

---

# **dipplanner Documentation**

***Release 0.3a-40-g1237***

**Thomas Chiroux**

June 17, 2013



# CONTENTS

<b>1</b>	<b>dipplanner Quick Start</b>	<b>3</b>
1.1	Description . . . . .	3
1.2	Get the sources . . . . .	5
1.3	Installing . . . . .	5
1.4	Using . . . . .	5
1.5	Units : SI or Imperial . . . . .	6
1.6	References . . . . .	7
1.7	Open Source and Licence . . . . .	7
<b>2</b>	<b>dipplanner command-line documentation</b>	<b>9</b>
2.1	help . . . . .	9
2.2	version . . . . .	10
2.3	config files . . . . .	10
2.4	tanks and segments . . . . .	10
2.5	dive parameters . . . . .	12
2.6	Advanced Parameters . . . . .	15
2.7	Output Parameters . . . . .	16
<b>3</b>	<b>dipplanner config file documentation</b>	<b>17</b>
3.1	tanks . . . . .	17
3.2	dive profiles . . . . .	19
3.3	segments . . . . .	20
3.4	Controlling the output . . . . .	21
3.5	general dive parameters . . . . .	22
3.6	advanced dive parameters . . . . .	26
3.7	Examples . . . . .	29
<b>4</b>	<b>Templates</b>	<b>37</b>
4.1	mission . . . . .	37
4.2	dives . . . . .	37
4.3	settings . . . . .	38
<b>5</b>	<b>REST Api</b>	<b>39</b>
5.1	Common errors . . . . .	39
5.2	/api/v1/mission/ . . . . .	40
5.3	/api/v1/mission/status . . . . .	43
5.4	/api/v1/mission/calculate . . . . .	43
5.5	/api/v1/missiondives/ . . . . .	44
5.6	/api/v1/missiondives/<dive_id>/ . . . . .	46

5.7	/api/v1/mission/dives/<dive_id>/tanks/	49
5.8	/api/v1/mission/dives/<dive_id>/tanks/<tank_id>/	54
5.9	/api/v1/mission/dives/<dive_id>/input_segments/	59
5.10	/api/v1/mission/dives/<dive_id>/input_segments/<segment_id>/	62
5.11	/api/v1/mission/dives/<dive_id>/output_segments/	67
5.12	/api/v1/mission/dives/<dive_id>/output_segments/<segment_id>/	68
5.13	/api/v1/settings/	70
5.14	/api/v1/settings/EACHSETTINGSATTRIBUTE/	70
<b>6</b>	<b>Source code documentation</b>	<b>71</b>
6.1	mission	71
6.2	dive	74
6.3	segment	80
6.4	tank	90
6.5	model: buhlmann	98
6.6	Exceptions	107
6.7	tools	108
6.8	main	110
6.9	settings	111
6.10	gui	113
<b>7</b>	<b>Release Notes</b>	<b>119</b>
7.1	v0.3 (ongoing)	119
7.2	v0.2	119
7.3	v0.1	120
<b>8</b>	<b>TODOs</b>	<b>121</b>
<b>9</b>	<b>Indices and tables</b>	<b>123</b>
	<b>Python Module Index</b>	<b>125</b>

**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



# 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

- Bühlmann ZH-L16B with Gradient Factor
- Bühlmann 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 principle 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 ppH<sub>2</sub>O)
  - ...
- 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: virtualenv

It is recommended (but optional) to setup a virtualenv 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 automatically chosen 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 override 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 output 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

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/gpl.html>



# DIPPLANNER COMMAND-LINE DOCUMENTATION

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

```
dipplanner [-h] [--help]
            [--version]
            [-c] [--config]
            [--surfaceinterval]
            [--model]
            [--gflow]
            [--gfhigh]
            [--water]
            [--altitude]
            [--diveconstrate]
            [--decoconstrate]
            [--descentrate]
            [--ascentrate]
            [--maxppo2]
            [--minppo2]
            [--maxend]
            [--samegasfordeco]
            [--multilevel]
            [--automatictankrefill] [--notankrefill]
            [--forcesegmenttime]
            [--depthcalcmethod]
            [--travelswitch]
            [--surfacetemp]
            [--ambientpressureatsea]
            [--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 help message and exit

## 2.2 version

### **-version**

show version number and exit

## 2.3 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 occurrence 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)

---

## 2.4 tanks and segments

### **-t** <STRING>, **-tank** <STRING>

specify a tank that will be used for the dive

#### •Short Format

`"tank_name"`

`-tank_name:` (str) name for the tank

In the short format, you only refer to an existing tank for the dive. So, the Tank MUST be defined before in one of the config file(s)

#### •Long 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\_o2** (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

•Examples:

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"
```

---

## 2.5 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 planned 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

```
diplanner --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

```
diplanner --maxppo2=1.4
```

**-minppo2=<VALUE>**

minimum allowed ppo2 for this dive.

Default: 0.21

Example:

Set the min allowed ppo2 at 0.19

```
diplanner --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

```
diplanner --maxend=35
```

---

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

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

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

---

**-forcesegmenttime**

if set, each input segment will be done at the full time of the segment.

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

Example:

```
diplanner --forcesegmenttime
```

**-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
```

**-multilevel**

if set, switch dipplanner in multilevel mode.

TODO: complete this

**-automatictankrefill**

is set, the tanks will be automatically refilled between repetitive dives.

Example:

automatic tank refill between repetitive dives

```
dipplanner --automatictankrefill
```

**-notankrefill**

is set, the tanks will NOT be automatically refilled between repetitive dives.

Example:

no tank refill between repetitive dives

```
dipplanner --notankrefill
```

## 2.6 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
```

**-ambientpressureatsea=<VALUE>**

Change ambient pressure at sea level (float) (in bar)

Default: 1.01325 b

Example:

change ambient pressure at sea level to 1 bar

```
dipplanner --ambientpressureatsea=1.0
```

## 2.7 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
```

# 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 tanks

At first you'll need to specify all the tanks you're planning to use in the entire mission.

Later, for each dive, you'll only select one or several tanks previously defined

Tanks definitions are specific sections in the config file, in the form:

```
[tank:NAME]
```

where NAME represent the name you've choosen for the tank.

**Warning:** NAME of tanks MUST be unique in the entire mission.

---

**Hint:** you can provide in the config file as many **[tank:NAME]** sections as you want.

---

to define each tank, you can provide one of the following options:

**f\_o2 = <VALUE>**  
(float) fraction of oxygen in the tank. Between 0.0 and 1.0

**f\_he = <VALUE>**  
(float) fraction of helium in the tank. Between 0.0 and 1.0

**f\_n2 = <VALUE>**  
(float) fraction of nitrogen in the tank. Between 0.0 and 1.0

---

**Note:** you're only needed to specify two of the 3 gases to define the tank  
dipplanner will automatically complete the fraction of the third gas.

---

**Note:** None of the previous fraction of gas is mandatory.  
By default, dipplanner will create an Air Tank (f\_o2=0.21, f\_he=0.0, f\_n2=0.79)

---

**volume = <VALUE>**  
(float) Volume of the tank in liter

**pressure = <VALUE>**  
(float) Pressure of the tank in bar

**rule = <VALUE>**  
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

**max\_ppo2 = <VALUE>**  
(float) max\_ppo2 allowed for this tank use.

---

**Note:** if not provided, dipplanner will calculate the value automatically

---

**mod = <VALUE>**  
(float) maximum operating depth allowed for this tank use.

---

**Note:** if not provided, dipplanner will calculate the value automatically

---

Example:

Mission 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.

