# dipplanner Documentation

***Release 0.3nightly***

**Thomas Chiroux**

September 18, 2012

# CONTENTS

**Warning:** This software is provided free of charge for experienced divers to evaluate diving profiles. This software is highly experimental, it probably contains bugs and should not be relied upon for actual dives. Use it at your own risk.
If you follow dive schedules generated by this software you could suffer decompression sickness, serious injury or possibly death. The author does not warrant that this software accurately reflects Buhlmann's algorithms, or that it will produce safe, reliable, results.
Diving in general is fraught with risk, and decompression diving using mixed gases and rebreathers adds significantly more risk. This software is not intended for uneducated users. This software and the decompression schedules it produces are tools for experienced divers only.
IF YOU DO NOT UNDERSTAND OR DO NOT AGREE TO THIS STATEMENT DO NOT USE THIS SOFTWARE

CHAPTER

ONE

# DIPPLANNER QUICK START

**Warning:** This software is provided free of charge for experienced divers to evaluate diving profiles. This software is highly experimental, it probably contains bugs and should not be relied upon for actual dives. Use it at your own risk.
If you follow dive schedules generated by this software you could suffer decompression sickness, serious injury or possibly death. The author does not warrant that this software accurately reflects Buhlmann's algorithms, or that it will produce safe, reliable, results.
Diving in general is fraught with risk, and decompression diving using mixed gases and rebreathers adds significantly more risk. This software is not intended for uneducated users. This software and the decompression schedules it produces are tools for experienced divers only.
IF YOU DO NOT UNDERSTAND OR DO NOT AGREE TO THIS STATEMENT DO NOT USE THIS SOFTWARE

## 1.1 Description

### 1.1.1 Model supported

- Bülhmann ZH-L16B with Gradient Factor
- Bülhmann ZH-L16C with Gradient Factor

### 1.1.2 Gases

any mix of oxygen, helium and nitrogen is supported, this include:

- Air
- Nitrox
- Oxygen
- Trimix
- Heliox
- ...

### 1.1.3 Other Features

- Open Circuit Dives
- CCR Dives
  - setpoint
  - bailout
- Tank planification
  - Tank size and pressure
  - Simple or Double Tank
  - gaz consumption rate
  - rules for warning (rules of third, rules in remaining pressure, etc..)
  - automatic (or non automatic) refill of Tank between successive dives
- Multigaz (multi Tank)
- Automatic selection of best gaz
- Multilevel Dives
- Altitude Dives
- Successive Dives
- No Flight Time Calculation
- CNS Tracking
- OTU Tracking
- MOD Calculation
- END Calculation
- Adjustable gradient factor and stop depth level

### 1.1.4 Calculations

The main principe of dipplanner calculation are to be as accurate as possible, using the most precise algorithm.

- All internal calculation in SI Units
- Most precise calculations (trying to avoid any approximation):
  - second to second decompression time calculation (adjustable)
  - real water density calculation
  - real gaz pressure calculation in tanks using van der waals
  - real surface pressure calculation (using Antoine equation for ppH2O)
  - ...
- Every parameter is adjustable

**Warning:** in result of the "precise" calculations, the planned dives may be shorter than other equivalent dive planification programs.
Users of dipplanner MUST be aware of this difference and adjust the conservatisms to their wishes.
On the other way, with the most precise calculations, dipplanner try to avoid unexpected effects (or worse: unseen effects) of approximations. (approximation effect to conservatism are sometimes counter-intuitive and may in some cases reduce the conservatism without warning)

## 1.2 Get the sources

latest source code (unstable)

please use git to clone the repository :

```
git clone git://github.com/ThomasChiroux/dipplanner.git
```

### 1.2.1 Preparation: virtualvenv

It is recommended (but optional) to setup a virtualvenv before running or installing. In this way, you will not break or change your real environment. So after getting the source, go in the dipplanner directory:

```
cd dipplanner
mkdir venv
virtualenv venv
source venv/bin/activate
```

## 1.3 Installing

Inside the dipplanner source directory, run:

```
python setup.py install
```

## 1.4 Using

This version if currently only usable in command line

### 1.4.1 Run the program

ex:

```
dipplanner --help
```

### 1.4.2 Planning one dive

To plan a dive, you should at least provide one tank and one dive segment. Here is below a sample for a 12l tank with 200b of air and a dive of 25min at 30m

```
dipplanner -t "airtank;0.21;0.0;12;200;50b" -s "30;25*60;airtank;0.0"
```

You can provide more than one tank and of course multiple segments (they will be processed in the order you provided it) Deco tanks will be automaticaly choosen if appropriate. Here is below a sample for a trimix dive : bi 12l-cylinder of Tx21/30 and Deco Nx80 (S80), 50m - 20mins:

```
dipplanner -t "tx;0.21;0.30;24;200;50b" -t "deco;0.8;0.0;12;200;50b" -s "50;20*60;tx;0.0"
```

### 1.4.3 Change some parameters of the dive

See

```
dipplanner --help
```

output to see all available parameters

You can also read *dipplanner command-line documentation*

### 1.4.4 Config files

You can provide one or several config files to the program. The config file can overide any default parameter. see config/default_config.cfg for all the details

parameter are changed using this order:

1. default parameter
2. parameter set in config files
3. parameter set in command line

You can also read *dipplanner config file documentation*

## 1.5 Units : SI or Imperial

dipplanner uses only SI unit internally. However a config parameter or a command line parameter can change this behaviour. and use imperial Units

**if imperial unit is set in a config file :** all the parameters from this config file and all the config files read after will be considered imperial (including command line parameters) But all the parameter in previous config files will still be considered as SI All the ouput will be done in imperial units

**if imperial unit is set in command line :** all the parameter given in command line will be considered imperial, but not the parameters eventually given using config files All the output will be done in imperial units

SI and imperial unit converter uses the following correspondances:

- bar <–> psi
- liter <–> cubic feet
- meter <–> feet

## 1.6 References

- at first this program is a python rewrite of MV-Plan a dive planning tool written in java by Guy Wittig
- ref used for programming and understand algorithms.

## 1.7 Open Source and Licence

CHAPTER

TWO

# DIPPLANNER COMMAND-LINE DOCUMENTATION

when invoking dipplanner, you may specify any of these options:

```
dipplanner [-h] [--help]
           [-c] [--config]
           [--surfaceinterval]
           [--model]
           [--gflow]
           [--gfhigh]
           [--water]
           [--altitude]
           [--diveconsrate]
           [--decoconsrate]
           [--descentrate]
           [--ascentrate]
           [--maxppo2]
           [--minppo2]
           [--maxend]
           [--samegasfordeco]
           [--forcesegmenttime]
           [--depthcalcmethod]
           [--travelswitch]
           [--surfacetemp]
           [--ambiantpressureatsea]
           [--template]
           [-t] [-tank]
           [-s] [-segment]
```

Either presence of tank and segment inside a config file or in command line are needed for this program to run

## 2.1 help

**-h, -help**
show this help message and exit

## 2.2 config files

**-c** <STRING>, **-config** <STRING>
path for config file.

Default : ./config.cfg

see *dipplanner config file documentation* for more informations on config files

---

**Note:** Multiple config files MAY be loaded by providing multiple -c options.

each config file may contain different parameter, but some parameters may also appear in multiple config files. In that case, the last occurence of that parameter is used.

except of tanks and segments: all tanks and segments provided are used for the dives (see config-file documentation for more infos)

---

## 2.3 tanks and segments

**-t** <STRING>, **-tank** <STRING>
specify a tank that will be used for the dive

Format:

```
"tank_name;f_o2;f_he;Volume(l);Pressure(bar);Minimum gas rule"
```

- tank_name: (str) (you choose the name) for the tank
- f_02: (float) fraction of oxygen in the tank. Between 0.0 and 1.0
- f_he: (float) fraction of helium in the tank. Between 0.0 and 1.0
- Volume: (float) Volume of the tank in bar
- Pressure: (float) Pressure of the tank in bar
- Minimum gas rule: (str) quantity of gas that should remain in the tank after the dive

There two format for minimum gas rule:

- quantity of bar that should remain in the tank:

  format: "[0-9]+b"

  ex: "50b": it should remain 50 bar in the tank at the end of the dive

- "fraction rule" (like the rule of third in cave diving)

  format: "1/[0-9]"

  ex1: "1/3" : 1/3 of the tank to go in, 1/3 of the tank to go back and it should remain 1/3 of the tank at the end of the dive

  ex2: "1/6" : 1/6 of the tank to go in, 1/6 of the tank to go back and it should remain 2/3 of the tank at the end of the dive

Example:

12l tank filled with 200b or air. It should remain 50b at the end of the dive.

```
"airtank;0.21;0.0;12;200,50b"
```

---

**Note:** Multiple tanks may be provided

ex:

dipplanner -t “airtank;0.21;0.0;12;200,50b” -t “nitrox;0.80;0.0;12;200;50b”

---

**-s** <STRING>, **-segment** <STRING>
Input segments used for the dive

Format:

```
"depth;duration;tank;setpoint"
```

- depth: (float) in meter
- duration: (float) in seconds (operators are allowed like: ‘30 * 60’)
- tank: name of the tank (the ‘tank_name’ specified in -t option)
- setpoint: (float) 0.0 if OC, setpoint if CCR

---

**Note:** If you specify a setpoint > 0.0, the dive will automatically switch in CCR mode.

---

Example:

20 min at 30 meter using tank: airtank in OC mode

```
"30;20*60;airtank;0.0"
```

---

**Note:** You can specify multiple segments

ex:

dipplanner -s “30;1000;airtank;0.0” -s “20;800;airtank;0.0”

---

## 2.4 dive parameters

**-surfaceinterval**=<VALUE>
Optional Surface Interval in seconds

If provided, dipplanner will calculate a surface decompression before diving.

Example:

One hour of surface interval

```
dipplanner --surfaceinterval=3600
```

**-model**=<ZHL16b|ZHL16c>
Set the decompression model used for the calculations: either buhlmann ZHL16b or buhlmann ZHL16c

Default: ZHL16c

Example:

```
dipplanner --model=ZHL16b
```

**-gflow**=<VALUE>
GF low: (int) in %, between 0 and 100

Default: 30%

Example:

GF low of 25%

```
dipplanner --gflow=25%
```

---

**Note:** Internally, GFlow is a float number between 0.0 and 1.0, but for convenience, the argument in command line is provided in % value, between 0 and 100. The conversion is done automatically.

---

**-gfhigh**=<VALUE>
GF high: (int) in %, between 0 and 100

Default: 80%

Example:

GF high of 85%

```
dipplanner --gfhigh=85%
```

---

**Note:** Internally, GFhigh is a float number between 0.0 and 1.0, but for convenience, the argument in command line is provided in % value, between 0 and 100. The conversion is done automatically.

---

**-water**=<sea|fresh>
specify in which type of water you will do the dive: sea or fresh

Default: sea

Example:

Do a dive in a lake

```
dipplanner --water=fresh
```

**-altitude**=<VALUE>
altitude (int) of the dive in meter.

**Warning:** It's very important to specify this parameter if you do a dive in altitude

Default: 0m (sea level)

Example:

Dive at 1400m

```
dipplanner --altitude=1400
```

**-diveconsrate**=<VALUE>
gas consumption rate (float) during dive (in l/minute).

Is it used for tank monitoring and associated with tank size, pressure and tank rules, it will warn you if your plannified dive ends without enough gas.

