IL-2 FB Mission Parser Documentation

Release 1.1.0.dev3

Alexander Oblovatniy, Alexander Kamyhin

Dec 06, 2017

Contents

Documentation

Note: This is project's documentation in English. Visit Wiki to get information in Russian.

. .

`il2fb-mission-parser` — is a free Python library for parsing mission files of IL-2 FB aviasimulator. It helps you to convert a text file in a tricky format into a pretty Python object.

Contents:

CHAPTER 1

Installation

Note: Russian version

This library is as easy to install as any other Python library. The common way is to use PyPI:

```
$ pip install il2fb-mission-parser
```

CHAPTER 2

Usage

Note: Russian version

The main purpose of this library is to parse a whole mission file.

2.1 Parse by file name

The most common use-case is to give path to a mission file and get a parsed result:

```
>>> from il2fb.parsers.mission import MissionParser
>>> parser = MissionParser()
>>> mission = parser.parse("path/to/your/mission.mis")
```

This will put a big dictionary into a `mission` variable. That's it. You do not need to do something else.

2.2 Parse sequence of lines

`parse_mission` function can accept not only a path to a file, but any object which can generate a sequence of lines: a text file, list, generator and so on. For example:

```
>>> with open("path/to/your/mission.mis") as f:
...     mission = parser.parse(f)
```

Or:

```
>>> lines = [
...     "[Wing]",
...     "  r0100",
...     "[r0100]",
...     "  Planes 1",
```

```
...     "  Skill 1",
...     "  Class air.A_20C",
...     "  Fuel 100",
...     "  weapons default",
... ]
>>> mission = parser.parse(lines)
```

2.3 Dealing with result

Since the output dictionary can be really big and complex, it's recommended to use t_dict, aadict or SuperDict library to make access to elements of result easier.

You can go forward to *description of output format* to get to know what is contained inside `mission` or you can continue reading this chapter.

2.4 Behind the scene

Let's talk about what's going on above. This library provides a Python module called `il2fb.parsers.mission.sections` which has a lot of parsers for each kind of section in mission files (see *all of them*).

`MissionParser` is just like a swiss-knife and combines all of the other parsers in itself, processes the whole mission file and gives all you need at one time.

You can use any other parser separately for your needs also (see below).

2.5 Manual section parsing

Each parser listed in *Section parsing* extends an abstract class `SectionParser`, so they share a common approach for section processing.

Note: Since these parsers were designed to be used by the `MissionParser`, which is a one-pass parser, they can parse only one line at a time. It's just a side-effect that you can use them for your needs.

If you really need to parse some section, you need to prepare string lines and tell parser the name of section. E.g.:

```
>>> lines = [
...     "MAP Moscow/sload.ini",
...     "TIME 11.75",
...     "TIMECONSTANT 1",
...     "WEAPONSCONSTANT 1",
...     "CloudType 1",
...     "CloudHeight 1500.0",
...     "player fiLLv24fi00",
...     "army 1",
...     "playerNum 0",
... ]
>>> from il2fb.parsers.mission.sections.main import MainSectionParser
>>> p = MainSectionParser()
>>> p.start('MAIN')
True
```

```
>>> for line in lines:
...     p.parse_line(line)
...
>>> p.stop()
{
    'location_loader': 'Moscow/sload.ini',
    'time': {
        'is_fixed': True,
        'value': datetime.time(11, 45),
    },
    'cloud_base': 1500,
    'weather_conditions': <constant 'Conditions.good'>,
    'player': {
        'aircraft_index': 0,
        'belligerent': <constant 'Belligerents.red'>,
        'fixed_weapons': True,
        'flight_id': 'fiLLv24fi00',
    },
}
```

As you can see, you need to import a desired parser and create it's instance.

Then you need to `start()` parser and provide a name of section you are going to parse. Method will return `True` if parser can handle sections with the given name or `False` otherwise.

Note: section names can contain prefixes and suffixes such as `0_*` or `*_0`. They can have dynamic values and they can be used as a part of output result, so we cannot make strict mapping of section names to parsers. That's why each parser checks whether it can handle sections with a given name.

Now it's a time to feed the parser with some data. As it was mentioned above, you can pass only one line at a time to `parse_line()` method. You can do it in any suitable manner.

When you have passed all the data, call `stop()` method to stop parsing. This method will return fully-parsed data which is a dictionary in general.

CHAPTER 3

Mission parser

Note: Russian version

`MissionParser` is responsible for parsing whole mission files or sequences of strings which look like mission definition.

This parser detects sections in the stream of strings, selects a proper parser for a certain section and combines results from all parsers into a single whole. The output of this parser is a `dict`.

Since many sections are optional (e.g., a list of moving ground units, their routes, a list of available aircrafts at airfields, etc.) and some sections may not be available in previous versions of the game (e.g., `MDS`), so we cannot talk about a clear and predefined structure of parser's result. To understand what may be in the output, it will be much easier and clearer to *use project's demo*.

Here we will go through the very principles on which the final result is formed. It *may* contain the following elements:

- *location_loader*
- *player*
- *targets*
- *conditions*
  - *time_info*
  - *meteorology*
  - *scouting*
  - *respawn_time*
  - *radar*
  - *communication*

- *home_bases*
- *crater_visibility_muptipliers*

- *objects*
  - *moving_units*
  - *flights*
  - *home_bases*
  - *stationary*
  - *buildings*
  - *cameras*
  - *markers*
  - *rockets*