```
[tank:airtank]
f_o2=0.21
f_he=0
```

```
volume=12
pressure=230
rule=50b

[tank:decotank]
f_o2=0.80
f_he=0
volume=12
pressure=200
rule=30b
```

## 3.2 dive profiles

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

```
[dive:NAME]
```

where NAME represent a name for the dive (it can be a number if you want). The dives whill be processed in croissant order

To define a dive, you **MUST** specify which tank(s) this specific dive will use. The Tank(s) must be defined before in the [tank:XXX] sections.

---

**Hint:** you can provide in the config file as many **[dive:NAME]** sections as you want.

---

the [dive:XXXX] sections use the following options:

**tanks = <VALUE>**

list of the tank name you'll be using during this dive.

The names **MUST** match the name provided in the [tank:NAME] sections.

ex:

```
tanks = airtank, decotank
```

in this example, the dive will use both airtank and decotank

**surface\_interval = <VALUE>**

surface interval (in seconds)

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

**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

---

**Note:** if you use this parameter inside a [dive:NAME] section, it will override the global paramater for this specific Dive.

---

Example:

do not refill tank before this dive

```
automatic_tank_refill = false
```

## 3.3 segments

A segment belongs to a dive, so each segment section **MUST** refer to a dive.

The section structure must respect the following pattern:

```
[dive:DIVENAME:segment:SEGNAME]
```

dive:DIVENAME refers to the previously defined [dive:NAME] section

---

**Hint:** you can provide in the config file as many [dive:NAME:segment:SEGNAME] sections as you want.

---

SEGNAME is a name you choose for a segment, it can also be numbers

segment sections uses the following options:

**depth = <VALUE>**

(float) in meter

**time = <VALUE>**

(float) in seconds (operators are allowed like: '30 \* 60')

duration of the segment (in seconds)

**tank = <VALUE>**

(str)

name of the tank (the 'tank\_name' specified in -t option or in config files)

**set\_point = <VALUE>**

(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

```
[dive:1:segment:bottom]
depth = 30
time = 20*60
tank = airtank
set_point = 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

```
[dive:1:segment:1]
depth = 30
time = 20*60
tank = airtank
set_point = 0.0
```



```
[dive:1:segment:2]
depth = 20
time = 25*60
tank = airtank
set_point = 0.0
```

### 3.3.1 Examples

Full example with two subsequent dives, with a surface interval of 1h30 between the two

```
[tank:airtank]
```

```
f_o2=0.21
f_he=0
volume=15
pressure=230
rule=50b
```

```
[tank:decotank]
```

```
f_o2=0.80
f_he=0
volume=12
pressure=230
rule=50b
```

```
[dive:1]
```

```
tanks = airtank, decotank
```

```
[dive:1:segment:bottom]
```

```
depth = 30
time = 20*60
tank = airtank
set_point = 0.0
```

```
[dive:2]
```

```
surface_interval = 90*60
automatic_tank_refill = true
tanks = airtank
```

```
[dive:2:segment:bottom]
```

```
depth = 20
time = 30*60
tank = airtank
set_point = 0.0
```

## 3.4 Controlling 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
```

## 3.5 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 be 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 planned 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
```

## 3.6 advanced dive parameters

advanced 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<sup>3</sup>) 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 planification 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

**ambient\_pressure\_sea\_level = <VALUE>**

Change ambient pressure at sea level (float) (in bar)

Default: 1.01325 b

Example:

change ambient pressure at sea level to 1 bar

[advanced]

```
ambient_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)

---



## 3.7 Examples

### 3.7.1 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
# ===== tank profiles =====
# tanks are givent using [tank:NAME] section, where NAME represent a name
# for the tank.
# Tank names must be unique within the same Mission
# you can add as many tank sections as you want

[tank:airtank]
f_o2=0.21
f_he=0
volume=15
pressure=230
rule=50b

[tank:decotank]
f_o2=0.80
f_he=0
volume=12
pressure=230
rule=50b

# ===== dive profiles =====
# repetitive dives are given using [dive:NAME] section, where NAME represent a
# name (or number if you want) of a dive.
# the dives will be done in give, order.
# Dive names must be unique within the same Mission
# you can add as many dive sections as you want

[dive:1]

# define the tanks used in this dive: coma-separated tank names:
tanks = airtank, decotank

# ----- dive segment -----
# multiples segment are given using: [dive:NAME:segment:SEGNAME], where
# NAME represent the name (or number if you want) of the dive and
# SEGNAME represent the name (or number if you prefer) of the segment
# at least ONT segment (for a dive) is mandatory
[dive:1:segment:bottom]
depth = 30
time = 20*60
tank = airtank
set_point = 0.0

[dive:2]
# surface_interval (in seconds)
# for repetitive dives, you can specify the surface time between the previous
# dive and this dive
surface_interval = 60*60
```

```
# automatic tank refill: if set to true, used tanks for this dive will be
# refilled before the dive
automatic_tank_refill = true

tanks = airtank

# segment list for this dive. At least ONE segment is mandatory

[dive:2:segment:bottom]

depth = 20
time = 30*60
tank = airtank
set_point = 0.0
```

### 3.7.2 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

# ===== tank profiles =====
# tanks are givent using [tank:NAME] section, where NAME represent a name
# for the tank.
# Tank names must be unique within the same Mission
# you can add as many tank sections as you want

[tank:airtank]
f_o2=0.21
f_he=0
volume=15
pressure=230
rule=50b

[tank:decotank]
f_o2=0.80
f_he=0
volume=12
pressure=230
rule=50b

# ===== dive profiles =====
# repetitive dives are given using [dive:NAME] section, where NAME represent a
# name (or number if you want) of a dive.
# the dives will be done in give, order.
# Dive names must be unique within the same Mission
# you can add as many dive sections as you want
```

**[dive:1]**

```
# define the tanks used in this dive: coma-separated tank names:
tanks = airtank, decotank
```

```
# ----- dive segment -----
# multiples segment are given using: [dive:NAME:segment:SEGNAME], where
# NAME represent the name (or number if you want) of the dive and
# SEGNAME represent the name (or number if you prefer) of the segment
# at least ONT segment (for a dive) is mandatory
```

**[dive:1:segment:bottom]**

```
depth = 30
time = 20*60
tank = airtank
set_point = 0.0
```

**[dive:2]**

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

```
# automatic tank refill: if set to true, used tanks for this dive will be
# refilled before the dive
automatic_tank_refill = true
```

```
tanks = airtank
```

```
# segment list for this dive. At least ONE segment is mandatory
```

**[dive:2:segment:bottom]**

```
depth = 20
time = 30*60
tank = airtank
set_point = 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)
ambient_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
```

### 3.7.3 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.
```

#### [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)  
ambient_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
```





# 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 mission

the main object passed in template is the mission object. It contains:

- mission.tanks: the tank dictionary defined for the mission
- mission.dives: the dives dictionary
- mission.settings: the global settings for the mission

## 4.2 dives

Dives are in the mission. To easilly iterate dive, you can use the following snippet:

```
{% for dive in mission.dives.values() %}  
  
...  
  