Default: 17 l/min

Example:

Plan a dive with 25 l/min dive consumption rate

```
dipplanner --diveconsrate=25
```

---

**Note:** Internally, the consumption rates are in l/second, but for convenience, the argument in command line is provided in l/min. The conversion is done automatically

---

**-decoconsrate**=<VALUE>
gas consumption rate (float) during deco (in l/minute).

Default: 12 l/min

Example:

Plan a dive with 20 l/min deco consumption rate

```
dipplanner --decoconsrate=20
```

---

**Note:** Internally, the consumption rates are in l/second, but for convenience, the argument in command line is provided in l/min. The conversion is done automatically

---

**-descentrate**=<VALUE>
descent rate (float) (in m/minute).

Default: 20 m/min

Example:

Plan a dive with 17 m/min descent rate

```
dipplanner --descentrate=17
```

---

**Note:** Internally, the ascent and descent rates are in m/second, but for convenience, the argument in command line is provided in m/min. The conversion is done automatically

---

**-ascentrate**=<VALUE>
ascent rate (float) (in m/minute).

Default: 10 m/min

Example:

Plan a dive with 9 m/min ascent rate

```
dipplanner --ascentrate=9
```

---

**Note:** Internally, the ascent and descent rates are in m/second, but for convenience, the argument in command line is provided in m/min. The conversion is done automatically

---

**-maxppo2**=<VALUE>
max allowed ppo2 (float) for this dive.

Default: 1.6

Example:

Set the max allowed ppo2 at 1.4

```
dipplanner --maxppo2=1.4
```

**-minppo2**=<VALUE>
minimum allowed ppo2 for this dive.

Default: 0.21

Example:

Set the min allowed ppo2 at 0.19

```
dipplanner --minppo2=0.19
```

**-maxend**=<VALUE>
max END (Equivalent narcosis Depth) allowed for this dive, in meter

Default: 30 m

Example:

Set the max END at 35m

```
dipplanner --maxend=35
```

---

**Note:** end calculation is based on narcotic index for all gases.

By default, dipplanner considers that oxygen is narcotic (same narcotic index than nitrogen)

All narcotic indexes can by changed in the config file, in the [advanced] section

---

**-samegasfordeco**
if set, do not use deco tanks (or bailout) for decompressions

Default: <not set>

By default, dipplanner will automatically switch to best mix for deco and if CCR, it will switch to deco bailout if it's best for decompression.

If you set this option, dipplanner will keep the last bottom gas used in OC or will still use CCR setpoint of last segment for deco

Example:

force the use of same gas for deco

```
dipplanner --samegasfordeco
```

**-forcesegmenttime**
if set, each input segment will be dove at the full time of the segment.

By default the segment time is shortened by descent or ascent time

Example:

```
dipplanner --forcesegmenttime
```

## 2.5 Advanced Parameters

**-depthcalcmethod**=<simple|complex>
method used for pressure from depth calculation.

- simple method uses only +10m = +1bar
- complex methods uses real water density calculation

Default: complex

Example:

switch depth calc method to simple

```
dipplanner --depthcalcmethod=simple
```

**-travelswitch**=<late|early>
Travel switch method (late or early).

- if late, it will keep the travel as long as possible (until either MOD or max END)
- if early, it will switch to bottom tank as soon as is it breathable

Default: late

Example:

switch travel switch to early

```
dipplanner --travelswitch=early
```

**-surfacetemp**=<VALUE>
Temperature at surface (float) in celcius

Default: 20 °C

Example:

change surface temperature to 30 °C

```
dipplanner --surfacetemp=30
```

**-ambiantpressureatsea**=<VALUE>
Change ambiant pressure at sea level (float) (in bar)

Default: 1.01325 b

Example:

change ambiant pressure at sea level to 1 bar

```
dipplanner --ambiantpressureatsea=1.0
```

## 2.6 Output Parameters

**-template**=<TEMPLATE>
Name of the template to be used The template file should be present in templates directory

see *Templates* for more infos on templates

Default: default-color.tpl

Example:

switch to html template and store the ouput in a html file

```
dipplanner --template=default.html > dive1.html
```

CHAPTER

THREE

# DIPPLANNER CONFIG FILE DOCUMENTATION

In config files, you can

- change all the dipplanner parameters (a little bit more than in command line)
- specify repetitive dives (in command-line, only one dive can be specified)

You may provide more than one config file in dipplanner command-line: it's up to you to organise the config like as you wish.

For example, you MAY create one config file for your dive parameters and another config file for your set of repetitive dives.

## 3.1 dive profiles

Dive profiles are specific sections in the config file, in the form:

```
[diveXXXX]
```

where XXXX represent a number. The dives whill be processed in crossant order

Inside a [diveXXXX] section you specify tanks, segments and surface_interval

### 3.1.1 tanks

Format:

```
tankXXX=tank_name;f_o2;f_he;Volume(l);Pressure(bar);Minimum gas rule
```

- tank_name: (str) (you choose the name) for the tank
- f_02: (float) fraction of oxygen in the tank. Between 0.0 and 1.0
- f_he: (float) fraction of helium in the tank. Between 0.0 and 1.0
- Volume: (float) Volume of the tank in bar
- Pressure: (float) Pressure of the tank in bar
- Minimum gas rule: (str) quantity of gas that should remain in the tank after the dive

  There two format for minimum gas rule:

- quantity of bar that should remain in the tank:

  format: "[0-9]+b"

  ex: "50b": it should remain 50 bar in the tank at the end of the dive

- "fraction rule" (like the rule of third in cave diving)

  format: "1/[0-9]"

  ex1: "1/3" : 1/3 of the tank to go in, 1/3 of the tank to go back and it should remain 1/3 of the tank at the end of the dive

  ex2: "1/6" : 1/6 of the tank to go in, 1/6 of the tank to go back and it should remain 2/3 of the tank at the end of the dive

---

**Note:** tank(s) list is only mandatory for the first dive.

On subsequent dive, if you choose not to specify tank(s), previous dive tanks will be used.

If 'automatic_tank_refill' is set to True, the tank will be full before the dive. If set to False, it'll use the remaining gas from last dive

> **Warning:** If, for a [dive] at least ONE tank is provided, ALL the Tank(s) MUST be specified (dipplanner will not add the new tank(s) to the previous one: dipplanner will reset the tank list with the new one.)

---

Example:

dive num 1 with two tanks:

12l tank filled with 200b or air. It should remain 50b at the end of the dive.

and

12l tank filled with Nitrox80. It should remain 30b at the end of the dive.

```
[dive1]

tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b
```

---

**Note:** this example is incomplete: it misses segments: see below

---

### 3.1.2 segments

Format:

```
segmentXXX="depth;duration;tank;setpoint"
```

- depth: (float) in meter
- duration: (float) in seconds (operators are allowed like: '30 * 60')
- tank: name of the tank (the 'tank_name' specified in -t option)
- setpoint: (float) 0.0 if OC, setpoint if CCR

**Note:** If you specify a setpoint > 0.0, the dive will automatically switch in CCR mode.

Example:

20 min at 30 meter using tank: airtank in OC mode

```
[dive1]

tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b

segment1=30;20*60;airtank;0.0
```

20 min at 30 meter using tank: airtank in OC mode

and then

25 min at 20 meter using tank: airtank in OC mode

```
[dive1]

tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b

segment1=30;20*60;airtank;0.0
segment2=20;25*60;airtank;0.0
```

### 3.1.3 surface_interval

surface interval (in seconds)

for repetitive dives, you can specify the surface time between the previous dive and this dive

### 3.1.4 Examples

Full example with two subsequent dives, with a surface interval of 1h30 between the two, using the same tanks for the two dives

```
[dive1]

tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b

segment1=30;20*60;airtank;0.0
segment2=20;25*60;airtank;0.0

[dive2]

surface_interval = 90*60

segment1=22;40*60;airtank;0.0
```

## 3.2 Controling the output

It's done via the section:

```
[output]
```

**template = <TEMPLATE>**
Name of the template to be used The template file should be present in templates directory

see *Templates* for more infos on templates

Default: default-color.tpl

Example:

switch to html template

```
[output]

template = default.html
```

## 3.3 general dive parameters

general dive parameters are in the section:

```
[general]
```

**deco_model = <ZHL16b|ZHL16c>**
Set the decompression model used for the calculations: either buhlmann ZHL16b or buhlmann ZHL16c

Default: ZHL16c

Example:

switch to ZHL16b deco model

```
[general]

deco_model = ZHL16b
```

**max_ppo2 = <VALUE>**
max allowed ppo2 (float) for this dive.

Default: 1.6

Example:

Set the max allowed ppo2 at 1.4

```
[general]

max_ppo2 = 1.4
```

**min_ppo2 = <VALUE>**
minimum allowed ppo2 for this dive.

Default: 0.21

Example:

Set the min allowed ppo2 at 0.19

```
[general]

max_ppo2 = 1.4
```

**max_end = <VALUE>**
max END (Equivalent narcosis Depth) allowed for this dive, in meter

Default: 30 m

Example:

Set the max END at 35m

```
[general]

max_end = 35
```

---

**Note:** end calculation is based on narcotic index for all gases.

By default, dipplanner considers that oxygen is narcotic (same narcotic index than nitrogen)

All narcotic indexes can by changed in the config file, in the [advanced] section

---

**descent_rate = <VALUE>**
descent rate (float) (in m/minute).

Default: 20 m/min

Example:

Plan a dive with 17 m/min descent rate

```
[general]

descent_rate = 17
```

---

**Note:** Internally, the ascent and descent rates are in m/second, but for convenience, the argument in command line is provided in m/min. The conversion is done automatically

---

**ascent_rate = <VALUE>**
ascent rate (float) (in m/minute).

Default: 10 m/min

Example:

Plan a dive with 9 m/min ascent rate

```
[general]

ascent_rate = 9
```

---

**Note:** Internally, the ascent and descent rates are in m/second, but for convenience, the argument in command line is provided in m/min. The conversion is done automatically

---

**gf_low = <VALUE>**
GF low: (int) in %, between 0 and 100

Default: 30%

Example:

GF low of 25%

```
[general]

gf_low = 25
```

---

**Note:** Internally, GFlow is a float number between 0.0 and 1.0, but for convenience, the argument in command line is provided in % value, between 0 and 100. The conversion is done automatically.

---

**gf_high = <VALUE>**

GF high: (int) in %, between 0 and 100

Default: 80%

Example:

GF high of 85%

```
[general]

gf_low = 85
```

---

**Note:** Internally, GFhigh is a float number between 0.0 and 1.0, but for convenience, the argument in command line is provided in % value, between 0 and 100. The conversion is done automatically.

---

**water = <sea|fresh>**

specify in which type of water you will do the dive: sea or fresh

Default: sea

Example:

Do a dive in a lake

```
[general]

water = fresh
```

**altitude = <VALUE>**

altitude (int) of the dive in meter.

> **Warning:** It's very important to specify this parameter if you do a dive in altitude

Default: 0m (sea level)

Example:

Dive at 1400m

```
[general]

altitude = 1400
```

**dive_consumption_rate = <VALUE>**

gas consumption rate (float) during dive (in l/minute).

Is it used for tank monitoring and associated with tank size, pressure and tank rules, it will warn you if your plannified dive ends without enough gas.

Default: 17 l/min

Example:

Plan a dive with 25 l/min dive consumption rate

```
[general]

dive_consumption_rate = 25
```

---

**Note:** Internally, the consumption rates are in l/second, but for convenience, the argument in command line is provided in l/min. The conversion is done automatically

---

**deco_consumption_rate = <VALUE>**
gas consumption rate (float) during deco (in l/minute).

Default: 12 l/min

Example:

Plan a dive with 20 l/min deco consumption rate

```
[general]

dive_consumption_rate = 20
```

---

**Note:** Internally, the consumption rates are in l/second, but for convenience, the argument in command line is provided in l/min. The conversion is done automatically

---

**run_time = <true|false>**
if true: segments represents runtime,

if false, segments represents segtime (in this case, the full time of the segment will be done and the descent and/or ascent time will be in addition.