Example of parser result:

```
{
   'location_loader': 'Slovakia/load_online.ini',
   'player': {
      'aircraft_index': 0,
      'belligerent': Belligerents.red,
      'fixed_weapons': False,
      'flight_id': None,
   },
   'targets': [
      {
         'type': TargetTypes.recon,
         'priority': TargetPriorities.secondary,
         'in_sleep_mode': True,
         'delay': 50,
         'requires_landing': False,
         'pos': Point2D(133978.0, 87574.0),
         'radius': 1150,
      },
      {
         'type': TargetTypes.recon,
         'priority': TargetPriorities.primary,
         'in_sleep_mode': True,
         'delay': 40,
         'requires_landing': True,
         'pos': Point2D(134459.0, 85239.0),
         'radius': 300,
         'object': {
             'waypoint': 0,
             'id': '1_Chief',
             'pos': Point2D(134360.0, 85346.0),
         },
      },
   ],
   'conditions': {
      'time_info': {
         'date': datetime.date(1945, 4, 5),
         'time': datetime.time(10, 0),
         'is_fixed': False,
```

```
    },
    'meteorology': {
        'cloud_base': 1300,
        'gust': Gust.none,
        'turbulence': Turbulence.none,
        'weather': Conditions.hazy,
        'wind': {
            'direction': 180.0,
            'speed': 2.0,
        },
    },
    'communication': {
        'ai_radio_silence': False,
        'tower_communication': True,
        'vectoring': True,
    },
    'scouting': {
        'only_scouts_complete_targets': False,
        'scouts_affect_radar': False,
        'ships_affect_radar': False,
    },
    'respawn_time': {
        'artillery': 1000000,
        'balloons': 1000000,
        'searchlights': 1000000,
        'ships': {
            'big': 1000000,
            'small': 1000000,
        },
    },
    'radar': {
        'advanced_mode': False,
        'refresh_interval': 0,
        'scouts': {
            'alpha': 5,
            'max_height': 1500,
            'max_range': 2,
        },
        'ships': {
            'big': {
                'max_height': 5000,
                'max_range': 100,
                'min_height': 100,
            },
            'small': {
                'max_height': 2000,
                'max_range': 25,
                'min_height': 0,
            },
        },
    },
    'home_bases': {
        'hide_ai_aircrafts_after_landing': False,
        'hide_players_count': False,
        'hide_unpopulated': True,
    },
    'crater_visibility_muptipliers': {
        'gt_1000kg': 1.0,
```

```
        'le_1000kg': 1.0,
        'le_100kg': 1.0,
    },
},
'objects': {
    'moving_units': [
        {
            'id': '0_Chief',
            'type': UnitTypes.train,
            'code': 'Germany_CargoTrainA/AA',
            'belligerent': Belligerents.blue,
            'route': [
                GroundRoutePoint(
                    pos=Point2D(21380.02, 41700.34),
                    is_checkpoint=True,
                    delay=10,
                    section_length=2,
                    speed=11.0,
                ),
                GroundRoutePoint(
                    pos=Point2D(21500.00, 41700.00),
                    is_checkpoint=False,
                ),
            ],
        },
    ],
    'flights': [
        {
            'ai_only': False,
            'air_force': AirForces.luftwaffe,
            'aircrafts': [
                {
                    'index': 0,
                    'has_markings': True,
                    'skill': Skills.ace,
                },
            ],
            'code': 'Do217_K2',
            'count': 1,
            'flight_index': 0,
            'fuel': 100,
            'id': 'g0100',
            'regiment': None,
            'squadron_index': 0,
            'weapons': 'default',
            'with_parachutes': True,
            'route': [
                FlightRouteTakeoffPoint(
                    type=RoutePointTypes.takeoff_normal,
                    pos=Point3D(193373.53, 99288.17, 0.0),
                    speed=0.0,
                    formation=None,
                    radio_silence=False,
                    delay=10,
                    spacing=20,
                ),
                FlightRoutePoint(
                    type=RoutePointTypes.landing_straight,
```

```
                pos=Point3D(185304.27, 54570.12, 0.00),
                speed=0.00,
                formation=None,
                radio_silence=True,
            ),
        ],
    },
],
'home_bases': [
    {
        'belligerent': Belligerents.red,
        'friction': {
            'enabled': False,
            'value': 3.8,
        },
        'pos': Point2D(151796.0, 71045.0),
        'radar': {
            'max_height': 5000,
            'min_height': 0,
            'range': 50,
        },
        'range': 3000,
        'show_default_icon': False,
        'spawning': {
            'aircraft_limitations': {
                'allowed_aircrafts': [
                    {
                        'code': 'Il-2_3',
                        'limit': None,
                        'weapon_limitations': [
                            '4xRS82',
                            '4xBRS82',
                            '4xRS132',
                        ]
                    },
                    {
                        'code': 'Il-2_M3',
                        'limit': None,
                        'weapon_limitations': [
                            '4xBRS132',
                            '4xM13',
                            '216xAJ-2',
                        ],
                    },
                ],
                'consider_lost': True,
                'consider_stationary': True,
                'enabled': True,
            },
            'allowed_air_forces': [
                AirForces.vvs_rkka,
            ],
            'enabled': True,
            'in_air': {
                'conditions': {
                    'always': False,
                    'if_deck_is_full': False,
                },
```

```
                'heading': 0,
                'height': 1000,
                'speed': 200,
            },
            'in_stationary': {
                'enabled': False,
                'return_to_start_position': False,
            },
          'max_pilots': 0,
          'with_parachutes': True,
        },
    },
],
'stationary': [
    StationaryObject(
        belligerent=Belligerents.none,
        id='6_Static',
        code='Smoke20',
        pos=Point2D(151404.61, 89009.57),
        rotation_angle=0.00,
        type=UnitTypes.stationary,
    ),
],
'buildings': [
    Building(
        id='0_bld',
        belligerent=Belligerents.red,
        code='Tent_Pyramid_US',
        pos=Point2D(43471.34, 57962.08),
        rotation_angle=270.00,
    ),
],
'cameras': [
    StaticCamera(
        belligerent=Belligerents.blue,
        pos=Point3D(38426.0, 65212.0, 35.0),
    ),
],
'markers': [
    FrontMarker(
        id='FrontMarker0',
        belligerent=Belligerents.red,
        pos=Point2D(7636.65, 94683.02),
    ),
],
'rockets': [
    Rocket(
        id='0_Rocket',
        code='Fi103_V1_ramp',
        belligerent=Belligerents.blue,
        pos=Point2D(84141.38, 114216.82),
        rotation_angle=0.00,
        delay=60.0,
        count=10,
        period=80.0,
        destination=Point2D(83433.91, 115445.49),
    ),
],
```

```
    },
}
```

3.1 location_loader

Contains name of location loader which is defined in *MAIN section*. Usually this element is always present.

