spaces and the second one uses tabs:

```
wp␣pro1
␣␣␣␣sub1
␣␣␣␣sub2
␣␣␣␣␣␣␣␣sub2a
␣␣␣␣␣␣␣␣sub2b
␣␣␣␣sub3

wp␣pro2
→aaa
```

```
10      ⇥bbb
11      ⇥   ⇥xxx
12      ⇥   ⇥yyy
13      ⇥ccc
```

4. A bad example, rejected by *TimeFlies*. In line 4 both spaces and a tab character are used:

```
1  wp␣pro
2  ␣␣␣␣aaa
3  ␣␣␣␣bbb
4  ␣␣␣␣  ⇥xxx
5  ␣␣␣␣␣␣␣␣yyy
6  ␣␣␣␣ccc
7
```

   This is what *TimeFlies* has to say about it:

```
1  > timeflies bad-wp-def.fly
2  bad-wp-def.fly:4: ERROR : indentation contains both spaces and tabs
3  1 error.
```

5. Another bad example that *TimeFlies* will reject. Lines 2 through to 4 are indented with spaces, line 5 and 6 use tabs:

```
1  wp␣pro
2  ␣␣␣␣aaa
3  ␣␣␣␣bbb
4  ␣␣␣␣␣␣␣␣xxx
5     ⇥   ⇥yyy
6     ⇥ccc
7
```

   And *TimeFlies* says:

```
1  > timeflies bad-wp-def.fly
2  bad-wp-def.fly:5: ERROR : work package indentation error
3  bad-wp-def.fly:6: ERROR : work package indentation error
4  2 errors.
```

3.2.4 Day blocks

Syntax: `day <date> [<in> <out>]`

A day block starts with a the keyword `day` followed by the day's date and optionally the arrival and leaving times at work which, when given, define the day's *working period*. Either both times must be given or none at all. If they are not given, no time at work is assumed for that day. This is used e.g. for leave days or sick days. A day block ends at the next `day` keyword, i.e. at the beginning of a new day block.

3.2.5 Activities

Syntax: `- <wpid> <time> [; <comment>]`

An *activity* is a period of time spent working on a work package. An activity must appear in a day block. The work package id `<wpid>` must be a valid fully qualified work package id, i.e. a matching work package must have been defined before.

An activity can only appear in a day block.

3.2.6 Time off Work

Syntax: `off <time> [; <comment>]`

Declare times off work in a day block, yet inside the time span of that day.

An `off` statement can only appear in a day block.

3.2.7 Sickness

Syntax: `sick [<time>] [; <comment>]`

Declare sick time for a day, yet outside the day's working period. If `<time>` is not given, then the day's required working time is assumed, i.e. a full day of sick leave.

A `sick` statement can only appear in a day block.

3.2.8 Block Sickness (Several Days)

Syntax: `sick-days <first> <last> [; <comment>]`

This is a short form to define an extended period of sickness, i.e. several days. The sickness period starts on day `<first>` and ends on (and includes) day `<last>`.

3.2.9 Leave

Syntax: `leave [<time>] [; <comment>]`

Declare leave for a day, yet outside the day's working period. Use this for leave periods in order of hours. If `<time>` is not given, then the day's required working time is assumed, i.e. a full day of leave.

A `leave` statement can only appear in a day block.

3.2.10 Block Leave (Several Days)

Syntax: `leave-days <first> <last> [; <comment>]`

This is a short form to define an extended period of leave, i.e. several days. The leave period starts on day `<first>` and ends on (and includes) day `<last>`.

3.2.11 Public Holidays

Syntax: `public-holiday [; <comment>]`. The keyword `public-holiday` can be abbreviated as `phol`.

The current day is marked as a public holiday, i.e. a day where the required time at work is null.

A `public-holiday` statement can only appear in a day block.

3.2.12 Required Work Hours

Syntax: `must-hours ((<day>|<start>..<end>)=<hours>)*` where `<day>`, `<start>` and `<end>`, must one of `mon`, `tue`, `wed`, `thu`, `fri`, `sat` or `sun`.

This lets you define the times you need to work on the different days of the week. A working time arrangement consists of the hours of work for all days of a week. For one single working time arrangement, all hours have to be given in one single `must-hours` statement. Day ranges using the `<start>..<end>=<hours>` notation and single day specifications using the `<day>=<hours>` notation can be freely combined.

If a day is not covered in a `must-hours` statement, the hours for that day are assumed to be 0. Thus, an empty `must-hours` statement (not containing any day range or single day item) is perfectly valid. This could e.g. be used to declare a period of sabbatical.

To change the working time arrangements (e.g. to reflect going from full time to 80% or back), you use multiple `must-hours` statements.

The first following example defines a standard 40 hour working week (which is the default arrangement, also assumed for any day block with no applicable `must-hours`):

```
must-hours mon..fri=8.0
```

This example defines an 80% working time week with Mondays off:

```
must-hours tue=8.0 wed=8.0 thu=8.0 fri=8.0
```

Or you work Thursdays and Fridays only half days:

```
must-hours mon..wed=8.0 thu..fri=4.0
```

The last example shows multiple changes in working time arrangement:

```
day 2012-01-01
must-hours tue..fri=8.0

# ...

day 2012-06-01
```

```
must-hours mon=8.0 tue=8.0 wed=8.0 thu=4.0 fri=4.0

# ...

day 2012-09-01
must-hours mon=8.0 tue=8.0 wed=8.0 thu=8.0 fri=8.0

# ...
```

If a `must-hours` statement appears inside a day block, the given working times are applicable from the date of that day block onwards until one day prior to the date of the chronologically next day block with a `must-hours` statement. That `must-hours` statement will be applicable from then onwards.

If a `must-hours` statement appears outside a day block (before the very first day block in the input), then the given working times are applicable until one day prior to the chronologically first day block with a `must-hours` statement. That `must-hours` statement will be applicable from then onwards.

In above explanation of applicability the word *chronologically* is very important because day blocks in *TimeFlies* do not need to appear in chronological order in the input. The applicability of the `must-hours` statements is determined by the chronological order, though, which is not necessarily in the same order as the statements' order of appearance in the input file(s).

3.2.13 Importing Files

Syntax: `import <file>`.

The named file is imported. This means the file's content is processed in the same way as if it had appeared in the importing file instead of the import statement.

This allows you to e.g. split your daily logs by month, separate work package definitions from daily logs or share work package definitions amongst a group of users while keeping daily logs private.

One single work package definition (a single `work-package` statement in abbreviated hierarchical form using white space indentation) cannot be broken up across input file boundaries. To combine a work package hierarchy from multiple input files each input file must contain at least one complete `work-package` statement each.

Invalid example:

```
# File 1: looks good
wp pro; project of some sort
    aaa; part aaa
    bbb; part bbb
```

```
# File 2; bad, does not start with a work-package statement
        xxx; detail xxx
        yyy; detail yyy
    ccc; part ccc
```

Correct form:

```
# File 1: same as in above example
wp pro; project of some sort
    aaa; part aaa
    bbb; part bbb
```

```
# File 2: correct, starts with a work-package statement
wp pro
    bbb
        xxx; detail xxx
        yyy; detail yyy
    ccc; part ccc
```