Default: true

Example:

force segment time

```
[general]

run_time = false
```

**use_oc_deco = <true|false>**
if false, do not use deco tanks (or bailout) for decompressions

Default: true

By default, dipplanner will automatically switch to best mix for deco and if CCR, it will switch to deco bailout if it's best for decompression.

If you set this option, dipplanner will keep the last bottom gas used in OC or will still use CCR setpoint of last segment for deco

Example:

force the use of same gas for deco

```
[general]

use_oc_deco = false
```

**multilevel_mode = <true|false>**

---

**Todo**

check the usage of the multilevel_mode option in config-files

---

> **Warning:** do not use this option for the moment

Default: false

Example:

set multilevel mode to true

```
[general]

multilevel_mode = true
```

**automatic_tank_refill = <true|false>**

If 'automatic_tank_refill' is set to True, the tank will be full before the dive. If set to False, it'll use the remaining gas from last dive

Default: true

Example:

do not refill tank between dives

```
[general]

automatic_tank_refill = false
```

## 3.4 advanced dive parameters

avanced dive parameters are in the section:

```
[advanced]
```

> **Warning:** unless knowing what you're doing, this prefs should not be changed by the user

**fresh_water_density = <VALUE>**

Water density for fresh water (float)

Default: 1.0

**sea_water_density = <VALUE>**

Water density for sea water (float)

Default: 1.03

**absolute_max_ppo2 = <VALUE>**
In addition to max_ppo2, dipplanner uses and 'absolute_max_ppo2' which should never

Default: 2.0

**absolute_min_ppo2 = <VALUE>**
In addition to max_ppo2, dipplanner uses and 'absolute_max_ppo2' which should never

Default: 0.16

**absolute_max_tank_pressure = <VALUE>**
maximum pressure for a tank. (float) – in bar It's impossible to create a tank with higher pressure than this value

Default: 300b

**absolute_max_tank_size = <VALUE>**
maximum size for a tank (float) – in liter ($dm^3$) It's impossible to create a tank larger than this value

Default: 40l

---

**Note:** to handle double (connected) tanks, dipplanner considers one big tank, that's why 40l is the limit : 2x20l

---

**surface_temp = <VALUE>**
Temperature at surface (float) in celcius

Default: 20 °C

Example:

change surface temperature to 30 °C

```
[advanced]

surface_temp = 30
```

**he_narcotic_value = <VALUE>**
narcotic value for helium (float)

Default: 0.23

**n2_narcotic_value = <VALUE>**
narcotic value for nitrogen (float)

Default: 1.0

**o2_narcotic_value = <VALUE>**
narcotic value for oxygen (float)

Default: 1.0

**ar_narcotic_value = <VALUE>**
narcotic value for argon (float)

Default: 2.33

**stop_depth_increment = <VALUE>**
increment for each depth stop (int) in meter

When in ascent phase, do the deco stop every 'stop_depth_increment'.

By default, dipplanner do the deco stop every 3m

Default: 3m

**last_stop_depth = <VALUE>**
in meter : last stop before surfacing

Default: 3m

**stop_time_increment = <VALUE>**
Set the time increment used for the calculations of the dive model.

Default: 1s

Dipplanner use by default a time increment of 1s, which is more accurate than other dive plannification tools (which usually take 1 min).

But it has a serious performance impact. If you encounter some performance problem with dipplanner and do not want so much precision, you can raise this value

**force_all_stops = <true|false>**
one deco stop begun, force to stop to each deco depth stop

Default: true

**ambiant_pressure_sea_level = <VALUE>**
Change ambiant pressure at sea level (float) (in bar)

Default: 1.01325 b

Example:

change ambiant pressure at sea level to 1 bar

```
[advanced]

ambiant_pressure_sea_level = 1.0
```

**method_for_depth_calculation = <simple|complex>**
method used for pressure from depth calculation.

- simple method uses only +10m = +1bar
- complex methods uses real water density calculation

Default: complex

Example:

switch depth calc method to simple

```
[advanced]

method_for_depth_calculation = simple
```

**travel_switch = <late|early>**
Travel switch method (late or early).

- if late, it will keep the travel as long as possible (until either MOD or max END)
- if early, it will switch to bottom tank as soon as is it breathable

Default: late

Example:

switch travel switch to early

```
[advanced]

travel_switch = early
```

**flight_altitude = <VALUE>**
this parameter used in no flight time calculation : it's the parameter needed to calculate decompression until the altitude of the flight

Default: 2450

---

**Note:** the default value represents the maximum 'altitude equivalent' tolerated in flight by international regulation (8000 feet = 2 438.4 meters rounded to 2450m)

---

## 3.5 Examples

### 3.5.1 Small Example: only set dives

```
# dipplanner config file
# this file is used by the command line tool and
# override the defaults parameters or input some dive profiles

# ============================== dive profiles ================================

[dive1]

tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b

segment1=30;20*60;airtank;0.0
segment2=20;25*60;airtank;0.0

[dive2]

surface_interval = 90*60

segment1=22;40*60;airtank;0.0
```

### 3.5.2 Full Example

Config file with all the settings set below.

---

**Note:** the default_config.cfg in ./configs directory set all the parameters to their default values (wich is not the case in the following example)

---

```
# dipplanner config file
# this file is used by the command line tool and
# override the defaults parameters or input some dive profiles

# ============================== dive profiles ================================

[dive1]
```

```
tank1=airtank;0.21;0.0;12;200,50b
tank2=nitrox;0.80;0.0;12;200;30b

segment1=30;20*60;airtank;0.0
segment2=20;25*60;airtank;0.0

[dive2]

surface_interval = 90*60

segment1=22;40*60;airtank;0.0

# ============================== Other parameters ==============================
[output]
# template used for output result
# templating uses jinja2, see documentation for more infos
template = default.html

[general]
# deco model
# choose between buhlmann ZHL16b or ZHL16c
# ZHL16c is the default
deco_model = ZHL16b

# ppo2
# defines the max and min_ppo2
max_ppo2 = 1.4
min_ppo2 = 0.19

# max end
# defines the max END for the dives
max_end = 35

# decent and ascent rate, in m/minute
descent_rate = 17
ascent_rate = 9

# Gradient factors in %
gf_low = 35
gf_high = 85

# type of water
# possible values :
# sea -- sea water
# fresh -- fresh water
water = fresh

# dive altitude
# in meter
altitude = 1400

# consumption rates
# in liter / minute (the program does the conversion internally)
dive_consumption_rate = 25
deco_consumption_rate = 20

# run_time flag
# if true: segments represents runtime,
```

```
# if false, segments represents segtime
run_time = false

# Use Open Circuit deco flag
# if True, use enabled gases of decomp in oc or bailout
use_oc_deco = false

# multilevel_mode
multilevel_mode = false

# automatic_tank_refill
# if 'automatic_tank_refill' is set to True, the tank will be full before the
# dive. If set to False, it'll use the remaining gas from last dive
automatic_tank_refill = false

# ========================== Advanced  parameters ==============================
# ========================= "Internal" Settings ================================
# !!!   unless knowing what you're doing, this prefs should not be changed   !!!
# !!!                                by the user                              !!!
# ============================================================================
[advanced]
# water density kg/l
fresh_water_density = 1.0
sea_water_density = 1.03

absolute_max_ppo2 = 2.0
absolute_min_ppo2 = 0.16

# in bar
absolute_max_tank_pressure = 300

# in liter
absolute_max_tank_size = 40

# temperature at surface
surface_temp = 30

he_narcotic_value = 0.23
n2_narcotic_value = 1.0
o2_narcotic_value = 1.0
ar_narcotic_value = 2.33

# in meter
stop_depth_increment = 3

# in meter : last stop before surfacing
last_stop_depth = 3

# in second
stop_time_increment = 1

# one deco stop begun, force to stop to each deco depth
# stop
force_all_stops = true

# surface pressure at sea level (in bar)
ambiant_pressure_sea_level = 1.0
```

```
# either simple (/10) or complex
method_for_depth_calculation = simple

# travel switch method
# if 'late', dipplanner will try to keep the travel as long as possible
#   until either MOD or max END
# if 'early', dipplanner will switch to bottom tank as soon as is it breathable
travel_switch = early

# flight altitude
# parameter used in no flight time calculation
flight_altitude = 2450
```

### 3.5.3 Config file with default values

```
# dipplanner config file
# this file is used by the command line tool and
# override the defaults parameters or input some dive profiles

# This file represent default configuration, without any dive profile nor tank.

# ============================== dive profiles ================================
# repetitive dives are given using [diveXXX] section, where XXX represent a
# number.
# the dives will be done in croissant order.
#[dive1]

# Tank list for this dive:
# Format: tankXXX=tank_name;fO2;fHe;Volume(l);Pressure(bar)
#tank1 = airtank;0.21;0.0;15;230;50b

# segment list for this dive. At least ONE segment is mandatory
# Format: segmentXXX=depth(m);duration(s);tank_name;set_point(for ccr)
#segment1 = 30;20*60;airtank;0.0

#[dive2]
# surface_interval (in seconds)
# for repetitive dives, you can specify the surface time between the previous
# dive and this dive
#surface_interval = 60*60

# Tanks
# see dive 1 for more explanation
# tank list is not mandatory for repetitive dives : if not given
# last dive tanks will be used.
# if 'automatic_tank_refill' is set to True, the tank will be full before the
# dive. If set to False, it'll use the remaining gas from last dive
# If at least ONE tank is provided for a repetitive dive, ALL the Tank MUST
# be specified
# newtank = txtank;0.21;0.30;15;230;50b
# tank1 = airtank;0.21;0.0;15;230;50b

# segment list for this dive. At least ONE segment is mandatory
#segment1 = 20;30*60;airtank;0.0

#[dive3]...
```

```
# ============================== Other parameters ==============================
[output]
# template used for output result
# templating uses jinja2, see documentation for more infos
template = default-color.tpl

[general]
# deco model
# choose between buhlmann ZHL16b or ZHL16c
# ZHL16c is the default
deco_model = ZHL16c

# ppo2
# defines the max and min_ppo2
# default values :
#   max_ppo2 : 1.6
#   min_ppo2 : 0.21
max_ppo2 = 1.6
min_ppo2 = 0.21

# max end
# defines the max END for the dives
# default value (in meter):
# max_end : 30
max_end = 30

# decent and ascent rate, in m/minute
descent_rate = 20
ascent_rate = 10

# Gradient factors in %
gf_low = 30
gf_high = 80

# type of water
# possible values :
# sea -- sea water
# fresh -- fresh water
water = sea

# dive altitude
# in meter
altitude = 0

# consumption rates
# in liter / minute (the program does the conversion internally)
dive_consumption_rate = 17
deco_consumption_rate = 12

# run_time flag
# if true: segments represents runtime,
# if false, segments represents segtime
run_time = true

# Use Open Circuit deco flag
# if True, use enabled gases of decomp in oc or bailout
use_oc_deco = true
```

```
# multilevel_mode
multilevel_mode = false

# automatic_tank_refill
# if 'automatic_tank_refill' is set to True, the tank will be full before the
# dive. If set to False, it'll use the remaining gas from last dive
automatic_tank_refill = true

# ========================== Advanced  parameters ==============================
# ========================= "Internal" Settings ================================
# !!!   unless knowing what you're doing, this prefs should not be changed   !!!
# !!!                                  by the user                            !!!
# ==============================================================================
[advanced]
# water density kg/l
fresh_water_density = 1.0
sea_water_density = 1.03
absolute_max_ppo2 = 2.0
absolute_min_ppo2 = 0.16

# in bar
absolute_max_tank_pressure = 300

# in liter
absolute_max_tank_size = 40

# temperature at surface
surface_temp = 20

he_narcotic_value = 0.23
n2_narcotic_value = 1.0
o2_narcotic_value = 1.0
ar_narcotic_value = 2.33

# in meter
stop_depth_increment = 3

# in meter : last stop before surfacing
last_stop_depth = 3

# in second
stop_time_increment = 1

# one deco stop begun, force to stop to each deco depth
# stop
force_all_stops = true

# surface pressure at sea level (in bar)
ambiant_pressure_sea_level = 1.01325

# either simple (/10) or complex
method_for_depth_calculation = complex

# travel switch method
# if 'late', dipplanner will try to keep the travel as long as possible
#   until either MOD or max END
# if 'early', dipplanner will switch to bottom tank as soon as is it breathable
travel_switch = late
```

```
# flight altitude
# parameter used in no flight time calculation : it's the parameter needed
# to calculate decompression until the altitude of the flight
# the default value represents the maximum 'altitude equivalent' tolerated
# in flight by international regulation
# (8000 feet = 2 438.4 meters rounded to 2450m)
flight_altitude = 2450
```

CHAPTER

FOUR

# TEMPLATES

dipplanner uses Jinja2 template engine.