{% endfor %}
```

Dive attributes are described here: [dive](#)

## 4.3 settings

settings attributes are described here: *settings*

# REST API

Diplanner REST API is a json API. All request and response MUST have a Content-Type: application/json

## 5.1 Common errors

The whole API share the same error handling

### 5.1.1 Bad content type (400)

This error is returned when the request did not use the application/json Content-Type.

ex:

```
$ curl -v -X GET http://127.0.0.1:8080/api/v1/mission/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 400 Bad Request
< Date: Mon, 01 Oct 2012 05:26:08 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 35
< Content-Type: application/json
<
* Closing connection #0
{"message": "400: Bad ContentType"}
```

### 5.1.2 Not found (404)

### 5.1.3 Method Not allowed (405)

This occurs when you try to use some method (GET, PUT, POST, PATCH, etc..) which is not defined in the API for this specific URI

ex:

## 5.2 /api/v1/mission/

### 5.2.1 method: GET

returns a json object with all the content of the mission

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 05:27:31 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 8389
< Content-Type: application/json
<
```

and the result contains (for example)

```
{"dives": [{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "run_time": 20.0, "run_time": 0.0, "time": 1800.0, "type": "const"}], "metadata": "Dive to 20.0 for 1740.0s\n"}
```

### 5.2.2 method: POST

creates a new mission object and loads the given mission details posted in the request.

---

**Note:** to be able to POST a mission, the current mission **MUST** be empty. you'll probably need to call DELETE first.

---

if the request is accepted and correctly processed, the API will return 201 CREATED and the json dumps of the created mission

ex:

```
$ curl -v -X POST -d @/tmp/mission.json -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
```

```
> Content-Length: 8389
> Expect: 100-continue
>
* Done waiting for 100-continue
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 05:46:46 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 2814
< Content-Type: application/json
<
```

and the result contains (for example):

```
{"dives": [{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "min_gas": 454.3577760377562, "remaining_gas": 1791.9917863155865}}, {"current_depth": 5867, "given_name": "Air", "min_gas": 454.3577760377562, "remaining_gas": 1791.9917863155865}], "is_complete": false}
```

## errors

### forbidden

this error is raised when you try to POST a new mission when the current mission is not empty : before posting a new mission structure, you MUST before call DELETE

ex:

```
$ curl -v -X POST -d @/tmp/mission.json -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/
* Could not resolve host: ; Erreur inconnue
* Closing connection #0
curl: (6) Could not resolve host: ; Erreur inconnue
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 403 Forbidden
< Date: Mon, 01 Oct 2012 05:33:05 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 90
< Content-Type: application/json
<
* Closing connection #0
{"message": "403: Forbidden: you MUST delete the current mission before create a new one"}
```

### 5.2.3 method: PATCH

Updates the current Mission object because Mission is essentially a list a dives, it's preferable to use POST on /mission/dives/ if you want to update the list of dives.

However, it's still possible to use PATCH on /mission/ if you want. This method can also update the description of the mission

ex:

```
$ curl -v -X PATCH -d '{ "description": "coucou" }' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 27
>
* upload completely sent off: 27 out of 27 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 16:16:18 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 8393
< Content-Type: application/json
<
```

with resulting content (for example, with empty dive list for readability)

```
{"dives": [], "description": "coucou"}
```

## 5.2.4 method: DELETE

erase the current Mission if the delete operation succeed, the api returns 200 OK with an empty mission structure

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/
* About
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 05:39:01 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 32
< Content-Type: application/json
<
* Closing connection #0
{"dives": [], "description": ""}
```

## 5.3 /api/v1/mission/status

### 5.3.1 method: GET

returns the current status of the mission

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/status
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/status HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 05:54:48 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 28
< Content-Type: application/json
<
* Closing connection #0
{"status": "Not Calculated"}
```

return may be one of the following results :

```
{ 'status': 'Not Calculated' }
```

or:

```
{ 'status': 'Calculated but Changed' }
```

or:

```
{ 'status': 'Calculated and Up to date' }
```

## 5.4 /api/v1/mission/calculate

### 5.4.1 method: POST

Calculate all the dives of the mission

ex:

```
$ curl -v -X POST -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/calculate
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/calculate HTTP/1.1
> User-Agent: curl/7.27.0
```

```
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 06:11:50 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 39
< Content-Type: application/json
<
* Closing connection #0
{"status": "Calculated and Up to date"}
```

## 5.5 /api/v1/mission/dives/

### 5.5.1 method: GET

returns a json object with all the dives of the mission

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 05:27:31 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 8389
< Content-Type: application/json
<
```

and the result contains (for example)

```
{"dives": [{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "max_o2": 1.6, "min_o2": 0.16, "min_ppo2": 0.16, "rule": "50b", "rule_type": "rule", "type": "rule", "value": 50}, "rule": "50b", "rule_type": "rule", "type": "rule", "value": 50, "status": "Calculated and Up to date"}]}
```

### 5.5.2 method: POST

add a dive in the mission the dive can be an empty dive if not data is sent with the POST. if data is sent, dipplanner will try to configure the dive with the datas

returns json structure of the created dive

ex 1 (with no data):



```
$ curl -v -X POST -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 15:24:40 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 300
< Content-Type: application/json
<
* Closing connection #0
{"current_depth": 0.0, "pp_o2": 0.0, "current_tank": {}, "output_segments": [], "run_time": 0, "tan
```

ex 2 (with dive data):

```
$ curl -v -X POST -d @/tmp/dive1.json -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0     0     0  --:--:-- --:--:-- --:--:--    0* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 4392
> Expect: 100-continue
>
* Done waiting for 100-continue
  0 4392    0     0    0     0     0     0  --:--:--  0:00:01 --:--:--    0} [data not shown]
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 15:28:03 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 1388
< Content-Type: application/json
<
{ [data not shown]
100 5780 100 1388 100 4392 1370 4335 0:00:01 0:00:01 --:--:-- 4339
* Closing connection #0
```

with the sended data:

```
{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": t
```

### 5.5.3 method: DELETE

delete all the dives of the mission

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 07:33:00 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 13
< Content-Type: application/json
<
* Closing connection #0
{"dives": []}
```

## 5.6 /api/v1/mission/dives/<div\_id>/

### 5.6.1 method: GET

returns a json object with the content of a specific dive

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 07:10:17 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 4392
< Content-Type: application/json
<
```

and the result contains (for example) :

```
{ "current_depth": 0, "pp_o2": 0.0, "current_tank": { "tank_rule": "50b", "max_ppo2": 1.6, "in_use": true }
```

## errors

### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 07:13:25 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.6.2 method: PATCH

Update a parameter for this specific dive. If a parameter is a list of objects (like tanks, and segments), the entire list will be overwritten by the PATCH method

ex : file /tmp/modified\_dive1.json contains:

```
{ "metadata": "Coucou", "model": "ZHL16B", "surface_interval": 1664, "is_repetitive_dive": true }
```