3.2 player

Contains a `dict` with information about player which is defined in *MAIN section*. Usually this element is always present.

3.3 targets

Contains a list of targets which are defined in *Target section*.

3.4 conditions

Contains a `dict` with information about different conditions in mission:

3.4.1 time_info

A `dict` with information about date and time from *MAIN section* and *SEASON section*.

3.4.2 meteorology

A `dict` with information about meteorology from *MAIN section* and *WEATHER section*.

3.4.3 scouting

A `dict` with information about scouting from *MDS section*. Can also contain lists of scouts separately per each belligerent (see *MDS_Scouts section*).

3.4.4 respawn_time

Contains result of parsing *RespawnTime section*.

3.4.5 radar

Contains common settings for radars from *MDS section*.

3.4.6 communication

Contains common communication settings from *MDS section*.

3.4.7 home_bases

Contains common settings for home bases from *MDS section*.

3.4.8 crater_visibility_muptipliers

Contains settings for craters visibility from *MDS section*.

3.5 objects

A `dict` which contains lists of objects defined in mission:

3.5.1 moving_units

List of moving ground units which is defined in *Chiefs section*. Each unit also contains own route which is defined in *Chief Road section*.

3.5.2 flights

List of AI flights. Information is taken from *Flight info sections* which are listed in *Wing section*. Each flight also contains own route which is defined in *Flight route section*.

3.5.3 home_bases

List of airfields which are defined in *BornPlace sections*. Airfields also may contain information about allowed air forces from *BornPlace air forces sections* and information about allowed aircrafts from *BornPlace aircrafts sections*.

3.5.4 stationary

List of stationary objects defined in *NStationary section*.

3.5.5 buildings

List of buildings defined in *Buildings section*.

3.5.6 cameras

List of stationary cameras defined in *StaticCamera section*.

3.5.7 markers

List of frontline markers defined in *FrontMarker section*.

3.5.8 rockets

List of rockets defined in *Rocket section*.

CHAPTER 4

Comments in missions

Note: Russian version

As it is known, many creators of missions put some comments and notes for themselves directly inside mission file.

`il2fb-mission-parser` treats as comments everything that stands to the right from the following delimiters including delimiters themselves:

1. `;`
2. `#`
3. `//`
4. `--`

Example:

```
[Target]
  1 2 0 0 750 19750 4275 500 ;0
  1 2 0 0 750 21096 14030 500 #1
  1 2 0 0 750 21971 19014 500 //2
  1 2 0 0 750 17744 27538 500 --3
```

Comment blocks are not supported.

CHAPTER 5

Section parsing

Note: Russian version

This chapter describes output formats for section parsers. Datailed description of input data aslo included.

Note: Format of mission files can be very tricky in some sections and original key names may be misleading. We made our best to adopt strange things for normal humans, but some questions still may appear in your mind. Get ready!

Note: If you are not familiar with missions, take a look at some of them.

5.1 MAIN section

Note: Russian version

`MainSectionParser` is responsible for parsing `MAIN` section. This section contains one key-value pair per each line.

Section example:

```
[MAIN]
  MAP Moscow/sload.ini
  TIME 11.75
  TIMECONSTANT 1
  WEAPONSCONSTANT 1
  CloudType 1
  CloudHeight 1500.0
```

```
  player fiLLv24fi00
  army 1
  playerNum 0
```

Output example:

```
{
    'location_loader': 'Moscow/sload.ini',
    'time': {
        'value': datetime.time(11, 45),
        'is_fixed': True,
    },
    'weather_conditions': Conditions.good,
    'cloud_base': 1500,
    'player': {
        'belligerent': Belligerents.red,
        'flight_id': "fiLLv24fi00",
        'aircraft_index': 0,
        'fixed_weapons': True,
    },
}
```

As you can see, we have a `dict` as a result.

Description:

MAP Name of location loader. Location loaders contain information about locations (textures, air pressure, air temperature, list of map labels, etc) and can be found inside `fb_maps*.SFS` archives.

Output path `location_loader`

Output type `str`

Output value original string value

TIME Initial time in mission. Defined as a real number. Integer part defines hour. Fractional part defines minutes as a fraction of 60 minutes, so `0.75` is `60 * 0.75 = 45` minutes indeed.

Output path `time.value`

Output type `datetime.time`

TIMECONSTANT Whether time specified by `TIME` must be fixed during all mission long.

Output path `time.is_fixed`

Output type `bool`

Output value `True` if `1`, `False` otherwise

WEAPONSCONSTANT Whether player's loadout is fixed (usually used in single player).

Output path `player.fixed_weapons`

Output type `bool`

Output value `True` if `1`, `False` otherwise

CloudType Describes type of weather by code in range `[0-6]`.

Output path `weather_conditions`

Output type complex weather conditions constant

CloudHeight A real number which defines cloud base.

Output path `cloud_base`

Output type `int`

Output value original value converted to integer number

`player`[1] ID of AI flight which player will be the part of during single mission or campaign mission.

Output path `player.flight_id`

Output type `str`

Output value original string value or `None` if not present

`army`[1] Code number of player's belligerent. This value is primarily used to correctly define types of targets for a particular player.

For example, this value equals to `1` and there are 2 targets defined for mission: 1) destroy an object; 2) protect objects in an area.

In this case, Allies will see these targets on map without changes.

But for the Axis these targets will be displayed with the opposite meaning, i.e.: 1) protect an object; 2) destroy objects in an area.

This principle works only if there are only 2 belligerents in mission: red and blue.

Output path `player.belligerent`

Output type complex belligerents constant

`playerNum`[1] Player's position in flight defined by `player`. It's always equal to `0` if `player` is not set.

Output path `player.aircraft_index`

Output type `int`

Output value original value converted to integer number

Footnotes:

5.2 SEASON section

Note: Russian version

`SeasonSectionParser` is responsible for parsing `SEASON` section. This section describes mission's date and contains 3 lines with key-value pairs. Each line contains year, month and day respectively.