For all documentation about jinja2, please see their own documentation

This document will focus on dipplanner objects sent to jinja2 templating system and how to use them.

dipplanner sends two objects to the template engine:

- settings: contains all the parameters used for the dives
- [Dive, ...]: a list of Dive objects

It only one dive is calculated, dipplanner still send a list of dive, with one element in the list.

Settings will be usefull to display some dive parameters, like configured GF for example:

```
Configuration : GF:{{ settings.GF_LOW*100 }}-{{ settings.GF_HIGH*100 }}
```

## 4.1 dives

Because dipplanner sends a list of Dives, the template MUST iterate this list, event for one element :

```
{% for dive in dives %}

...

{% endfor %}
```

Dive attributes are described here: *dive*

## 4.2 settings

settings attributes are described here: *settings*

CHAPTER

FIVE

# SOURCE CODE DOCUMENTATION

## 5.1 dive

dive class module

Each Dive represent one dive (and only one) For successive dives, it is possible to provide the parameters of the previous dive in order to calculate the next one.

**exception** dipplanner.dive.**NothingToProcess**(*description=''*)
raised when the is no input segments to process

**__init__**(*description=''*)
constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()
String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**exception** dipplanner.dive.**InstanciationError**(*description=''*)
raised when the Dive constructor encounters a problem. In this case, it can not continue

**__init__**(*description=''*)
constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()
String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**exception** dipplanner.dive.**ProcessingError**(*description=''*)
raised when the is no input segments to process

**__init__**(*description=''*)
constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()
String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**exception** dipplanner.dive.**InfiniteDeco**(*description=''*)
raised when the deco time becomes enourmous (like infinite)

**__init__**(*description=''*)
constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()
String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**class** dipplanner.dive.**Dive**(*known_segments*, *known_tanks*, *previous_profile=None*)
Conducts dive based on inputSegments, knownGases, and an existing model. Iterates through dive segments updating the Model. When all dive segments are processed then calls ascend(0.0) to return to the surface.

The previous_profile (Model) can be either null in which case a new model is created, or can be an existing model with tissue loadings.

Gas switching is done on the final ascent if OC deco or bailout is specified.

Outputs profile to a List of dive segments

Attributes:

- input_segments – (list) Stores enabled input dive segment objects
- output_segments – (list) Stores output segments produced by this class
- tanks – (list) Stores enabled dive tank objects
- current_tank – current tank object
- current_depth – current dive depth
- ambiant_pressure – (current) ambiant pressure
- current_f_he – current gas fraction of He
- current_f_n2 – current gas fraction of N2
- current_f_o2 – current gas fraction of O2
- model – model used for this dive
- run_time – runTime
- pp_o2 – CCR ppO2, if OC : 0.0
- is_closed_circuit – Flag to store CC or OC
- in_final_ascent – flag for final ascent
- is_repetative_dive – Flag for repetative dives
- surface_interval – for surf. int. in seconds
- no_flight_time_value – calculated no flight time
- metadata – description for the dive

**__init__**(*known_segments*, *known_tanks*, *previous_profile=None*)
Constructor for Profile class

For fist dive, instanciate the profile class with no model (profile will create one for you) For repetative dives, instanciate profile class with the previous model

***Keyword Arguments:***

**known_segments** – list of input segments

**known_tanks** – list of tanks for this dive

**previous_profile** (Model) – model object of the precedent dive

**Return:** <nothing>

---

**Note:** the constructor should not fail. If something if wrong, it MUST still instantiate itself, with errors in his own object

---

**__str__**()
Return a human readable name of the segment

***Keyword Arguments:*** <none>

***Return:***

**str – a string with the result of the calculation of the dives** using the default template

***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode

***Keyword Arguments:*** <none>

***Return:***

**ustr – an unicode string with the result of the calculation of** the dives using the default template

***Raise:*** <nothing>

**__cmp__**(*otherdive*)
Compare a dive to another dive, based on run_time

***Keyword arguments:*** otherdive (Dive) – another dive object

***Returns:*** Integer – result of cmp()

***Raise:*** <nothing>

**output**(*template=None*)
Returns the dive profile calculated, using the template given in settings or command lines. (and not only the default template)

***Keyword Arguments:*** <none>

***Return:***

**str – a string with the result of the calculation of the dives** using the choosen template

***Raise:*** <nothing>

**do_surface_interval**(*time*)
Conducts a surface interval by performing a constant depth calculation on air at zero meters

***Keyword Arguments:***

**time** (int) – duration of the interval, in seconds

***Returns:*** <nothing>

***Raise:*** <Exceptions from model>

**get_surface_interval**()
Returns surface interval in mm:ss format

***Keyword Arguments:*** <nothing>

***Returns:*** str – surface interval time in mmm:ss format

***Raise:*** <nothing>

**refill_tanks**()
refile all tanks defined in this dive it is used for repetitive dives

***Keyword Arguments:*** <none>

***Returns:*** <nothing>

***Raise:*** <nothing>

**is_dive_segments**()
Returns true if there are loaded dive segments else false means there is nothing to process

***Keyword Arguments:*** <none>

***Returns:*** True (bool) – if there is at least one input dive segment to process False (bool) – if there is no dive segment to process

***Raise:*** <nothing>

**do_dive_without_exceptions**()
Call do_dive, and handle exceptions internally : do not raise any "dive related" exception : add the exception inside self.dive_exceptions instead.

***Keyword Arguments:*** <none>

***Return:*** <nothing>

***Raise:*** <nothing>

**do_dive**()
Process the dive

***Keyword Arguments:*** <none>

***Return:*** <nothing>

***Raise:*** NothingToProcess – if there is no input segment to process or <Exceptions from model>

**get_no_flight_hhmmss**()
Returns no flight time (if calculated) in hhmmss format instead of an int in seconds

---

**Note:** This method does not calculate no_flight_time you need to call no_flight_time() or no_flight_time_wo_exception() before.

---

***Keyword Arguments:*** <none>

**Returns:** str – "hh:mm:ss" no flight time str – "" if no flight time is not calculated

**no_flight_time_wo_exception**(*altitude=2450*, *tank=None*)
Call no_flight_time, and handle exceptions internally: do not raise any "dive related" exception: add the exception inside self.dive_exceptions instead.

***Keyword Arguments:***

**altitude** (int) – in meter : altitude used for the calculation

**flight_ascent_rate** (float) – in m/s

**tank** (Tank) – optionnal: it is possible to provide a tank while calling no_flight_time to force "no flight deco" with another mix than air. In this case, we will 'consume' the tank When the tank is empty, it automatically switch to air

***Returns:*** int – no fight time in seconds

***Raise:*** <nothing>

**no_flight_time**(*altitude=2450*, *tank=None*)
Evaluate the no flight time by 'ascending' to the choosen flight altitude. Ascending will generate the necessary 'stop' at the current depth (which is 0m). The stop time represents the no flight time

***Keyword Arguments:***

**altitude** (int) – in meter : altitude used for the calculation

**flight_ascent_rate** (float) – in m/s

**tank** (Tank) – optionnal: it is possible to provide a tank while calling no_flight_time to force "no flight deco" with another mix than air. In this case, we will 'consume' the tank When the tank is empty, it automatically switch to air

***Returns:*** int – no fight time in seconds

***Raise:***

**InfiniteDeco - if the no flight time can not achieve enough** decompression to be able to go to give altitude

**ascend**(*target_depth*)
Ascend to target depth, decompressing if necessary. If inFinalAscent then gradient factors start changing, and automatic gas selection is made.

This method is called by do_dive()

***Keyword Arguments:***

**target_depth** (float) – in meter, target depth for the ascend

Returns: <nothing>

Raise: <Exceptions from model>

**do_gas_calcs**()
Estimate gas consumption for all output segments and set this into the respective gas objects

***Keyword Arguments:*** <none>

***Returns:*** <nothing>

***Raise:*** <Exceptions from tank>

**set_deco_gas**(*depth*)
Select appropriate deco gas for the depth specified Returns true if a gas switch occured

***Keyword Arguments:***

**depth** (float) – target depth to make the choice

***Returns:*** True – if gas swich occured False – if no gas switch occured

***Raise:*** <Exceptions from tank>

## 5.2 segment

Segment classes A segment is a portion of a dive in the same depth (depth + duration)

**exception** dipplanner.segment.**UnauthorizedMod**(*description*)
raised when the MOD is not possible according to the depth(s) of the segment

**__init__**(*description*)
constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()
String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**class** dipplanner.segment.**Segment**
Base class for all types of segments

*Attributes:*

•type (str) – type of segment

types of segments can be :

–const = “Constant Depth”

–ascent = “Ascent”

–descent = “Descent”

–deco = “Decompression”

–waypoint = “Waypoint”

–surf = “Surface”

•in_use (boolean) – True if segment is used

•depth (float) – depth of this segment, in meter

- time (float) – duration of this segment, in seconds
- run_time (float) – runtime in profile
- setpoint (float) – setpoint for CCR
- tank (Tank) – refer to tank object used in this segment

**__init__**()
constructor for Segment base class, just defines all the parameters

***Keyword arguments:*** <none>

***Returns:*** <nothing>

***Raise:*** <nothing>

**__str__**()
Return a human readable name of the segment

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**check**()
check if it's a valid segment Should be executed before calculating dives

the check does not return anything if nok, but raises Exceptions

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** <DiveExceptions>

**check_mod**(*max_ppo2=None*)
checks the mod for this segment according to the used tank.

***Keyword arguments:***

**max_ppo2** (float) – max tolerated ppo2

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**check_min_od**()
checks the minimum od for this segment according to the used tank. (hypoxic cases)

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**get_time_str**()
returns segment time in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**get_run_time_str**()
returns runtime in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**get_p_absolute**(*method='complex'*)
returns the absolute pression in bar (1atm = 1ATA = 1.01325 bar = 14.70psi)

Simple method : 10m = +1 bar Complex method : use real density of water, T°, etc...