```
$ curl -v -X PATCH -d @/tmp/created_dive1.json -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/3
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0     0     0  --:--:-- --:--:-- --:--:--    0* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/3 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 1370
> Expect: 100-continue
>
* Done waiting for 100-continue
```

```
0 1370 0 0 0 0 0 0 --:--:-- 0:00:01 --:--:-- 0} [data not shown]
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 15:51:35 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 1370
< Content-Type: application/json
<
{ [data not shown]
100 2740 100 1370 100 1370 1355 1355 0:00:01 0:00:01 --:--:-- 1356
* Closing connection #0
```

which returns (for example):

```
{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": t
```

We can see in this example that “model” was not updated, because the given value is not a valid value. The API currently silently errors and update only valid value.

This behaviour may change in the future.

### 5.6.3 method: DELETE

delete a specific dive and returns the list of remaining dives

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 07:34:13 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 3974
< Content-Type: application/json
<
```

and the request content will be (for example):

```
{"dives": [{"current_depth": 0, "pp_o2": 0.0, "current_tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": t
```

#### errors

##### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:



## errors

### not found

If the given `dive_id` is not found, the API will return a simple 404:

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/t
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 17:09:05 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.7.2 method: POST

add a tank to this dive

Returns the json dump of the newly created Tank

ex:

```
$ curl -v -X POST -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/t
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 17:22:06 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 301
< Content-Type: application/json
<
* Closing connection #0
{"tank_rule": "30b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 12.0, "f_o2": 0.21,
```

ex2 (creation of a nitrox tank):

```
$ curl -v -X POST -d '{"tank_rule": "50b", "f_n2": 0.7, "tank_vol": 12.0, "f_o2": 0.3, "tank_pressur
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 117
>
* upload completely sent off: 117 out of 117 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 17:28:58 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 304
< Content-Type: application/json
<
* Closing connection #0
{"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.7, "tank_vol": 12.0, "f_o2": 0.3, "f_he": 0.0}
```

---

**Note:** There is a special feature about MOD : in the json structure, when creating tanks, you can specify the MOD in meter, but you can also ask dipplanner to calculate MOD automatically based on max\_ppo2 with the json parameter: { "mod": "auto" } If not given, dipplanner will use the default MOD (for air) and may raise an error if you're try to create a tank with a shallower MOD

---

## errors

### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X POST -d '{"tank_rule": "50b", "essure": 230.0, "f_he": 0.0 }' -H "Content-type: applicati
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/42/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 102
>
* upload completely sent off: 102 out of 102 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 17:38:39 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
```

```
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

### server error

If the given parameter are wrong and does not permits the creation of the tank, dipplanner will return error 500 with the reason of the error

ex:

```
$ curl -v -X POST -d '{"tank_rule": "50b", "f_n2": 0.7, "tank_vol": 12.0, "f_o2": 0.3, "tank_pressure": 1.0}'
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 102
>
* upload completely sent off: 102 out of 102 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 500 Internal Server Error
< Date: Mon, 01 Oct 2012 17:40:17 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 52
< Content-Type: application/json
<
* Closing connection #0
{"message": "500: MOD exceed maximum tolerable MOD"}
```

ex2:

```
$ curl -v -X POST -d '{"tank_rule": "50b", "f_n2": 0.7, "tank_vol": 12.0, "f_o2": 0.5, "tank_pressure": 1.0}'
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 102
>
* upload completely sent off: 102 out of 102 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 500 Internal Server Error
< Date: Mon, 01 Oct 2012 17:40:44 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 54
< Content-Type: application/json
<
```



```
* Closing connection #0
{"message": "500: Proportion of O2+He+N2 is not 100%"}
```

### 5.7.3 method: DELETE

delete all the tanks of this dive returns the new list of tanks (empty list in this case)

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 17:55:48 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 13
< Content-Type: application/json
<
* Closing connection #0
{"tanks": []}
```

#### errors

##### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/42/tanks/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 17:59:08 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
```

```
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.8 /api/v1/mission/dives/<dive\_id>/tanks/<tank\_id>/

### 5.8.1 method: GET

returns a specific tank for a specific dive

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/tan
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 17:08:03 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 315
< Content-Type: application/json
<
* Closing connection #0
{"tank_rule": "30b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 15.0, "f_o2": 0.21,
```

### errors

#### not found

If the given dive\_id or the given tank\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/tan
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 17:09:51 GMT
```

```
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/tan
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1/tanks/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 17:10:29 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 39
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: tank (42) not found"}
```

## 5.8.2 method: PATCH

Update parameter(s) for this specific tank.

ex 1: change the tank rule from “30b” to “50b”

```
$ curl -v -X PATCH -d '{"tank_rule": "50b"}' -H "Content-type: application/json" http://11:8080/api/v1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 19
>
* upload completely sent off: 19 out of 19 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 18:09:07 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 315
< Content-Type: application/json
<
* Closing connection #0
{"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 15.0, "f_o2": 0.21,
```

## errors

### not found

If the given dive\_id or the given tank\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X PATCH -d '{"tank_rule":"50b"}' -H "Content-type: application/json" http://127.0.0.1:8080
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/42/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 19
>
* upload completely sent off: 19 out of 19 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 18:10:39 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X PATCH -d '{"tank_rule":"50b"}' -H "Content-type: application/json" http://127.0.0.1:8080
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/tanks/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 19
>
* upload completely sent off: 19 out of 19 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 18:11:22 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: tank_id (42) not found"}
```

## server error

If the given parameter are wrong and does not permits the creation of the tank, dipplanner will return error 500 with the reason of the error

ex:

```
$ curl -v -X PATCH -d '{"tank_rule":"50b", "f_o2": 0.66, "f_n2": 0.34}' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/tanks/1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 47
>
* upload completely sent off: 47 out of 47 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 500 Internal Server Error
< Date: Mon, 01 Oct 2012 19:12:56 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 52
< Content-Type: application/json
<
* Closing connection #0
{"message": "500: MOD exceed maximum tolerable MOD"}
```

ex2:

```
$ curl -v -X PATCH -d '{"tank_rule":"50b", "f_o2": 0.66}' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/tanks/1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 33
>
* upload completely sent off: 33 out of 33 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 500 Internal Server Error
< Date: Mon, 01 Oct 2012 19:12:02 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 54
< Content-Type: application/json
<
* Closing connection #0
{"message": "500: Proportion of O2+He+N2 is not 100%"}
```

## 5.8.3 method: DELETE

delete the given tank returns the list of remaining tanks in this dive

ex (starting with a list of 3 AirTanks):

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 17:57:58 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 617
< Content-Type: application/json
<
* Closing connection #0
{"tanks": [{"tank_rule": "30b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 12.0, "t
```

## errors

### not found

If the given dive\_id or if the tank\_id is not found, the API will return a simple 404:

ex 1:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/42/tanks/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 18:00:07 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
```