Parser returns a dictionary with `datetime.date` object which is accessible by `date` key.

Section example:

```
[SEASON]
  Year 1942
  Month 8
  Day 25
```

Output example:

[1] For single player mode only.

```
{
    'date': datetime.date(1942, 8, 25),
}
```

5.3 WEATHER section

Note: Russian version

`WeatherSectionParser` is responsible for parsing `WEATHER` section. This section describes additional weather conditions and contains one key-value pair per each line.

Section example:

```
[WEATHER]
  WindDirection 120.0
  WindSpeed 3.0
  Gust 0
  Turbulence 4
```

Output example:

```
{
    'weather': {
        'wind': {
            'direction': 120.0,
            'speed': 3.0,
        },
        'gust': Gust.none,
        'turbulence': Turbulence.very_strong,
    },
}
```

Output contains a `dict` with `weather` element.

Description:

`WindDirection` Wind direction in degrees.

Output path `weather.wind.direction`

Output type `float`

Output value original value converted to float number

`WindSpeed` Wind speed in meters per second.

Output path `weather.wind.speed`

Output type `float`

Output value original value converted to float number

`Gust` Number in range `[0,8,10,12]` which defines strength of wind gusts.

Output path `weather.gust`

Output type complex gust constant

`Turbulence` Number in range `[0,3,4,5,6]` which defines strength of wind turbulence.

Output path `weather.turbulence`

Output type complex turbulence constant

5.4 RespawnTime section

Note: Russian version

`RespawnTimeSectionParser` is responsible for parsing `RespawnTime` section. This section defines respawn time for different types of stationary objects.

Section example:

```
[RespawnTime]
  Bigship 1000000
  Ship 1000000
  Aeroanchored 1000000
  Artillery 1000000
  Searchlight 1000000
```

Output example:

```
{
    'respawn_time': {
        'ships': {
            'big': 1000000,
            'small': 1000000,
        },
        'balloons': 1000000,
        'artillery': 1000000,
        'searchlights': 1000000,
    },
}
```

Output contains a `dict` with `respawn_time` element.

Description:

Respawn time is measured by seconds.

`Bigship` Respawn time for big ships[1].

Output path `respawn_time.ships.big`

Output type `int`

Output value original value converted to integer number

`Ship` Respawn time for small ships[1].

Output path `respawn_time.ships.small`

Output type `int`

Output value original value converted to integer number

`Aeroanchored` Respawn time for balloons.

[1] See what big and small ships are: *ships categories*.

Output path `respawn_time.balloons`

Output type `int`

Output value original value converted to integer number

`Artillery` Respawn time for artillery.

Output path `respawn_time.artillery`

Output type `int`

Output value original value converted to integer number

`Searchlight` Respawn time for searchlights.

Output path `respawn_time.searchlights`

Output type `int`

Output value original value converted to integer number

Footnotes:

5.5 MDS section

Note: Russian version

`MDSSectionParser` is responsible for parsing `MDS` section. It contains one key-value pair per each line. Section describes different conditions in mission including Fog of War (FoW), AI and some other settings.

Note: MDS FoW functions are only enabled if the difficulty option "No FoW Icons" is **not** selected. This is a convenient way for server host to disable all FoW features without editing all mission files separately.

Section example:

```
[MDS]
  MDS_Radar_SetRadarToAdvanceMode 1
  MDS_Radar_RefreshInterval 0
  MDS_Radar_DisableVectoring 0
  MDS_Radar_EnableTowerCommunications 1
  MDS_Radar_ShipsAsRadar 0
  MDS_Radar_ShipRadar_MaxRange 100
  MDS_Radar_ShipRadar_MinHeight 100
  MDS_Radar_ShipRadar_MaxHeight 5000
  MDS_Radar_ShipSmallRadar_MaxRange 25
  MDS_Radar_ShipSmallRadar_MinHeight 0
  MDS_Radar_ShipSmallRadar_MaxHeight 2000
  MDS_Radar_ScoutsAsRadar 0
  MDS_Radar_ScoutRadar_MaxRange 2
  MDS_Radar_ScoutRadar_DeltaHeight 1500
  MDS_Radar_ScoutGroundObjects_Alpha 5
  MDS_Radar_ScoutCompleteRecon 0
  MDS_Misc_DisableAIRadioChatter 0
  MDS_Misc_DespawnAIPlanesAfterLanding 1
```

```
  MDS_Radar_HideUnpopulatedAirstripsFromMinimap 0
  MDS_Misc_HidePlayersCountOnHomeBase 0
  MDS_Misc_BombsCat1_CratersVisibilityMultiplier 1.0
  MDS_Misc_BombsCat2_CratersVisibilityMultiplier 1.0
  MDS_Misc_BombsCat3_CratersVisibilityMultiplier 1.0
```

Output example:

```
{
    'conditions': {
        'radar': {
            'advanced_mode': True,
            'refresh_interval': 0,
            'ships': {
                'big': {
                    'max_range': 100,
                    'min_height': 100,
                    'max_height': 5000,
                },
                'small': {
                    'max_range': 25,
                    'min_height': 0,
                    'max_height': 2000,
                },
            },
            'scouts': {
                'max_range': 2,
                'max_height': 1500,
                'alpha': 5,
            },
        },
        'scouting': {
            'scouts_affect_radar': False,
            'ships_affect_radar': False,
            'only_scouts_complete_targets': False,
        },
        'home_bases': {
            'hide_unpopulated': False,
            'hide_players_count': False,
            'hide_ai_aircrafts_after_landing': True,
        },
        'communication': {
            'vectoring': True,
            'tower_communication': True,
            'ai_radio_silence': False,
        },
        'crater_visibility_muptipliers': {
            'le_100kg': 1.0,
            'le_1000kg': 1.0,
            'gt_1000kg': 1.0,
        },
    },
}
```

Output contains a `dict` with a `conditions` element.