***Keyword arguments:*** method – 'simple' or 'complex'

***Returns:*** float – indicating the absolute pressure in bar

***Raise:*** ValueError – when providing a bad method

**get_end**()
Calculates and returns E.N.D : Equivalent Narcosis Depth

Instead of Mvplan, it uses a 'calculation method' based on narcosis effet of all gas used, assuming there is no trace of other gases (like argon) in the breathing gas, but compare the narcotic effect with surface gas, wich is 'air' and contains a small amount of argon

***Keyword arguments:*** <none>

***Returns:*** integer – Equivalent Narcosis Depth in meter

***Raise:*** <nothing>

**gas_used**()
returns the quantity (in liter) of gas used for this segment

(this method is empty)

**class** dipplanner.segment.**SegmentDive**(*depth*, *time*, *tank*, *setpoint=0*)
Specialisation of segment class for dive segments

**__init__**(*depth*, *time*, *tank*, *setpoint=0*)
Constructor for SegmentDive class. Look at base class for more explanations

***Keyword arguments:***

**depth** (float) – in meter, the (constant) depth for this segment

**time** (float) – in second, duration of this segment

**tank** (Tank) – object instance of Tank class : describe the tank used in this segment

**setpoint** (float) – for CCR, setpoint used for this segment for OC : setpoint should be zero

***Returns:*** <nothing>

***Raise:***

**UnauthorizedMod – if depth is incompatible** with either min or max mod

**gas_used**()
calculates returns the quantity (in liter) of gas used for this segment

***Keyword arguments:*** <none>

***Returns:*** float – in liter, quantity of gas used

**__str__**()
Return a human readable name of the segment

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**check**()
check if it's a valid segment Should be executed before calculating dives

the check does not return anything if nok, but raises Exceptions

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** <DiveExceptions>

**check_min_od**()
checks the minimum od for this segment according to the used tank. (hypoxic cases)

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**check_mod**(*max_ppo2=None*)
checks the mod for this segment according to the used tank.

***Keyword arguments:***

**max_ppo2** (float) – max tolerated ppo2

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**get_end**()
Calculates and returns E.N.D : Equivalent Narcosis Depth

Instead of Mvplan, it uses a 'calculation method' based on narcosis effet of all gas used, assuming there is no trace of other gases (like argon) in the breathing gas, but compare the narcotic effect with surface gas, wich is 'air' and contains a small amount of argon

***Keyword arguments:*** <none>

***Returns:*** integer – Equivalent Narcosis Depth in meter

***Raise:*** <nothing>

**get_p_absolute**(*method='complex'*)
returns the absolute pression in bar (1atm = 1ATA = 1.01325 bar = 14.70psi)

Simple method : 10m = +1 bar Complex method : use real density of water, T°, etc...

***Keyword arguments:*** method – 'simple' or 'complex'

***Returns:*** float – indicating the absolute pressure in bar

***Raise:*** ValueError – when providing a bad method

**get_run_time_str**()
returns runtime in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**get_time_str**()
returns segment time in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**class** dipplanner.segment.**SegmentDeco**(*depth*, *time*, *tank*, *setpoint=0*)
Specialisation of segment class for deco segments

**__init__**(*depth*, *time*, *tank*, *setpoint=0*)
Constructor for SegmentDeco class. Look at base class for more explanations

In deco segment, we also have to manage some new parameters :

- gf_used : which gradient factor is used
- control_compartment : who is the control compartement
- mv_max : max M-value for the compartment

***Keyword arguments:***

**depth** (float) – in meter, the (constant) depth for this segment

**time** (float) – in second, duration of this segment

**tank** (Tank) – object instance of Tank class : describe the tank used in this segment

**setpoint** (float) – for CCR, setpoint used for this segment for OC : setpoint should be zero

***Returns:*** <nothing>

***Raise:***

**UnauthorizedMod – if depth is incompatible** with either min or max mod

**gas_used**()
calculates returns the quantity (in liter) of gas used for this segment

***Keyword arguments:*** <none>

***Returns:*** float – in liter, quantity of gas used

**__str__**()
Return a human readable name of the segment

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**check**()
check if it's a valid segment Should be executed before calculating dives

the check does not return anything if nok, but raises Exceptions

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** <DiveExceptions>

**check_min_od**()
checks the minimum od for this segment according to the used tank. (hypoxic cases)

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**check_mod**(*max_ppo2=None*)
checks the mod for this segment according to the used tank.

***Keyword arguments:***

**max_ppo2** (float) – max tolerated ppo2

***Returns:*** True (bool) – if check is ok

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**get_end**()
Calculates and returns E.N.D : Equivalent Narcosis Depth

Instead of Mvplan, it uses a 'calculation method' based on narcosis effet of all gas used, assuming there is no trace of other gases (like argon) in the breathing gas, but compare the narcotic effect with surface gas, wich is 'air' and contains a small amount of argon

***Keyword arguments:*** <none>

***Returns:*** integer – Equivalent Narcosis Depth in meter

***Raise:*** <nothing>

**get_p_absolute**(*method='complex'*)
returns the absolute pression in bar (1atm = 1ATA = 1.01325 bar = 14.70psi)

Simple method : 10m = +1 bar Complex method : use real density of water, T°, etc...

***Keyword arguments:*** method – 'simple' or 'complex'

***Returns:*** float – indicating the absolute pressure in bar

***Raise:*** ValueError – when providing a bad method

**get_run_time_str**()
returns runtime in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**get_time_str**()
returns segment time in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**class** dipplanner.segment.**SegmentAscDesc**(*start_depth*, *end_depth*, *rate*, *tank*, *setpoint=0*)
Specialisation of segment class for Ascent or Descent segments

**__init__**(*start_depth*, *end_depth*, *rate*, *tank*, *setpoint=0*)
Constructor for SegmentAscDesc class. Look at base class for more explanations in this segment, we do not specify time, but rate (of ascending or descending) and start_depth and end_depth The comparaison between start and end depth determine if asc or desc rate is given in m/min

***Keyword arguments:***

**start_depth** (float) – in meter, the starting depth for this segment

**end_depth** (float) – in meter, the ending depth for this segment

**rate** (float) – in m/s, rate of ascending or descending

**tank** (Tank) – object instance of Tank class : describe the tank used in this segment

**setpoint** (float) – for CCR, setpoint used for this segment for OC : setpoint should be zero

***Returns:*** <nothing>

***Raise:***

**UnauthorizedMod – if depth is incompatible** with either min or max mod

**check_mod**(*max_ppo2=None*)
checks the mod for this segment according to the used tank.

***Keyword arguments:***

**max_ppo2** (float) – max tolerated ppo2

***Returns:*** <nothing>

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**check_min_od**()
checks the minimum od for this segment according to the used tank. (hypoxic cases)

***Keyword arguments:*** <none>

***Returns:*** <nothing>

***Raise:*** UnauthorizedMod – if segments goes below max mod or upper min mod

**__str__**()
Return a human readable name of the segment

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode

***Keyword arguments:*** <none>

***Returns:*** str – representation of the segment in the form: (see __repr__)

***Raise:*** <nothing>

**check**()
check if it's a valid segment Should be executed before calculating dives

the check does not return anything if nok, but raises Exceptions

***Keyword arguments:*** <none>

***Returns:*** True (bool) – if check is ok

***Raise:*** <DiveExceptions>

**gas_used**()
calc. and returns the quantity (in liter) of gas used in this segment because it's ascend or descent, the gas is not used at the same rate during the segment Because the rate is the same during all the segment, we can use the consumption at the average depth of the segment

***Keyword arguments:*** <none>

***Returns:*** float – in liter, quantity of gas used

**get_end**()
Calculates and returns E.N.D : Equivalent Narcosis Depth

Instead of Mvplan, it uses a 'calculation method' based on narcosis effet of all gas used, assuming there is no trace of other gases (like argon) in the breathing gas, but compare the narcotic effect with surface gas, wich is 'air' and contains a small amount of argon

***Keyword arguments:*** <none>

***Returns:*** integer – Equivalent Narcosis Depth in meter

***Raise:*** <nothing>

**get_p_absolute**(*method='complex'*)
returns the absolute pression in bar (1atm = 1ATA = 1.01325 bar = 14.70psi)

Simple method : 10m = +1 bar Complex method : use real density of water, T°, etc...

***Keyword arguments:*** method – 'simple' or 'complex'

***Returns:*** float – indicating the absolute pressure in bar

***Raise:*** ValueError – when providing a bad method

**get_run_time_str**()
returns runtime in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

**get_time_str**()
returns segment time in the form MMM:SS

***Keyword Arguments:*** <none>

***Returns:*** string – segment time in the form MMM:SS

***Raise:*** <nothing>

## 5.3 tank

Contains a Tank Class

---

**Note:** in MVPlan, this class was the 'Gas' class

---

**exception** dipplanner.tank.**InvalidGas**(*description*)

Exception raised when the gas informations provided for the Tank are invalid

**__init__**(*description*)

constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()

String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

***Raise:*** <nothing>

**__unicode__**()

unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**exception** dipplanner.tank.**InvalidTank**(*description*)

Exception raised when the tank infos provided are invalid

**__init__**(*description*)

constructor : call the upper constructor and set the logger

***Keyword Arguments:***

**description** (str) – text describing the error

***Return:*** <nothing>

***Raise:*** <nothing>

**__str__**()

String representing the object

***Keyword Arguments:*** <none>

***Return:*** str – a string describing the Exception

*Raise:* <nothing>

**__unicode__**()
unicode string representing the object

*Keyword Arguments:* <none>

*Return:* ustr – a unicode string describing the Exception

*Raise:* <nothing>

**exception** dipplanner.tank.**InvalidMod**(*description*)
Exception raised when the given MOD is incompatible with the gas provided for the tank

**__init__**(*description*)
constructor : call the upper constructor and set the logger

*Keyword Arguments:*

**description** (str) – text describing the error

*Return:* <nothing>

*Raise:* <nothing>

**__str__**()
String representing the object

*Keyword Arguments:* <none>

*Return:* str – a string describing the Exception

*Raise:* <nothing>

**__unicode__**()
unicode string representing the object

*Keyword Arguments:* <none>

*Return:* ustr – a unicode string describing the Exception

*Raise:* <nothing>

**exception** dipplanner.tank.**EmptyTank**(*description*)
Exception raised when trying to consume more gas in tank than the remaining gas

**__init__**(*description*)
constructor : call the upper constructor and set the logger

*Keyword Arguments:*

**description** (str) – text describing the error

*Return:* <nothing>

*Raise:* <nothing>

**__str__**()
String representing the object

*Keyword Arguments:* <none>

*Return:* str – a string describing the Exception

*Raise:* <nothing>

**__unicode__**()
unicode string representing the object

***Keyword Arguments:*** <none>

***Return:*** ustr – a unicode string describing the Exception

***Raise:*** <nothing>

**class** dipplanner.tank.**Tank**(*f_o2=0.21*, *f_he=0.0*, *max_ppo2=1.6*, *mod=None*, *tank_vol=12.0*, *tank_pressure=200*, *tank_rule='30b'*)

This class implements a representation of dive tanks wich contains breathing Gas We provide proportion of N2, O2, He, calculates MOD and volumes during the dives We can also (optionally) provide the type of tanks : volume and pressure

*Attributes:*

- f_o2 (float) – fraction of oxygen in the gas in % (>= 0.0 & <= 1.0)
- f_he (float) – fraction of helium in the gas in % (>= 0.0 & <= 1.0)
- **f_n2 (float) – fraction of nitrogen in the gas in %** (>= 0.0 & <= 1.0)
- max_ppo2 (float) – maximum tolerated ppo2 for this tank
- tank_vol (float) – volume of tank in liter
- tank_pressure (float) – pressure of tank in bar
- mod (float) – maximum operating depth of the tank
- in_use (boolean) – is the tank used for the dive of not
- total_gas (float) – total gas volume of the tank in liter
- used_gas (float) – used gas in liter
- remaining_gas (float) – remaining gas in liter
- min_gas (float) – minimum remaining gas in liter

**__init__**(*f_o2=0.21*, *f_he=0.0*, *max_ppo2=1.6*, *mod=None*, *tank_vol=12.0*, *tank_pressure=200*, *tank_rule='30b'*)