## 5.9 /api/v1/mission/dives/<diverge\_id>/input\_segments/

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/input_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/input_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:04:02 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.9.2 method: POST

add an input\_segment to this dive

Returns the json dump of the newly created Segment

ex:

```
$ curl -v -X POST -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/input_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 21:15:07 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 106
< Content-Type: application/json
<
* Closing connection #0
{"tank": {}, "in_use": true, "setpoint": 0.0, "depth": 0.0, "run_time": 0.0, "time": 0.0, "type": "normal"}
```

ex 2:

```
$ curl -v -X POST -d '{"tank": {}, "depth": 25, "time": 2000}' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/1/input_segments/ HTTP/1.1
```



```

> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 40
>
* upload completely sent off: 40 out of 40 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 201 Created
< Date: Mon, 01 Oct 2012 21:14:19 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 405
< Content-Type: application/json
<
* Closing connection #0
{"tank": {"tank_rule": "30b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 12.0, "f_

```

## errors

### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:

```

$ curl -v -X POST -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST /api/v1/mission/dives/42/input_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:05:07 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}

```

### 5.9.3 method: DELETE

delete all the input\_segments of this dive returns the new list of input\_segments (empty list in this case)

ex:

```

$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/apis/1/input_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected

```

```
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1/input_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:23:33 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 22
< Content-Type: application/json
<
* Closing connection #0
{"input_segments": []}
```

## errors

### not found

If the given dive\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/input_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/42/input_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:24:02 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.10 /api/v1/mission/dives/<dive\_id>/input\_segments/<segment\_id>/

### 5.10.1 method: GET

returns a specific input\_segment for a specific dive

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1/input_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:25:58 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 427
< Content-Type: application/json
<
* Closing connection #0
{"tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 15.0, "f_o2": 0.21}}
```

## errors

### not found

If the given dive\_id or the given segment\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/input_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/input_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:26:45 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
```

```
> GET /api/v1/mission/dives/1/input_segments/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:27:10 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: segment (42) not found"}
```

## 5.10.2 method: PATCH

Update parameter(s) for this specific input\_segment.

ex: change the depth from 30m to 45m

```
$ curl -v -X PATCH -d '{ "depth": 45 }' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/input_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 15
>
* upload completely sent off: 15 out of 15 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:32:34 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 425
< Content-Type: application/json
<
* Closing connection #0
{"tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 15.0, "f_
```

## errors

### not found

If the given dive\_id or the given segment\_id is not found, the API will return a simple 404:

ex:

```
$ curl -v -X PATCH -d '{ "depth": 45 }' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/42
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
```

```

* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/42/input_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 15
>
* upload completely sent off: 15 out of 15 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:34:06 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}

```

ex 2:

```

$ curl -v -X PATCH -d '{"depth": 45}' -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/42
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PATCH /api/v1/mission/dives/1/input_segments/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 15
>
* upload completely sent off: 15 out of 15 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:34:36 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 45
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: segment_id (42) not found"}

```

### 5.10.3 method: DELETE

delete the given input\_segment returns the list of remaining input\_segments in this dive

ex (starting with a list of 4 input\_segments):

```

$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/input_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1/input_segments/1 HTTP/1.1

```

```
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:36:55 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 344
< Content-Type: application/json
<
* Closing connection #0
{"input_segments": [{"tank": {}, "in_use": true, "setpoint": 0.0, "depth": 0.0, "run_time": 0.0, "t
```

## errors

### not found

If the given dive\_id or if the segment\_id is not found, the API will return a simple 404:

ex 1:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/42/input_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:37:42 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X DELETE -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> DELETE /api/v1/mission/dives/1/input_segments/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
```

Output segments are the result of the calculations, so they are read-only

return the calculated output\_segments

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/missiondives/1/output
* About to connect() to 127.0.0.1 port 8080 (#0)
* Trying 127.0.0.1...
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left     Speed

  0      0    0     0    0     0     0      0      0 --:--:-- --:--:-- --:--:--    0* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/missiondives/1/output_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:50:01 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 3028
< Content-Type: application/json
<
{ [data not shown]
100 3028 100 3028    0     0 1287k      0 --:--:-- --:--:-- --:--:-- 1478k
* Closing connection #0
{"output_segments": [{"tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79,
```

not found

ex 1:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/output_segments/
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/output_segments/ HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:54:52 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

## 5.12 /api/v1/mission/dives/<dive\_id>/output\_segments/<segment\_id>/

Output segments are the result of the calculations, so they are read-only

### 5.12.1 method: GET

return a specific output segment

ex:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/output_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
* connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1/output_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Date: Mon, 01 Oct 2012 21:51:27 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 428
< Content-Type: application/json
<
* Closing connection #0
{"tank": {"tank_rule": "50b", "max_ppo2": 1.6, "in_use": true, "f_n2": 0.79, "tank_vol": 15.0, "f_
```



## errors

### not found

If the given `dive_id` or if the `segment_id` is not found, the API will return a simple 404:

ex 1:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/42/output_segments/1
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/42/output_segments/1 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:55:16 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: dive_id (42) not found"}
```

ex 2:

```
$ curl -v -X GET -H "Content-type: application/json" http://127.0.0.1:8080/api/v1/mission/dives/1/output_segments/42
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1...
*   connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /api/v1/mission/dives/1/output_segments/42 HTTP/1.1
> User-Agent: curl/7.27.0
> Host: 127.0.0.1:8080
> Accept: */*
> Content-type: application/json
>
* HTTP 1.0, assume close after body
< HTTP/1.0 404 Not Found
< Date: Mon, 01 Oct 2012 21:55:45 GMT
< Server: WSGIServer/0.1 Python/2.7.3
< Content-Length: 42
< Content-Type: application/json
<
* Closing connection #0
{"message": "404: segment (42) not found"}
```

## **5.13 /api/v1/settings/**

**5.13.1 method: GET**

## **5.14 /api/v1/settings/EACHSETTINGSATTRIBUTE/**

**5.14.1 method: GET**

**5.14.2 method: PATCH**

# SOURCE CODE DOCUMENTATION

## 6.1 mission

Mission class module

A Mission is basically a list of repetitive dives. It could also be called a 'DiveTrip' for example

```
class dipplanner.mission.Mission (dive_or_divelist=None,    tank_or_tankdict=None,    description=None)
```

Mission Class

a Mission represents all repetitive dives for a given set of time. the Mission will keep consistency between repetitive dives

An isolated dive is a Mission with only one dive inside

**Attributes:**

**tanks** (dict) – list of each individual tank used (at least once) during the mission.

**dives** (list) – list of Dive object of the same mission

**description** (str) – description of the mission (OPTIONAL)

**status** (str) – actual status of the mission. Could be either:

- STATUS\_NONE: not calculated
- STATUS\_CHANGED: **changed (calculated, but something in input or parameter has changed and recalculation is needed)**
- STATUS\_OK: calculated (calculation is up to date)

---

**Todo**

Insert dive(s) in a certain position

---

**Todo**

change dive order, permutations, etc...