Description:

5.5.1 Radar

MDS_Radar_SetRadarToAdvanceMode Sets FoW to advanced mode: if this option is enabled, all FoW spotters on the map will show only those planes that are located inside assigned range & height limits. Range parameters are set for each home base object individually under home base `Base FoW` tab. If option is not set, player's side will see units' icons as long as it has at least one live radar.

Output path `conditions.radar.advance_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

MDS_Radar_RefreshInterval Radar refresh period (in seconds): tells the game how fast positions of detected objects are refreshed. Works with or without advanced radar mode.

Output path `conditions.radar.refresh_interval`

Output type `int`

Output value original value converted to integer number

MDS_Radar_ShipRadar_MaxRange Maximum range (in km) of detection of air targets by big ships.

Output path `conditions.radar.ships.big.max_range`

Output type `int`

Output value original value converted to integer number

MDS_Radar_ShipRadar_MinHeight Minimum height (in meters) of detection of air targets by big ships.

Output path `conditions.radar.ships.big.min_height`

Output type `int`

Output value original value converted to integer number

MDS_Radar_ShipRadar_MaxHeight Maximum height (in meters) of detection of air targets by big ships.

Output path `conditions.radar.ships.big.max_height`

Output type `int`

Output value original value converted to integer number

MDS_Radar_ShipSmallRadar_MaxRange Maximum range (in km) of detection of air targets by small ships.

Output path `conditions.radar.ships.small.max_range`

Output type `int`

Output value original value converted to integer number

MDS_Radar_ShipSmallRadar_MinHeight Minimum height (in meters) of detection of air targets by small ships.

Output path `conditions.radar.ships.small.min_height`

Output type `int`

Output value original value converted to integer number

`MDS_Radar_ShipSmallRadar_MaxHeight` Maximum height (in meters) of detection of air targets by small ships.

Output path `conditions.radar.ships.small.max_height`

Output type `int`

Output value original value converted to integer number

`MDS_Radar_ScoutRadar_MaxRange` Maximum scan range: determines the range (in km) in which scouts can identify other aircrafts.

Output path `conditions.radar.scouts.max_range`

Output type `int`

Output value original value converted to integer number

`MDS_Radar_ScoutRadar_DeltaHeight` Height limit of detection zone (in meters): defines the maximum altitude at which the reconnaissance aircraft can detect enemy ground targets.

Output path `conditions.radar.scouts.max_height`

Output type `int`

Output value original value converted to integer number

`MDS_Radar_ScoutGroundObjects_Alpha` Angle (in degrees) of earth scanning: determines the angle at which reconnaissance aircraft can detect enemy ground targets.

Output path `conditions.radar.scouts.alpha`

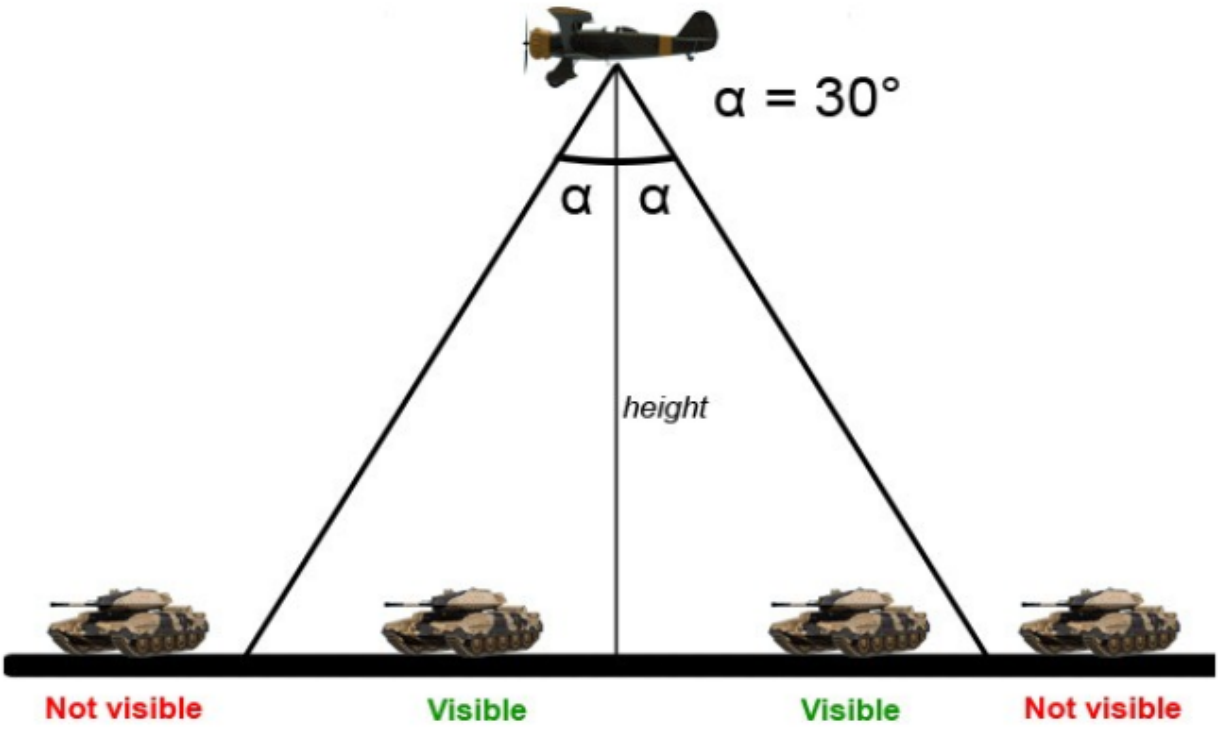
Output type `int`

Output value original value converted to integer number

Note: Scan delta height & scan alpha determine the range for which scouts can identify ground objects. The formula behind this is:

$$range = height * tan(alpha)$$

So, the higher the scouts are, the more area they cover.



Warning: The more scout planes you assign, the slower your game might run!

5.5.2 Scouting

`MDS_Radar_ShipsAsRadar` Treat ships as FoW spotters: makes ships spot enemy planes with their radars. Ships are divided into two groups. "Big Ships" that have powerful, long range radars and "Small Ships" that have less powerful, short range radars. If you want only big ships to act as FoW spotters, set all small ship settings to 0 and vice versa.