Constructor for Tank class

If nothing is provided, create a default 'Air' with 12l/200b tank and max_ppo2 to 1.6 (used to calculate mod) if mod not provided, mod is calculed based on max tolerable ppo2

***Keyword arguments:***

**f_o2** (float) – Fraction of O2 in the gaz in % value between 0.0 and 1.0

**f_he** (float) – Fraction of He in the gaz in % value between 0.0 and 1.0

**max_ppo2** (float) – sets the maximum ppo2 you want for this tank (default: settings.DEFAULT_MAX_PPO2)

**mod** (float) – Specify the mod you want. * if not provided, calculates the mod based on max_ppo2

- if provided and not compatible with max_ppo2: raise InvalidMod

**tank_vol** (float) – Volume of the tank in liter

**tank_pressure** (float) – Pressure of the tank, in bar

**tank_rule** (float) – rule for warning in the tank consumption must be either :

- **xxxb (ex: 50b means 50 bar minimum at** the end of the dive)
- **1/x**

- **ex** [1/3 for rule of thirds:]
  - 1/3 for way in,
  - 1/3 for way out,
  - 1/3 remains at the end of the dive)
- **ex2: 1/6 rule:**
  - 1/6 way IN,
  - 1/6 wau OUT,
  - 2/3 remains

***Returns:*** <nothing>

*Raise:*
•InvalidGas – see validate()
•InvalidMod – if mod > max mod based on max_ppo2 and see validate()
•InvalidTank – see validate()

**calculate_real_volume**(*tank_vol=None*, *tank_pressure=None*, *f_o2=None*, *f_he=None*, *temp=15*)

Calculate the real gas volume of the tank (in liter) based on Van der waals equation: (P+n2.a/V2).(V-n.b)=n.R.T

***Keyword arguments:***

**tank_vol** (float) – Volume of the tank in liter optional : if not provided, use self.tank_vol

**tank_pressure** (float) – Pressure of the tank in bar optional : if not provided, use self.tank_pressure

**f_o2** (float) – fraction of O2 in the gas optional : if not provided, use self.f_o2

**f_he** (float) – fraction of He in the gas optional : if not provided, use self.f_he

***Returns:*** float – total gas volume of the tank in liter

***Raise:*** <nothing>

**__str__**()

Return a human readable name of the tank

***Keyword arguments:*** <none>

***Returns:***

**str – name of the tank in the form:** “Air” “Nitrox 80” ...

***Raise:*** <nothing>

**__unicode__**()

Return a human readable name of the tank in unicode

***Keyword arguments:*** <none>

***Returns:***

**str – name of the tank in the form:** “Air” “Nitrox 80” ...

***Raise:*** <nothing>

**__cmp__**(*othertank*)

Compare a tank to another tank, based on MOD

***Keyword arguments:*** othertank (Tank) – another tank object

***Returns:*** integer – result of cmp()

***Raise:*** <nothing>

**_calculate_mod**(*max_ppo2*)

calculate and returns mod for a given ppo2 based on this tank info result in meter

***Keyword arguments:***

**max_ppo2** – maximum ppo2 accepted (float). Any value accepted, but should be > 0.0

***Returns:*** integer – Maximum Operating Depth in meter

***Raise:*** <nothing>

**_validate**()

Test the validity of the tank informations inside this object if validity check fails raise an Exception 'InvalidTank'

***Keyword arguments:*** <nothing>

***Returns:*** <nothing>

***Raise:***

- **InvalidGas – When proportions of gas exceed** 100% for example (or negatives values)
- **InvalidMod – if mod > max mod based on max_ppo2 or ABSOLUTE_MAX_MOD** ABSOLUTE_MAX_MOD is a global settings which can not be exceeded.
- **InvalidTank – when pressure or tank size exceed maximum values or are** incorrect (like negatives) values

**name**()

returns a Human readable name for the gaz and tanks Differnt possibilities: Air, Nitrox, Oxygen, Trimix, Heliox

***Keyword arguments:*** <none>

***Returns:***

**str – name of the tank in the form:** "Air" "Nitrox" ...

***Raise:*** <nothing>

**get_tank_info**()

returns tank infos : size, remaining vol example of tank info: 15l-90% (2800/3000l)

***Keyword arguments:*** <none>

***Returns:***

**str – infos of the tank in the form:** "12.0l-100.0% (2423.10/2423.10l)" ...

***Raise:*** <nothing>

**get_mod**(*max_ppo2=None*)

return mod (maximum operating depth) in meter if no argument provided, return the mod based on the current tank (and configured max_ppo2) if max_ppo2 is provided, returns the (new) mod based on the given ppo2

***Keyword arguments:***

**max_ppo2** (float) – ppo2 for mod calculation

***Returns:*** float – mod in meter

***Raise:*** <nothing>

**get_min_od**(*min_ppo2=0.16*)

return in meter the minimum operating depth for the gas in the tank return 0 if diving from/to surface is ok with this gaz

***Keyword arguments:***

**min_ppo2** (float) – minimum tolerated ppo2

***Returns:*** float – minimum operating depthin meter

***Raise:*** <nothing>

**get_mod_for_given_end**(*end*)

calculate a mod based on given end and based on gaz inside the tank

---

**Note:** end calculation is based on narcotic index for all gases.

By default, dipplanner considers that oxygen is narcotic (same narcotic index than nitrogen)

All narcotic indexes can by changed in the config file, in the [advanced] section

---

***Keyword arguments:***
**end** (int) – equivalent narcotic depth in meter
***Returns:*** int – mod: depth in meter based on given end
***Raise:*** <nothing>

**get_end_for_given_depth**(*depth*)
calculate end (equivalent narcotic depth) based on given depth and based on gaz inside the tank

---

**Note:** end calculation is based on narcotic index for all gases.

By default, dipplanner considers that oxygen is narcotic (same narcotic index than nitrogen)

All narcotic indexes can by changed in the config file, in the [advanced] section

---

***Keyword arguments:*** depth – int – in meter
***Returns:*** end – int – equivalent narcotic depth in meter
***Raise:*** <nothing>

**consume_gas**(*gas_consumed*)
Consume gas inside this tank
***Keyword arguments:***
**gas_consumed** (float) – gas consumed in liter
***Returns:*** float – remaining gas in liter
***Raise:*** <nothing>

**refill**()
Refill the tank
***Keyword arguments:*** <none>

***Returns:*** float – remaining gas in liter
***Raise:*** <nothing>

**check_rule**()
Checks the rule agains the remaining gas in the tank
***Keyword arguments:***
**gas_consumed** (float) – gas consumed in liter
***Returns:***
**bool – True is rule OK** False if rule Not OK
***Raise:*** <nothing>

## 5.4 model: buhlmann

### 5.4.1 Exceptions

Defines Exceptions for buhlmann model

**exception** dipplanner.model.buhlmann.model_exceptions.**ModelException**(*description*)
Generic Model Exception

**__init__**(*description*)
constructor : call the upper constructor and set the logger
***Keyword Arguments:***
**description** (str) – text describing the error
***Return:*** <nothing>
***Raise:*** <nothing>

**__str__**()
String representing the object
***Keyword Arguments:*** <none>
***Return:*** str – a string describing the Exception
***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object
***Keyword Arguments:*** <none>
***Return:*** ustr – a unicode string describing the Exception
***Raise:*** <nothing>

**exception** dipplanner.model.buhlmann.model_exceptions.**ModelStateException**(*description*)
Model State Exception

**__init__**(*description*)
constructor : call the upper constructor and set the logger
***Keyword Arguments:***
**description** (str) – text describing the error
***Return:*** <nothing>
***Raise:*** <nothing>

**__str__**()
String representing the object
***Keyword Arguments:*** <none>
***Return:*** str – a string describing the Exception
***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object
***Keyword Arguments:*** <none>
***Return:*** ustr – a unicode string describing the Exception
***Raise:*** <nothing>

### 5.4.2 compartment

Defines a Buhlmann compartment

**class** dipplanner.model.buhlmann.compartment.**Compartment**(*h_he=None*, *h_n2=None*, *a_he=None*, *b_he=None*, *a_n2=None*, *b_n2=None*)
Buhlmann compartment class
***Attributes:***
- h_he: helium halftime
- h_n2: nitrogen halftime
- a_he: helium : a coefficient
- a_n2: nitrogen : a coefficient
- b_he: helium : b coefficient
- b_n2: nitrogen : b coefficient
- k_he: helium : k coefficient (calculated)
- k_n2: nitrogen : k coefficient (calculated)
- pp_he: partial pressure of helium
- pp_n2: partial pressure of nitrogen

**__init__**(*h_he=None*, *h_n2=None*, *a_he=None*, *b_he=None*, *a_n2=None*, *b_n2=None*)
Constructor for Compartment can be called without params, in this case, does not initiate anything can be called with time params and coef and in this case initate the compartment time constants

***Keyword arguments:***

**h_he** (float) – helium halftime

**h_n2** (float) – nitrogen halftime

**a_he** (float) – helium: a coefficient

**b_he** (float) – helium: b coefficient

**a_n2** (float) – nitrogen: a coefficient

**b_n2** (float) – nitrogen: b coefficient

***Returns:*** <nothing>

***Raise:*** see set_compartment_time_constants method

**__deepcopy__**(*memo*)

deepcopy method will be called by copy.deepcopy

Used for "cloning" the object into another new object.

***Keyword Arguments:***

**memo** – not used here

***Returns:*** Compartment – Compartment object copy of itself

***Raise:*** <nothing>

**__str__**()

Return a human readable name of the segment

***Keyword Arguments:*** <none>

***Returns:***

**str – string representation of the compartment** ex: "He:0.0 N2:19.9991340314 gf:0.3 mv_at:2.98611129437 max_amb:15.8731803417 MV:6.6973840088"

***Raise:*** <nothing>

**__unicode__**()

Return a human readable name of the segment in unicode

***Keyword Arguments:*** <none>

***Returns:***

**ustr – unicode string representation of the compartment** ex: u"He:0.0 N2:19.9991340314 gf:0.3 mv_at:2.98611129437 max_amb:15.8731803417 MV:6.6973840088"

***Raise:*** <nothing>

**set_compartment_time_constants**(*h_he*, *h_n2*, *a_he*, *b_he*, *a_n2*, *b_n2*)

Sets the compartment's time constants

***Keyword arguments:***

**h_he** (float) – Helium Halftime

**h_n2** (float) – Nitrogen Halftime

**a_he** (float) – Helium : a coefficient

**b_he** (float) – Helium : b coefficient

**a_n2** (float) – Nitrogen : a coefficient

**b_n2** (float) – Nitrogen : b coefficient

***Returns:*** <nothing>

***Raise:*** <nothing>

**set_pp**(*pp_he*, *pp_n2*)

Sets partial pressures of He and N2

***Keyword arguments:***

**pp_he** (float) – partial pressure of Helium

**pp_n2** (float) – partial pressure of Nitrogen

***Return:*** <nothing>

***Raise:*** <nothing>

**const_depth**(*pp_he_inspired*, *pp_n2_inspired*, *seg_time*)

Constant depth calculations. Uses instananeous equation: P = Po + (Pi - Po)(1-e^-kt) store the new values in self.pp_he and self.pp_n2

***Keyword arguments:***

**pp_he_inspired** (float) – partial pressure of inspired helium

**pp_n2_inspired** (float) – partial pressure of inspired nitrogen

**seg_time** (float) – segment time in seconds

***Return:*** <nothing>

***Raise:*** ModelStateException – if pp or time < 0

**asc_desc**(*pp_he_inspired*, *pp_n2_inspired*, *rate_he*, *rate_n2*, *seg_time*)

Ascend or descent calculations. Uses equation : P=Pio+R(t -1/k)-[Pio-Po-(R/k)]e^-kt store the new values in self.pp_he and self.pp_n2