---

**Todo**

---

decide if we define a common list of tanks for a given mission in this case, each dive will be able to pickup some tank

---

## Todo

decide if settings is included in Mission object or remains global

---

`__init__ (dive_or_divelist=None, tank_or_tankdict=None, description=None)`

Mission constructor

Initialize the Mission object if no parameter is given, instantiate an 'empty' Mission

### **Keyword Arguments:**

**dive\_or\_divelist** – either a Dive object or a list of Dive object

**tank\_or\_tankdict** – either a Tank object or a dict of tanks

**description** (str)– description of the Mission

**Return:** <nothing>

**Raise:** TypeError: if dive\_or\_divelist contains another type than Dive

`__iter__ ()`

iterable

see `dipplanner.mission.Mission._forward()`

`__forward ()`

forward generator, used for iteration

`__len__ ()`

calculate and return len of Session object

usage example:

```
print(len(my_mission))
```

**Args:** <none>

### **Returns:**

**int** number of Dives in this mission

`dumps_dict ()`

dumps the Mission object in json format

**Keyword arguments:** <none>

**Returns:** dict – json dumps of Tank object

**Raise:** TypeError : if Mission is not serialisable

`loads_json (input_json)`

loads a json structure and update the mission object with the new values.

This method can be used in http PUT method to update object value

### **Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable

**clean** (*what='all'*)  
clean the mission

**Keyword arguments:**

**what** (str) – what the method should clean by default: all (clean all) Values allowed: ['all', 'dives', 'description', ]

**Returns:** <none>

**Raise:**

- ValueError : if wrong status code is given

**change\_status** (*status=None*)  
Change the status of the mission

if no status is given in args, toggle the status from OK to CHANGED else, update the status with the given value

**Keyword arguments:**

**status** (str) – status code (optionnal)

**Returns:** <none>

**Raise:**

- ValueError : if wrong status code is given

**add\_dive** (*dive\_or\_divelist*)  
add a Dive or a list of dive to the Mission  
they will be added at the end of the list

**Keyword Arguments:**

**dive\_or\_divelist** – either a Dive object or a list of Dive objects

**Return:** <nothing>

**Raise:** TypeError: if dive\_or\_divelist contains another type than Dive

**add\_tank** (*tank\_or\_tankdict*)  
add a Tank or a dict of tanks to the Mission  
if a tank already exists in the dict with the same name, the new one will not be added.

**Keyword Arguments:**

**tank\_or\_tanklist** – either a Tank object or a list of Tank objects

**Return:** <nothing>

**Raise:** TypeError: if tank\_or\_tanklist contains another type than Tank

**calculate** ()  
Calculate all the decompression planning for all dives in this mission

**\_\_str\_\_**  
x.\_\_str\_\_() <==> str(x)

## 6.2 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(mission,      known_segments=None,      known_tanks=None,      previ-
                           ous_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:

- mission – (Mission) Mission object the dive belongs to
- name – (str) name of the dive
- 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
- 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\_repetitive\_dive – Flag for repetitive dives
- surface\_interval – for surf. int. in seconds
- no\_flight\_time\_value – calculated no flight time
- metadata – description for the dive

```
__init__(mission, known_segments=None, known_tanks=None, previous_profile=None)
```

Constructor for Profile class

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

**Keyword Arguments:**

- mission** (Mission) – mission object the dive belongs to
- 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 is 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>

**add\_input\_segment** (segment\_or\_segmentlist)

add a Segment or a list of Segments to the Dive's input\_segments list. They will be added at the end of the list

**Keyword Arguments:**

**segment\_or\_segmentlist** – either a Segment object or a list of Segment objects

**Return:** <nothing>

**Raise:**

**TypeError:** if segment\_or\_segmentlist contains another type than Segment\*

**dumps\_dict** ()

dumps the Dive object in dict format for later json conversion

**Keyword arguments:** <none>

**Returns:** string – dict dumps of Dive object

**Raise:** TypeError : if Mission is not serialisable

**loads\_json** (input\_json)

loads a json structure and update the tank object with the new values.

This method can be used in http PUT method to update object value

TODO: do something with automatic\_tank\_refill: global or local (dive) parameter ?

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable

**set\_repetitive** (*previous\_dive*)

Make this dive a repetitive dive by cloning the previous model history into the current dive

**Keyword Arguments:**

**previous\_dive** (Dive) – previous dive profile

**Return:** <nothing>

**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=None*)

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 if not provided, uses the surface\_interval defined in this dive

**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 occurred

**Keyword Arguments:**

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

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

**Raise:** <Exceptions from tank>

## 6.3 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:**

- dive (Dive) – Dive object the segment belongs to
- 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 (displayed in profile informations)
- 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>

**dumps\_dict** ()

dumps the Segment object in json format

**Keyword arguments:** <none>

**Returns:** string – json dumps of Segment object

**Raise:** TypeError : if Tank is not serialisable

**loads\_json** (input\_json)

loads a json structure and update the segment object with the new values.

This method can be used in http PUT method to update object value

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable
- **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

**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=None, time=None, tank=None, setpoint=0)

Specialisation of segment class for dive segments

**\_\_init\_\_** (depth=None, time=None, tank=None, 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

**dumps\_dict()**  
dumps the Segment object in json format  
*Keyword arguments:* <none>  
*Returns:* string – json dumps of Segment object  
*Raise:* TypeError : if Tank is not serialisable

**get\_end()**  
Calculates and returns E.N.D : Equivalent Narcosis Depth



Instead of Mvplan, it uses a 'calculation method' based on narcosis effect 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, which 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 pressure 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>

**loads\_json** (*input\_json*)

loads a json structure and update the segment object with the new values.

This method can be used in http PUT method to update object value

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable
- **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

**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

**dumps\_dict** ()

dumps the Segment object in json format

**Keyword arguments:** <none>

**Returns:** string – json dumps of Segment object

**Raise:** TypeError : if Tank is not serialisable

**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>

**loads\_json** (*input\_json*)

loads a json structure and update the segment object with the new values.

This method can be used in http PUT method to update object value

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- **ValueError** : if json is not loadable
- **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

**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 comparison 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>

**dumps\_dict()**

dumps the Segment object in json format

**Keyword arguments:** <none>

**Returns:** string – json dumps of Segment object

**Raise:** TypeError : if Tank is not serialisable

**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>

**loads\_json(input\_json)**  
loads a json structure and update the segment object with the new values.

This method can be used in http PUT method to update object value

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable
- **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

## 6.4 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, volume=12.0, pressure=200, rule='30b', name=None*)

This class implements a representation of dive tanks which 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
- pressure
- remaining gas warning rule

---

**Note:** About imperial unit conversion

In ‘imperial’ countries (North America), it’s common to describe a tank with the volume of air stored in the cylinder at its working pressure (if you were to release it at the surface), instead of (internal volume \* pressure).

This makes it difficult if not impossible to switch between the two units without any approximation or implementation choices.

eg: 80-cubic-foot aluminium cylinder (AL80)

TODO: continuer le texte explicatif



*Attributes:*

- **f\_o2** (float) – fraction of oxygen in the gas in % ( $\geq 0.0$  &  $\leq 1.0$ )
- **f\_he** (float) – fraction of helium in the gas in % ( $\geq 0.0$  &  $\leq 1.0$ )
- **f\_n2** (float) – **fraction of nitrogen in the gas in %** ( $\geq 0.0$  &  $\leq 1.0$ )
- **max\_ppo2** (float) – maximum tolerated ppo2 for this tank
- **volume** (float) – volume of tank in liter
- **pressure** (float) – pressure of tank in bar
- **rule** (str) – tank rule for minimum gas calculation
- **mod** (float) – maximum operating depth of the tank
- **in\_use** (boolean) – is the tank used for the dive or 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
- **name** (str) – **name of the Tank if not provided**, try to set it automatically

\_\_init\_\_ (f\_o2=0.21, f\_he=0.0, max\_ppo2=1.6, mod=None, volume=12.0, pressure=200, rule='30b',  
name=None)  
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 calculated based on max tolerable ppo2

*Keyword arguments:*

- f\_o2** (float) – Fraction of O2 in the gas in % value between 0.0 and 1.0
- f\_he** (float) – Fraction of He in the gas 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
- volume** (float) – Volume of the tank in liter
- pressure** (float) – Pressure of the tank, in bar
- rule** – rule for warning in the tank consumption must be either : 'xxxb' or '1/x'.