Note:

`Big Ships` with powerful, long range radar All CVs (aircraft carriers), all battleships and all cruisers.

`Small Ships` with less powerful, short range radar All destroyers.

Output path `conditions.scouting.ships_affect_radar`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`MDS_Radar_ScoutsAsRadar` Recon planes are FoW spotters: this will enable selected recon planes to spot ground units. Only selected recon planes are able to identify ground units (see *MDS_Scouts section*).

Output path `conditions.scouting.scouts_affect_radar`

Output type `bool`

Output value `True` if `1`, `False` otherwise

MDS_Radar_ScoutCompleteRecon Determines whether reconnaissance aircrafts are the only aircrafts allowed to complete recon targets.

Output path `conditions.scouting.only_scouts_complete_targets`

Output type `bool`

Output value `True` if `1`, `False` otherwise

5.5.3 Homebases

MDS_Radar_HideUnpopulatedAirstripsFromMinimap Hide enemy and unused airfields from minimap.

Output path `conditions.home_bases.hide_unpopulated`

Output type `bool`

Output value `True` if `1`, `False` otherwise

MDS_Misc_HidePlayersCountOnHomeBase This option, if enabled, will hide number of players that is displayed beside each home base object on your map on briefing screen.

Output path `conditions.home_bases.hide_players_count`

Output type `bool`

Output value `True` if `1`, `False` otherwise

MDS_Misc_DespawnAIPlanesAfterLanding Despawn AI aircrafts after they land and park: in dog fight mode when AI aircraft land and park, they will vanish from the map and release game resources. They will also not interfere with live players.

Output path `conditions.home_bases.hide_ai_aircrafts_after_landing`

Output type `bool`

Output value `True` if `1`, `False` otherwise

5.5.4 Communication

MDS_Radar_DisableVectoring Disables two vectoring commands from ground control orders menu: `Vector to target` and `Vector to home`. This can simulate early war scenarios where own planes couldn't be tracked by means of radar, Y-Verfahren, etc. Works also in single player & coop missions.

Output path `conditions.communication.vectoring`

Output type `bool`

Output value inverted original value converted to integer number: `True` if `0`, `False` otherwise

MDS_Radar_EnableTowerCommunications Enables communications menu (tab key by default) for human players in dogfight.

Output path `conditions.communication.tower_communication`

Output type `bool`

Output value `True` if `1`, `False` otherwise

MDS_Misc_DisableAIRadioChatter Disable radio messages sent by AI planes in dogfight.

Output path `conditions.communication.ai_radio_silence`

Output type `bool`

Output value `True` if `1`, `False` otherwise

5.5.5 Craters

You can modify time before bomb/gun/rockets craters disappear. Default multiplier is set to 1.0 (80 seconds) for all of them. By changing multipliers, you can make craters visible for longer time. However this only works in single player mission and coop missions. Setting long crater durations in dogfight missions would cause inconsistency between players, since dogfight mode allows joining anytime.

`MDS_Misc_BombsCat1_CratersVisibilityMultiplier` Multiplier for visibility time for craters caused by guns and rockets and bombs which weight is less then or equal 100 kg.

Output path `conditions.crater_visibility_muptipliers.le_100kg`

Output type `float`

Output value original value converted to float number

`MDS_Misc_BombsCat2_CratersVisibilityMultiplier` Multiplier for visibility time for craters caused by torpedoes, TinyTim and bombs which weight is less then or equal 1000 kg.

Output path `conditions.crater_visibility_muptipliers.le_1000kg`

Output type `float`

Output value original value converted to float number

`MDS_Misc_BombsCat3_CratersVisibilityMultiplier` Multiplier for visibility time for craters caused by bombs which weight is greater then 1000 kg.

Output path `conditions.crater_visibility_muptipliers.gt_1000kg`

Output type `float`

Output value original value converted to float number

5.6 MDS_Scouts section

Note: Russian version

`MDSScoutsSectionParser` is responsible for parsing sections which starts with `MDS_Scouts_` prefix. Those sections define lists of aircrafts which can be used as scouts.

There are two known sections:

1. `MDS_Scouts_Red`
2. `MDS_Scouts_Blue`

Each line of those sections contains a code name of an aircraft.

Section example:

```
[MDS_Scouts_Red]
  B-25H-1NA
  B-25J-1NA
  BeaufighterMk21
```

Output example:

```
{
    'scouts_red': {
      'belligerent': Belligerents.red,
      'aircrafts': [
          "B-25H-1NA",
          "B-25J-1NA",
          "BeaufighterMk21",
      ],
    }
}
```

Output contains a dictionary with a single value. It can be accessed by `scouts_{suffix}` key, where `suffix` is original suffix, converted to lower case. So, possible keys are:

1. `scouts_red`
2. `scouts_blue`

The value itself is also a dictionary, which contains a list of aircraft code names and a belligerent constant.

5.7 Chiefs section

Note: Russian version

`ChiefsSectionParser` is responsible for parsing `Chiefs` section. It describes moving objects or their groups. Each of them is defined on a separate line. There are 4 types of moving objects:

1. usual vehicles;
2. armored vehicles;
3. trains;
4. ships.

First 3 types have same list of parameters. Ships have some extra parameters.

Section example:

```
[Chiefs]
  0_Chief Armor.1-BT7 2
  1_Chief Vehicles.GAZ67 1
  2_Chief Trains.USSR_FuelTrain/AA 1
  3_Chief Ships.Niobe 2
  4_Chief Ships.G5 1 60 3 2.0
```

Output example:

```
{
    'moving_units': [
        {
            'id': '0_Chief',
            'code': '1-BT7',
            'type': UnitTypes.armor,
            'belligerent': Belligerents.blue,
        },
        {
```

```
            'id': '1_Chief',
            'code': 'GAZ67',
            'type': UnitTypes.vehicle,
            'belligerent': Belligerents.red,
        },
        {
            'id': '2_Chief',
            'code': 'USSR_FuelTrain/AA',
            'type': UnitTypes.train,
            'belligerent': Belligerents.red,
        },
        {
            'id': '3_Chief',
            'code': 'Niobe',
            'type': UnitTypes.ship,
            'belligerent': Belligerents.blue,
        },
        {
            'id': '4_Chief',
            'code': 'G5',
            'type': UnitTypes.ship,
            'belligerent': Belligerents.red,
            'hibernation': 60,
            'skill': Skills.ace,
            'recharge_time': 2.0,
        },
    ],
}
```

Description:

The output of the parser is a dictionary with `moving_units` element. It contains a list of dictionaries containing information about each object.