***Keyword arguments:***

**pp_he_inspired** (float) – partial pressure of inspired helium

**pp_n2_inspired** (float) – partial pressure of inspired nitrogen

**rate_he** (float) – rate of change of pp_he

**rate_n2** (float) – rate of change of pp_n2

**seg_time** (float) – segment time in seconds

***Return:*** <nothing>

***Raise:*** ModelStateException – if pp or time < 0

**get_m_value_at**(*pressure*)

Gets M-Value for given ambient pressure using the Buhlmann equation Pm = Pa/b +a where:

- Pm = M-Value pressure,
- Pa = ambiant pressure
- a,b co-efficients

---

**Note:** Not used for decompression but for display of M-value limit line

---

---

**Note:** this method does not use gradient factors.

---

***Keyword arguments:***

**pressure** (float) – ambient pressure (in bar)

***Return:*** float – M_value: maximum tolerated pressure in bar

***Raise:*** <nothing>

**get_max_amb**(*gf*)

Gets Tolerated Absolute Pressure for the compartment for current pp of He and N2

***Keyword arguments:***

**gf** (float) – gradient factor : 0.1 to 1.0, typical 0.2 - 0.95

***Return:*** float – maximum tolerated pressure (absolute) in bar

***Raise:*** <nothing>

**get_mv**(*p_amb*)

Gets M-Value for a compartment, given an ambient pressure

***Keyword arguments:***

**p_amb** (float) – ambiant pressure

***Return:*** float – M-value
***Raise:*** <nothing>

### 5.4.3 gradient

gradient module

**class** dipplanner.model.buhlmann.gradient.**Gradient**(*gf_low*, *gf_high*)
Defines a Gradient Factor object

A GF Object maintains a low and high setting and is able to determine a GF for any depth between its initialisation depth (see setGfAtDepth()) and the surface.
***Attributes (self.):***
- gf_low (float) – low Gradient factor, from 0.0 to 1.0
- gf_high (float) – high Gradient factor, from 0.0 to 1.0
- gf (float) – current gf
- gf_slope (float) – slope of the linear equation
- gf_set (float) – Indicates that gf Slope has been initialised

**__init__**(*gf_low*, *gf_high*)
Constructor for Gradient object
***Keyword arguments:***
**gf_low** (float) – low Gradient factor, from 0.0 to 1.0
**gf_high** (float) – high Gradient factor, from 0.0 to 1.0
***Returns:*** <nothing>
***Raise:*** ValueError – if either gf_low of gf_high has wrong value

**__deepcopy__**(*memo*)
deepcopy method will be called by copy.deepcopy

Used for "cloning" the object into another new object.
***Keyword Arguments:***
**memo** – not used here
***Returns:*** Gradient – Gradient object copy of itself
***Raise:*** <nothing>

**get_gradient_factor**()
Returns current GF with bounds checking if gf < gf_low, returns gf_low
***Keyword arguments:*** <none>
***Returns:*** float – gf
***Raise:*** <nothing>

**set_gf_at_depth**(*depth*)
Sets the gf for a given depth. Must be called after setGfSlope() has initialised slope
***Keyword arguments:***
**depth** (float) – current depth, in meter
***Returns:*** <nothing>
***Raise:*** <nothing>

**set_gf_slope_at_depth**(*depth*)
Set gf Slope at specified depth. Typically called once to initialise the GF slope.
***Keyword arguments:***
**depth** (float) – current depth, in meter
***Returns:*** <nothing>
***Raise:*** <nothing>

**set_gf_low**(*value*)
Sets gf low setting

***Keyword arguments:***
**value** (float) – low Gf, between 0.0 and 1.0
***Returns:*** <nothing>
***Raise:*** ValueError – if either gf_low of gf_high has wrong value

**set_gf_high**(*value*)
Sets gf high setting
***Keyword arguments:***
**value** (float) – high Gf, between 0.0 and 1.0
***Returns:*** <nothing>
***Raise:*** ValueError – if gf_high has wrong value

**__str__**
x.__str__() <==> str(x)

### 5.4.4 oxygen toxicity

Oxygen Toxicity model

**class** dipplanner.model.buhlmann.oxygen_toxicity.**OxTox**
Defines a Oxygen Toxicity model
***Attributes:***
- cns (float) – central nervous system toxicity

  (see http://en.wikipedia.org/wiki/Oxygen_toxicity#Signs_and_symptoms)
- otu (float- – Oxygen toxicity Units
- max_ox (float) – maximum ppo2

**__init__**()
Constructor for OxTox class
***Keyword arguments:*** <none>
***Returns:*** <nothing>
***Raise:*** <nothing>

**__deepcopy__**(*memo*)
deepcopy method will be called by copy.deepcopy

Used for "cloning" the object into another new object.
***Keyword Arguments:***
**memo** – not used here
***Returns:*** Gradient – Gradient object copy of itself
***Raise:*** <nothing>

**add_o2**(*time*, *pp_o2*)
Adds oxygen load into model.

Uses NOAA lookup table to add percentage based on time and ppO2. Calculate OTU using formula OTU= T * (0.5/(pO2-0.5))^-(5/6)

this OTU formula need T (time) in minutes, so we need to convert the time in second to minutes while using this formula
***Keyword arguments:***
**pp_o2** (float) – partial pressure of oxygen

**time** (float) – time of segment (in seconds)
***Returns:*** <nothing>
***Raise:*** <nothing>

**remove_o2**(*time*)
Removes oxygen load from model during surface intervals

***Keyword arguments:***
**time** (float) – time of segment (in seconds)
***Returns:*** <nothing>
***Raise:*** <nothing>

**__str__**
x.__str__() <==> str(x)

### 5.4.5 model

Buhlmann model module

Contains: Model – class

**class** dipplanner.model.buhlmann.model.**Model**
Represents a Buhlmann model. Composed of a tissue array of Compartment[] Has an OxTox and Gradient object

Can throw a ModelStateException propagated from a Compartment if pressures or time is out of bounds.

Models are initialised by initModel() if they are new models, or validated by validateModel() if they are rebuild from a saved model.

The model is capable of ascending or descending via ascDec() using the ascDec() method of Compartment, or accounting for a constant depth using the constDepth() method of Compartment.
**Attributes:**
- tissues (list)– a list of Compartments
- gradient (Gradient) – gradient factor object
- ox_tox (OxTox) – OxTox object
- metadata (str) – Stores infos about where the model was created
- units (str) – only 'metric' allowed
- COMPS (int) – static info : number of compartments
- MODEL_VALIDATION_SUCCESS (int) – static const for validation success
- MODEL_VALIDATION_FAILURE (int) – static const for validation failure

**__init__**()
Constructor for model class
***Keyword arguments:*** <none>
***Returns:*** <nothing>
***Raise:*** <nothing>

**__deepcopy__**(*memo*)
deepcopy method will be called by copy.deepcopy

Used for "cloning" the object into another new object.
***Keyword Arguments:***
**memo** – not used here
***Returns:*** Model – Model object copy of itself
***Raise:*** <nothing>

**__str__**()
Return a human readable name of the segment
***Keyword Arguments:*** <none>
***Returns:*** str – string representation of the model
***Raise:*** <nothing>

**__unicode__**()
Return a human readable name of the segment in unicode
***Keyword Arguments:*** <none>

***Returns:*** ustr – unicode string representation of the model
***Raise:*** <nothing>

**init_gradient**()
Initialise the gradient attribute uses the default settings parameters for gf_low and high
***Keyword arguments:*** <none>
***Returns:*** <nothing>
***Raise:*** <nothing>

**set_time_constants**(*deco_model='ZHL16c'*)
Initialize time constants in buhlmann tissue list Only for metric values
***Keyword arguments:***
**deco_model** (str) – "ZHL16b" or "ZHL16c"
***Returns:*** <nothing>
***Raise:*** <nothing>

**validate_model**()
Validate model - checks over the model and looks for corruption

This is needed to check a model that has been loaded from XML Resets time constants
***Keyword arguments:*** <none>
***Returns:*** self.MODEL_VALIDATION_SUCCESS – if OK self.MODEL_VALIDATION_FAILURE – if not OK
***Raise:*** <nothing>

**control_compartment**()
Determine the controlling compartment at ceiling (1-16)
***Keyword arguments:*** <none>
***Returns:***
**integer – reference number of the controlling** compartment (between 1 to 16)
***Raise:*** <nothing>

**ceiling**()
Determine the current ceiling depth
***Keyword arguments:*** <none>
***Returns:*** float – ceiling depth in meter
***Raise:*** <nothing>

**ceiling_in_pabs**()
Determine the current ceiling
***Keyword arguments:*** <none>
***Returns:*** float – ceiling in bar (absolute pressure)
***Raise:*** <nothing>

**m_value**(*pressure*)
Determine the maximum M-Value for a given depth (pressure)
***Keyword arguments:***
**pressure** (float) – in bar
***Returns*** float – max M-Value
***Raise:*** <nothing>

**const_depth**(*pressure*, *seg_time*, *f_he*, *f_n2*, *pp_o2*)
Constant depth profile. Calls Compartment.constDepth for each compartment to update the model.

*Keyword arguments:*
**pressure** (float)– pressure of this depth of segment in bar

**seg_time** (float) – Time of segment in seconds

**f_he** (float) – fraction of inert gas Helium in inspired gas mix

**f_n2** (float) – fraction of inert gas Nitrogen in inspired gas mix

**pp_o2** (float) – For CCR mode, partial pressure of oxygen in bar. If == 0.0, then open circuit

***Returns:*** <nothing>
***Raise:*** ModelStateException

**asc_desc**(*start*, *finish*, *rate*, *f_he*, *f_n2*, *pp_o2*)
Ascend/Descend profile. Calls Compartment.asc_desc to update compartments

*Keyword arguments:*

**start** (float) – start pressure of this segment in bar (WARNING: not meter ! it's a pressure)

**finish** (float) – finish pressure of this segment in bar (WARNING: not meter ! it's a pressure)

**rate** (float) – rate of ascent or descent in m/s

**f_he** (float) – Fraction of inert gas Helium in inspired gas mix

**f_n2** (float) – Fraction of inert gas Nitrogen in inspired gas mix

**pp_o2** (float) – For CCR mode, partial pressure of oxygen in bar. If == 0.0, then open circuit

***Returns:*** <nothing>
***Raise:*** ModelStateException

## 5.5 Exceptions

Base Class for exceptions for dipplanner module

**exception** dipplanner.dipp_exception.**DipplannerException**(*description*)
Base exception class for dipplanner

**__init__**(*description*)
DipplannerException constructor
***Keyword Arguments:***
**description** (str) – text describing the error
***Return:*** <nothing>
***Raise:*** <nothing>

**__str__**()
String representing the object
***Keyword Arguments:*** <none>
***Return:*** str – a string describing the Exception
***Raise:*** <nothing>

**__unicode__**()
unicode string representing the object
***Keyword Arguments:*** <none>
***Return:*** ustr – a unicode string describing the Exception
***Raise:*** <nothing>

## 5.6 tools

tools modules

This modules contains some utility functions like unit conversions, etc...

dipplanner.tools.**seconds_to_mmss**(*seconds*)

Convert a value in seconds into a string representing the time in minutes and seconds (like 2:06) It does returns only minutes and seconds, not hours, minutes and seconds

***Keyword Arguments:***

**seconds** (float) – the duration in seconds

***Returns:***

**str – the time in minutes and seconds** ex: ” 23:45” “112:33” ...

***Raise:*** ValueError: when bad time values

dipplanner.tools.**seconds_to_hhmmss**(*seconds*)

Convert a value in seconds into a string representing the time in hour:minutes and seconds like 22:34:44

***Keyword Arguments:***

**seconds** (float) – the duration in seconds

***Returns:***

**str – the time in hour - minutes and seconds** ex: “00:23:45” “01:52:33”

***Raise:*** ValueError: when bad time values is given

dipplanner.tools.**altitude_or_depth_to_absolute_pressure**(*altitude_or_depth*)

output absolute pressure for give “depth” in meter.