---

**Note:** xxxb (ex: 50b) means 50 bar minimum at the end of the dive

---

---

**Note:** 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 way OUT, 2/3 remains

---

**name** (str) – set a specific name for the Tank. if not given, it will be generated automatically based on the gas.

**Returns:** <nothing>

**Raise:**

- InvalidGas – see validate()
- InvalidMod – if mod > max mod based on max\_ppo2 and see validate()
- InvalidTank – see validate()

**\_\_set\_min\_gas** ()

sets the minimum gas volume for this tank based on the min\_gas rule provided

**Keyword Arguments:** <none>

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

**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:** Tank – Tank object copy of itself

**Raise:** <nothing>

**calculate\_real\_volume** (volume=None, 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:**

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

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

**dumps\_dict** ()

dumps the Tank object in json format

**Keyword arguments:** <none>

**Returns:** string – json dumps of Tank object

**Raise:** TypeError : if Tank is not serialisable

**loads\_json** (*input\_json*)

loads a json structure and update the tank object with the new values.

This method can be used in http PUT method to update object value

**Keyword arguments:**

**input\_json** (string) – the json structure to be loaded

**Returns:** <none>

**Raise:**

- ValueError : if json is not loadable
- **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

**\_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

**automatic\_name** ( )

returns a Human readable name for the gaz and tanks Different 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 depth in 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 be 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 be 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 against 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>

## 6.5 model: buhlmann

### 6.5.1 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>

## 6.5.2 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 initiate 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 instantaneous equation:  $P = P_o + (P_i - P_o)(1 - e^{-kt})$  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 :  $P = P_{io} + R(t - 1/k) - [P_{io} - P_o - (R/k)]e^{-kt}$  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  $P_m = P_a/b + a$  where:

- $P_m$  = M-Value pressure,
- $P_a$  = ambient 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) – ambient pressure

**Return:** float – M-value

**Raise:** <nothing>

### 6.5.3 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* or *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)

## 6.5.4 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](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)

### 6.5.5 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
- deco\_model (str) – name of the deco model
- 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

## 6.6 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>

## 6.7 tools

tools modules

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

`dipplanner.tools.safe_eval_calculator(text_to_eval)`  
Small an safe eval function usable only for simple calculation (only the basic operators)

**Keyword Arguments:**  
**text\_to\_eval** (`str`) – the text (formula) that should be evaluated

**Returns:** `float` – the result of the calculation.

**Raise:**  
**ValueError:** when given expression is not constituted of only numbers and operators  
**SyntaxError:** when the given expression is incorrect

`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:  $p = 101325 \cdot (1 - 2.25577 \cdot 10^{-5} \cdot h)^{5.25588}$

**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 x10

**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](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>

## 6.8 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

---

### Todo

`logger.tutu(“message %s”, allo)` au lieu de `logger.tutu(“message %s” % allo)`

---

```
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.main(cli_arguments=['/home/docs/checkouts/readthedocs.org/user_builds/dipplanner/envs/pyconfr/bin/sphinx-build', '-b', 'latex', '-d', '_build/doctrees', '.', '_build/latex'])
```

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:** arguments (list of string) – list of arguments, like sys.argv

**Return:** <nothing>

**Raise:** <nothing>

## 6.9 settings

Global settings for dipplanner and 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
```

## 6.10 gui

dipplanner GUI module.

Starts the GUI

---

### Todo

uniquely name the tanks (in addition of tank\_id (int) AND handle Tank references (from tank list) to: current\_tank and segment.tank

Actually: each tank is separated

*WORK IN PROGRESS* (reste en cours: les dumps\_dict a modifier)

---

```
dipplanner.gui.instanciates_app (mission=None)  
defines all the routes and others parameters for the app
```

```
dipplanner.gui.start_gui (mission=None, http_host='localhost', http_port=8080)  
Starts the html GUI server
```

### 6.10.1 REST Api base object

```
class dipplanner.gui.rest_main_api.ApiBottle  
Main API for bottle
```

### 6.10.2 REST Api for Mission object

```
class dipplanner.gui.rest_mission.MissionApiBottle (mission=None)
```

At this time, there is only one Mission at a time for a dipplanner session (if the user need to work on another mission, he MUST close the current Mission first).

The API reflects this current limitation

**get ()**

GET method for the Mission object Api

returns a json dumps of the mission object.

**Keyword Arguments:** <none>

**Returns:** resp – response object with the json dump of the mission object

**Raise:** <nothing>

Sample usage:

```
curl -H "Content-Type: application/json" http://localhost:8080/api/v1/mission/
```

**get\_status ()**

returns the actual status of the mission

returns a json object with the current status of the Mission object.

ex:

```
{ 'status': 'Calculated and Up to date' }
```

**Keyword Arguments:** <none>

**Returns:** resp – response object with the status

**Raise:** <nothing>

**post ()**

POST method for the Mission object Api

create a new mission an new mission may only be created if the existing mission is empty (api user MUST call DELETE before calling POST)

if a json structure is POSTed with the request, dipplanner will try to load this stucture inside the mission object. A full structure may be given.

**Keyword Arguments:** <nothing>

**Returns:**

**resp – response object with the json dump of the newly created object**

**Raise:** <nothing>

**patch ()**

PATCH method for the Mission object Api

update the mission object

**Keyword Arguments:** <none>

**Returns:** resp – response object - HTTP 200 + the patched mission object

**Raise:** <nothing>

**calculate ()**

launch the calculation of the mission

returns a json object with the current status of the Mission object (after calculation).

ex:

```
{ 'status': 'Calculated and Up to date' }
```

**Keyword Arguments:** <none>

**Returns:** resp – response object with the status

**Raise:** <nothing>

**delete** (*resource\_id=None*)

DELETE method for the Mission object Api

**Keyword Arguments:** <none>

**Returns:** resp – response object (empty)

**Raise:** <nothing>

### 6.10.3 REST Api for Dive object

**class** dipplanner.gui.rest\_dive.**DiveApiBottle** (*mission=None*)

api for dives inside the mission object

**get** (*resource\_id=None*)

GET method for the Dive object Api

returns a json dumps of the dives in the current mission object.

**Keyword Arguments:**

**resource\_id** (str) – either a number representing the position of the dive (first dive is dive 1) or the name of the dive

**Returns:** resp – response object with the json dump of the dives

**Raise:** <nothing>

**post** ()

POST method for the Dive object Api

create a new dive for this mission

if a json structure is POSTed with the request, dipplanner will try to load this structure while instanciating the new dive A full dive structure may be given, but a partial structure will also be allowed. (all non given parameters will use the default values)

**Keyword Arguments:** <nothing>

**Returns:**

**resp – response object with the json dump of the newly created object**

**Raise:** <nothing>

**patch** (*resource\_id=None*)

PATCH method for the Dive object Api

update the dive object

if no resource\_id is given, returns 404 if resource\_id is given, try to patch the resource and returns the entire patched Dive with code 200 OK

**Keyword Arguments:**

**resource\_id** –str : number of the dive, starting by 1

**Returns:**

**resp** – response object - HTTP 200 + the list of remaining dives after the deletion

**Raise:** <nothing>

**delete** (*resource\_id=None*)

DELETE method for the Dive object Api

if no resource\_id is given, all the dives will be deleted. If resource\_id is given and exists, only one dive will be deleted

**Keyword Arguments:**

**resource\_id** –str : number of the dive, starting by 1

**Returns:**

**resp** – response object - HTTP 200 + the list of remaining dives after the deletion

**Raise:** <nothing>

## 6.10.4 REST Api for Tank object

**class** dipplanner.gui.rest\_tank.**TankApiBottle** (*mission=None*)

api for dives inside the mission object

**get** (*tank\_id=None*)

GET method for the Tank object Api

returns a json dumps of the tanks in the current mission object.

**Keyword Arguments:**

**resource\_id** (int) – number of the dive, first dive is dive 1

**Returns:** resp – response object with the json dump of the dives

**Raise:** <nothing>

**post** ()

POST method for the Tank object Api

create a new tank for this dive

if a json structure is POSTed with the request, dipplanner will try to load this structure while instanciating the new tank A full Tank structure may be given, but a partial structure will also be allowed. (all non given parameters will use the default values)

**Keyword Arguments:** <nothing>

**Returns:**

**resp** – response object with the json dump of the newly created object

**Raise:** <nothing>

**patch** (*tank\_id=None*)

PATCH method for the Tank object Api

update the tank object

if no tank\_id is given, returns 404 if tank\_id is given, try to patch the resource and returns the entire patched Tank with code 200 OK

**Keyword Arguments:**



**dive\_id** – (str): number of the tank, starting by 1

**tank\_id** –str : number of the dive, starting by 1

**Returns:**

**resp** – response object - HTTP 200 + the list of remaining dives after the deletion

**Raise:** <nothing>

**delete** (*tank\_id=None*)

DELETE method for the Tank object Api

if no resource\_id is given, all the tanks will be deleted. If resource\_id is given and exists, only one tank will be deleted

**Keyword Arguments:**

**tank\_id** –str : number of the tank, starting by 1

**Returns:**

**resp** – response object - HTTP 200 + the list of remaining tanks after the deletion

**Raise:** <nothing>

## 6.10.5 bottle HTTP error handling

**class** dipplanner.gui.error\_api.**ErrorApiBottle**

Error API for bottle

**error404** (*code*)

Handle 404 error and returns a json structure instead of classic html structure

**error405** (*code*)

Handle 405 error (method not allowed) and returns a json structure instead of classic html structure



# RELEASE NOTES

## 7.1 v0.3 (ongoing)

---

**Note:** v0.3 is not released yet.

Below are the feature already developed for the 0.3 target release.

See GitHub Issues for more infos.

---

### 7.1.1 New features

- global definition of Tanks in mission, Dives referring to mission's tanks
- new Mission object wich represent a set of repetitive dives
- No-flight time calculation
- non-blocking dive situations errors

### 7.1.2 Bug corrections

- tank infos on repetitive dives was wrong (only last tank status was diven, even for the firs(s) dive(s)).

## 7.2 v0.2

### 7.2.1 New features

- New submodel: buhlmann ZH-L16C
- Handle repetitive dives
- Variable pPH<sub>2</sub>O 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

## 7.2.2 Bug corrections

- Automatic gas selection was not well handled on Hypoxic trimix dives

## 7.3 v0.1

### 7.3.1 New features

- **Same ‘base’ fonctionnality 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’

# TODOS

---

## Todo

Insert dive(s) in a certain position

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/diplanner/envs/pyconfr/local/lib/python2.7/site-packages/diplanner-0.3a\_40\_g1237-py2.7.egg/diplanner/mission.py:docstring of diplanner.mission.Mission, line 20.)

---

## Todo

change dive order, permutations, etc...

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/diplanner/envs/pyconfr/local/lib/python2.7/site-packages/diplanner-0.3a\_40\_g1237-py2.7.egg/diplanner/mission.py:docstring of diplanner.mission.Mission, line 22.)

---

## Todo

decide if we define a common list of tanks for a given mission in this case, each dive will be able to pickup some tank

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/diplanner/envs/pyconfr/local/lib/python2.7/site-packages/diplanner-0.3a\_40\_g1237-py2.7.egg/diplanner/mission.py:docstring of diplanner.mission.Mission, line 24.)

---

## Todo

decide if settings is included in Mission object or remains global

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/diplanner/envs/pyconfr/local/lib/python2.7/site-packages/diplanner-0.3a\_40\_g1237-py2.7.egg/diplanner/mission.py:docstring of diplanner.mission.Mission, line 27.)

---

## Todo

logger.tutu("message %s", allo) au lieu de logger.tutu("message %s" % allo)

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/diplanner/envs/pyconfr/local/lib/python2.7/site-packages/diplanner-0.3a\_40\_g1237-py2.7.egg/diplanner/main.py:docstring of diplanner.main, line 9.)

---

### Todo

uniquely name the tanks (in addition of tank\_id (int) AND handle Tank references (from tank list) to: current\_tank and segment.tank

Actually: each tank is separated

*WORK IN PROGRESS* (reste en cours: les dumps\_dict a modifier)

---

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user\_builds/dipplanner/envs/pyconfr/local/lib/python2.7/site-packages/dipplanner-0.3a\_40\_g1237-py2.7.egg/dipplanner/gui/\_\_init\_\_.py:docstring of dipplanner.gui, line 5.)

---

### Todo

check the usage of the multilevel\_mode option in config-files

---

(The *original entry* is located in /var/build/user\_builds/dipplanner/checkouts/pyconfr/doc/config\_file.rst, line 641.)

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*





# PYTHON MODULE INDEX

## d

- `dipplanner.dipp_exception`, 107
- `dipplanner.dive`, 74
- `dipplanner.gui`, 113
- `dipplanner.gui.error_api`, 117
- `dipplanner.gui.rest_dive`, 115
- `dipplanner.gui.rest_main_api`, 113
- `dipplanner.gui.rest_mission`, 113
- `dipplanner.gui.rest_tank`, 116
- `dipplanner.main`, 110
- `dipplanner.mission`, 71
- `dipplanner.model.buhlmann.compartment`,  
99
- `dipplanner.model.buhlmann.gradient`, 102
- `dipplanner.model.buhlmann.model`, 104
- `dipplanner.model.buhlmann.model_exceptions`,  
98
- `dipplanner.model.buhlmann.oxygen_toxicity`,  
103
- `dipplanner.segment`, 80
- `dipplanner.settings`, 111
- `dipplanner.tank`, 90
- `dipplanner.tools`, 108