5.7.1 Common parameters

Let's examine common parameters using first line from example above:

```
0_Chief Armor.1-BT7 2
```

`0_Chief` Object's ID. Contains `Chief` word prefixed by a sequence number. This ID identifies a moving object or a group of them. In latter case, events log will contain this code followed by an in-group number of an object, e.g.:

```
[5:25:14 PM] 0_Chief9 destroyed by 1_Chief2 at 11149.903 43949.902
```

Here we can see that 9th object from group `0_Chief` was destroyed by 2nd object from group `1_Chief`.

Output path `id`

Output type `str`

Output value original string value

`Armor.1-BT7` Defines unit type and object's code.

Output path `type`

Output type complex unit type constant

Output path `code`

Output type `str`

Output value original string value

2 Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

5.7.2 Ships extra parameters

Ships have 3 extra parameters. Let's see an example:

```
3_Chief Ships.G5 1 60 3 2.0
```

First 3 parameters are similar to the ones described above. The other parameters are:

60 Hibernation time (in minutes): during this time a ship will be inactive. After that it will start following own route.

Output path `hibernation`

Output type `int`

Output value original value converted to integer number

3 Skill level of gunners managing anti-aircraft guns.

Output path `skill`

Output type complex skills constant

2.0 Recharge time (in minutes) of anti-aircraft guns of the ship.

Output path `recharge_time`

Output type `float`

Output value original value converted to float number

5.8 Chief Road section

Note: Russian version

`ChiefRoadSectionParser` is responsible for parsing `N_Chief_Road` section. Every object listed in *Chiefs section* has own route described in own `N_Chief_Road` section, where `N` is the sequence number within `Chiefs` section.

Section example:

```
[0_Chief_Road]
  21380.02 41700.34 120.00 10 2 3.055555582046509
  21500.00 41700.00 120.00
  50299.58 35699.85 120.00 0 3 2.6388890743255615
  60284.10 59142.93 120.00
  84682.13 98423.69 120.00
```

Output example:

```
{
    'route_0_Chief': [
        GroundRoutePoint(
            pos=Point2D(21380.02, 41700.34),
            is_checkpoint=True,
            delay=10,
            section_length=3,
            speed=11.0,
        ),
        GroundRoutePoint(
            pos=Point2D(21500.00, 41700.00),
            is_checkpoint=False,
        ),
        GroundRoutePoint(
            pos=Point2D(50299.58, 35699.85),
            is_checkpoint=True,
            delay=0,
            section_length=3,
            speed=9.5,
        ),
        GroundRoutePoint(
            pos=Point2D(60284.10, 59142.93),
            is_checkpoint=False,
        ),
        GroundRoutePoint(
            pos=Point2D(84682.13, 98423.69),
            is_checkpoint=False,
        ),
    ],
}
```

Description:

Each line in `N_Chief_Road` section describes a single waypoint. There are two types of waypoints: created by user and created automatically by full mission editor.

The output of the parser is a dictionary with `route_N_Chief` item which is a list of `GroundRoutePoint`.

Manually created waipoints have 6 parameters, while auto-created ones have only 3 of them. The last waypoint always has 3 parameters, and it is always defined by user. So, don't get mislead.

The purpose of intermediate auto-created waypoints is to create the most efficient route:

1. vehicles tend to move by roads, by bridges, and by the most flat terrains;
2. trains can move only by rails; intermediate points are created at the points where the direction is changed;
3. ships tend to follow coastlines and river banks if they come to them close enough.

Let's examine a description of a waypoint which was created manually by user. It has all parameters included:

```
21380.02 41700.34 120.00 10 2 3.055555582046509
```

`21380.02` X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

41700.34 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

120.00 This is the quite strange parameter. The true meaning is not known, but its value depends on the type of surface the point is located on. Also, the value is specific for different types of units:

1. Vehicles: the value for all manual waypoints is set to `120.0`. the value for auto-created waypoint can be set to `20.0` or `120.0`. The former value tells that the point in located on the road. The latter one tells that the point is located in the off-road. Negative values tell about start or end of a bridge. Usually, negative values come in pairs.
2. Trains: all waypoints have the value of `20.0`. This means that trains can move only by railways. Negative values tell about start or end of a bridge. Usually, negative values come in pairs.
3. Ships: all waypoints have the value of `120.0`. This means that ships can move only by water.

Output path this value is not present in the output.

10 Delay (in minutes): this parameter tells how much a unit have to wait until it starts movement to the next user-defined point.

Output path `delay`

Output type `int`

Output value original value converted to integer number

2 Section length. Here `section` means current user-defined waypoint, next user-defined point and all intermediate points between them.

Output path `section_length`

Output type `int`

Output value original value converted to integer number

3.055555582046509 The speed of the unit at the current point of the route. This parameter is set automatically by full mission editor depending on the unit type. Multiply value by `CHIEF_SPEED_COEFFICIENT` to get speed in `km/h`.

Output path `speed`

Output type `float`

Output value original value converted to float number and multiplied by `CHIEF_SPEED_COEFFICIENT`.

We decided to mark each user-defined waypoint as a checkpoint (except the last one).

Output path `is_check_point`

Output type `bool`

Output value `True` if point defines start of a section, `False` if it is an intermediate point or the last point

5.9 NStationary section

Note: Russian version

`NStationarySectionParser` is responsible for parsing `NStationary` section. Each line of this section describes a single stationary object (except buildings and houses).