•If depth is positive it’s considered altitude depth

•If depth is negative it’s considered depth in water

***Keyword Arguments:***

**altitude_or_depth** (float (signed)) – in meter

***Returns:*** float – resulting absolute pressure in bar

***Raise:*** ValueError if altitude > 10000m

dipplanner.tools.**altitude_to_pressure**(*altitude*)

Convert a given altitude in pressure in bar uses the formula: p = 101325.(1-2.25577.10^-5.h)^5.25588

***Keyword Arguments:***

**altitude** (float) – current altitude in meter

***Returns:*** float – resulting pressure in bar

***Raise:*** ValueError: when bad altitude is given (bad or <0 or > 10000 m)

dipplanner.tools.**depth_to_pressure**(*depth*, *method='complex'*)

Calculates depth pressure based on depth using a more complex method than only /10

***Keyword Arguments:***

**depth** (float) – in meter

***Returns:*** float – depth pressure in bar

***Raise:*** <nothing>

dipplanner.tools.**pressure_to_depth**(*pressure*, *method='complex'*)

calculates depth based on give pressure using a more complex method than only *10

***Keyword Arguments:***

**pressure** (float) – pressure in bar

***Returns:*** float – depth in meter

***Raise:*** <nothing>

dipplanner.tools.**calculate_pp_h2o_surf**(*temperature=20*)

Calculates and return vapor pressure of water at surface using Antoine equation (http://en.wikipedia.org/wiki/Vapour_pressure_of_water)

***Keyword Arguments:***

**temperature** (float) [OPTIONNAL] – in ° Celcius

***Returns:*** float – ppH2O in bar

***Raise:***

**ValueError – when temperature exceed maximum value for calculation** (>=374 °C)

dipplanner.tools.**convert_bar_to_psi**(*value*)

SI –> imperial pressure conversion function

***Keyword Arguments:***

**value** (float) – pressure in bar

***Returns:*** float – pressure in psi

***Raise:*** <nothing>

dipplanner.tools.**convert_psi_to_bar**(*value*)

imperial –> SI pressure conversion function

***Keyword Arguments:***

**value** (float) – pressure in psi

***Returns:*** float – pressure in bar

***Raise:*** <nothing>

dipplanner.tools.**convert_liter_to_cubicfeet**(*value*)

SI –> imperial volume conversion function

***Keyword Arguments:***

**value** (float) – volume in liter

***Returns:*** float – volume in cubicfeet

***Raise:*** <nothing>

dipplanner.tools.**convert_cubicfeet_to_liter**(*value*)

imperial –> SI volume conversion function

***Keyword Arguments:***

**value** (float) – volume in cubicfeet

***Returns:*** float – volume in liter

***Raise:*** <nothing>

dipplanner.tools.**convert_meter_to_feet**(*value*)

SI –> imperial distance conversion function

***Keyword Arguments:***

**value** (float) – volume in cubicfeet

***Returns:*** float – volume in liter

***Raise:*** <nothing>

dipplanner.tools.**convert_feet_to_meter**(*value*)

imperial –> SI distance conversion function

***Keyword Arguments:***

**value** (float) – volume in cubicfeet

***Returns:*** float – volume in liter

***Raise:*** <nothing>

## 5.7 main

main dipplanner module for command line usage.

This module is used by the only "executable" of the project: bin/dipplanner (which is an empty shell)

runs in command line and output resulting dive profile also initiate log files

dipplanner.main.**activate_debug**()

setup the default debug parameters

it's mainly used for test cases who needs also logging to be set
***Keyword Arguments:*** <none>
***Return:*** <nothing>
***Raise:*** <nothing>

dipplanner.main.**activate_debug_for_tests**()
setup the default debug parameters

it's mainly used for test cases who needs also logging to be set
***Keyword Arguments:*** <none>
***Return:*** <nothing>
***Raise:*** <nothing>

dipplanner.main.**parse_config_file**(*filenames*)
parse a config file and change default settings values
***Keyword Arguments:***
**filename** (str) – name (and path) of the config file to be parsed
***Returns:*** <nothing>
***Raise:*** Nothing, but can exit

dipplanner.main.**parse_arguments**()
parse all command lines options

could also exit from program because of wrong arguments
***Keyword Arguments:*** <none>
***Returns:***
**a tuple of two dicts:**
**args**

**dives**

- args is the result of argpaser
- dives dict is in the following form:

```
dives = { 'dive1': { 'tanks': {},
                     'segments': {},
                     'surface_interval':0 },
          'dive2': { 'tanks': {},
                     'segments': {},
                     'surface_interval':60 }}
```

***Raise:*** Nothing, but can exit

dipplanner.main.**main**()
main uses the parameters, tanks and dives given in config file(s) and/or command line, calculates the dives and return the output in stdout.
***Keyword Arguments:*** <none>
***Return:*** <nothing>
***Raise:*** <nothing>

## 5.8 settings

Global settings for dipplannerand their default values

All the settings can be changed by *dipplanner command-line documentation* and/or *dipplanner config file documentation*

dipplanner.settings.**FRESH_WATER_DENSITY = 1.0**
water density kg/l

dipplanner.settings.**SEA_WATER_DENSITY = 1.03**
water density kg/l

dipplanner.settings.**ABSOLUTE_MAX_PPO2 = 2.0**
in bar

dipplanner.settings.**ABSOLUTE_MIN_PPO2 = 0.16**
in bar

dipplanner.settings.**ABSOLUTE_MAX_TANK_PRESSURE = 300**
in bar

dipplanner.settings.**ABSOLUTE_MAX_TANK_SIZE = 40**
in liter

dipplanner.settings.**SURFACE_TEMP = 20**
Temperature at surface. Used to calculate PP_H2O_SURFACE

dipplanner.settings.**HE_NARCOTIC_VALUE = 0.23**
helium narcotic value

dipplanner.settings.**N2_NARCOTIC_VALUE = 1.0**
nitrogen narcotic value

dipplanner.settings.**O2_NARCOTIC_VALUE = 1.0**
oxygen narcotic value

dipplanner.settings.**AR_NARCOTIC_VALUE = 2.33**
argon narcotic value

dipplanner.settings.**STOP_DEPTH_INCREMENT = 3**
in meter

dipplanner.settings.**LAST_STOP_DEPTH = 3**
in meter : last stop before surfacing

dipplanner.settings.**STOP_TIME_INCREMENT = 1**
in second

dipplanner.settings.**FORCE_ALL_STOPS = True**
once deco stop begun, force to stop to each deco depth stop

dipplanner.settings.**AMBIANT_PRESSURE_SEA_LEVEL = 1.01325**
surface pressure (in bar)

dipplanner.settings.**METHOD_FOR_DEPTH_CALCULATION = 'complex'**
either simple (/10) or complex

dipplanner.settings.**TRAVEL_SWITCH = 'late'**
"late" or "early"

dipplanner.settings.**FLIGHT_ALTITUDE = 2450**
in meter practicly, this value can not be bigger than about 2850m because breathing air at 0m will only 'prepare the body' to a decompression until this value. To go higher, another 'stop' is needed between.

dipplanner.settings.**TEMPLATE = 'default-color.tpl'**
template should be in templates/ directory

dipplanner.settings.**DECO_MODEL = 'ZHL16c'**
ZHL16c or ZHL16b

dipplanner.settings.**WATER_DENSITY = 1.03**
water density kg/l

dipplanner.settings.**AMBIANT_PRESSURE_SURFACE = 1.01325**
surface pressure (in bar)

dipplanner.settings.**DEFAULT_MAX_PPO2 = 1.6**
in bar

dipplanner.settings.**DEFAULT_MIN_PPO2 = 0.21**
in bar

dipplanner.settings.**DEFAULT_MAX_END = 30**
in meter

dipplanner.settings.**DIVE_CONSUMPTION_RATE = 0.2833333333333333**
liter/s

dipplanner.settings.**DECO_CONSUMPTION_RATE = 0.2**
liter/s

dipplanner.settings.**DESCENT_RATE = 0.3333333333333333**
m/s

dipplanner.settings.**ASCENT_RATE = 0.16666666666666666**
m/s

dipplanner.settings.**RUN_TIME = True**
Warning : if RUN-TIME is True, the segment duration must include descent time if the duration is too small dipplanner will raise an error if True: segments represents runtime, if false, segments represents segtime

dipplanner.settings.**USE_OC_DECO = True**
if True, use enabled gases of decomp in oc or bailout

dipplanner.settings.**GF_LOW = 0.3**
% between 0.0 and 1.0

dipplanner.settings.**GF_HIGH = 0.8**
% between 0.0 and 1.0

dipplanner.settings.**MULTILEVEL_MODE = False**
TO CHECK

dipplanner.settings.**AUTOMATIC_TANK_REFILL = True**
automatic refill of tanks between dives

CHAPTER

SIX

# RELEASE NOTES

## 6.1 v0.3 (ongoing)

**Note:** v0.3 is not released yet.

Below are the feature already developped for the 0.3 target release.

See GitHub Issues for more infos.

### 6.1.1 New features

- No-flight time calculation
- non-blocking dive situations errors

### 6.1.2 Bug corrections

## 6.2 v0.2

### 6.2.1 New features

- New submodel: buhlmann ZH-L16C
- Handle repetitive dives
- Variable ppH2O in surface based on Temp and % humidity
- Limit CCR dive depth based on diluant values
- Export dive plannification in different format
- Config Files
- **Tanks:**
  - More accurate calculation of tank volume
  - minimum gas remaining rules in tanks
  - handle double tanks

### 6.2.2 Bug corrections

- Automatic gas selection was not well handled on Hypoxic trimix dives

## 6.3 v0.1

### 6.3.1 New features

- **Same 'base' functionnality as MVPlan (v1.5):**
  - buhlmann ZH-L16B
  - mv-value gradient conservatism (adjustable)
  - oxygen toxicity (OTU and CNS) calculation
  - support Open Circuit dives : air, nitrox, trimix, heliox...
  - support CCR dives with OC deco/bailout if wanted
  - support dive in altitude
  - automatic selection of deco when ascending
  - ...
- **But some differences:**
  - command line only (or direct use in python) : no GUI, no automatic update
  - only in english
  - **the unit system is only SI:**
    - meter
    - seconds
    - bar (almost SI: 1 bar = 10E5 pascal)
    - liter (dm3)
  - Uses 'Tanks' instead of 'Gases' in order to calculate gas consumption for each tank and raise error when dive reach empty tank
  - All time parameters and calculations are by default in seconds instead of minutes
  - the END 'Equivalent Narcosis depth' is calculated based on narcosis index of every breathed gases (N2, He and O2) instead of the 'Only N2 is narcotic' in MVPlan
  - water pressure is by default calculated using the 'real pressure' of water (wich can change between fresh or sea water) instead of '10m == 1bar'

CHAPTER

SEVEN

# TODOS

**Todo**

check the usage of the multilevel_mode option in config-files

(The *original entry* is located in /home/docs/sites/readthedocs.org/checkouts/readthedocs.org/user_builds/dipplanner/checkouts/0.3a/doc
line 518.)

CHAPTER

EIGHT

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

# PYTHON MODULE INDEX

## d