Section example:

```
[NStationary]
  0_Static vehicles.stationary.Stationary$Wagon1 1 152292.72 89662.80 360.00 0.0
  1_Static vehicles.artillery.Artillery$SdKfz251 2 31333.62 90757.91 600.29 0.0 0 1 1
  2_Static vehicles.planes.Plane$I_16TYPE24 1 134146.89 88005.43 336.92 0.0 null 2 1.
↪0 I-16type24_G1_RoW3.bmp 1
  3_Static ships.Ship$G5 1 83759.05 115021.15 360.00 0.0 60 3 1.4
```

Output example:

```
{
    'stationary': [
        StationaryObject(
            belligerent=Belligerents.red,
            id='0_Static',
            code='Wagon1',
            pos=Point2D(152292.72, 89662.80),
            rotation_angle=0.00,
            type=UnitTypes.stationary,
        ),
        StationaryArtillery(
            id='1_Static',
            belligerent=Belligerents.blue,
            code='SdKfz251',
            pos=Point2D(31333.62, 90757.91),
            rotation_angle=240.29,
            type=UnitTypes.artillery,
            awakening_time=0.0,
            range=0,
            skill=Skills.average,
            use_spotter=True,
        ),
        StationaryAircraft(
            id='2_Static',
            code='I_16TYPE24',
            belligerent=Belligerents.red,
            pos=Point2D(134146.89, 88005.43),
            rotation_angle=336.92,
            type=UnitTypes.aircraft,
            air_force=AirForces.vvs_rkka,
            allows_spawning=True,
            show_markings=True,
            is_restorable=True,
            skin="I-16type24_G1_RoW3.bmp",
        ),
        StationaryShip(
            belligerent=Belligerents.red,
            id='9_Static',
```

```
            code='G5',
            recharge_time=1.4,
            pos=Point2D(83759.05, 115021.15),
            rotation_angle=0.00,
            skill=Skills.ace,
            type=UnitTypes.ship,
            awakening_time=60.0,
        ),
    ],
}
```

The output of the parser is a dictionary with `stationary` item which contains a list of stationary objects.

Set of parameters may differ for different types of units:

1. all objects have at least 7 parameters;
2. artillery has 3 own extra parameters;
3. aircrafts have 5 own extra parameters;
4. ships have 3 own extra parameters.

Let's examine all of them:

5.9.1 Usual objects

Usual objects — these are all objects which have usual set of parameters, namely: ballons, lights, radio stations, trains, vehicles and so on.

We use `StationaryObject` data structure to store information about such objects.

Definition example:

```
0_Static vehicles.stationary.Stationary$Wagon1 1 152292.72 89662.80 360.00 0.0
```

`0_Static` Object ID which is given by full mission editor. Contains `Static` word prefixed by a sequence number.

Output path `id`

Output type `str`

Output value original string value

`vehicles.stationary.Stationary$Wagon1` Unit type (`stationary`) and code name (`Wagon1`).

Output path `type`

Output type complex unit type constant

Output path `code`

Output type `str`

Output value original string value

`1` Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

`152292.72` X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

`89662.80` Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`360.00` Angle of rotation.

Output path `rotation_angle`

Output type `float`

Output value original value converted to float number and taken modulo 360

`0.0` This parameter is not used by usual objects. It has a meaning only for artillery objects (see below).

5.9.2 Artillery

Artillery has all the same parameters as usual objects. Also artillery in new versions of game has some extra parameters which are described below.

We use `StationaryArtillery` data structure to store information about artillery.

Definition example:

```
1_Static vehicles.artillery.Artillery$SdKfz251 2 31333.62 90757.91 600.29 0.0 0 1 1
```

`0.0` Time of awakening (in minutes): it's a time which will pass since enemy unit enters object's range till object will react on that unit.

Output path `awakening_time`

Output type `float`

Output value original value converted to float number (`0.0` for objects from old game versions)

`0` Range of fire.

Output path `range`

Output type `int`

Output value original value converted to integer number (`0` for objects from old game versions)

`1` Skill level of gunners.

Output path `skill`

Output type complex skills constant (`None` for objects from old game versions)

1 Tells whether to use spotter or not.

Output path `use_spotter`

Output type `bool`

Output value `True` if `1`, `False` otherwise (`False` for objects from old game versions)

5.9.3 Aircrafts

Aircrafts have all the same parameters as usual objects. Also aircrafts in new versions of game have some extra parameters which are described below.

We use `StationaryAircraft` data structure to store information about aircrafts.

Definition example:

```
2_Static vehicles.planes.Plane$I_16TYPE24 2 134146.89 88005.43 336.92 0.0 de 2 1.0 I-
↪16type24_G1_RoW3.bmp 1
```

null Code name of the air force. E.g., `de` or `fr`. For some unknown reason air force of USSR has `null` code name in `NStationary` section.

Output path `air_force`

Output type complex air forces

Output value constant (`None` for objects from old game versions)

2 Polysemantic parameter which can have next values:

Value	Meaning
0	Usage of aircraft by humans is **not allowed**
1	Usage of aircraft by humans is **allowed**
2	Usage of aircraft by humans is **allowed**, object will be restored after successfull landing

Output path `allows_spawning`

Output type `bool`

Output value `True` if `1` or `2`, `False` otherwise (`False` for objects from old game versions)

Output path `restorable`

Output type `bool`

Output value `True` if `2`, `False` otherwise (`False` for objects from old game versions)

1.0 Not used (not present in old game versions).

I-16type24_G1_RoW3.bmp Skin name.

Output path `skin`

Output type `str`

Output value original string value or `None` if `null` (`None` for objects from old game versions)

Default `null`

1 Show markings or not.

Output path `show_markings`

Output type `bool`

Output value `True` if `1`, `False` otherwise (`None` for objects from old game versions)

5.9.4 Ships

Ships have all the same parameters as usual objects. Also ships in new versions of game have some extra parameters which are described below.

We use `StationaryShip` data structure to store information about ships.

Definition example:

```
3_Static ships.Ship$G5 1 83759.05 115021.15 360.00 0.0 60 3 1.4
```

60 Time of awakening (in minutes): it's a time which will pass since enemy unit enters ship's range till ship will react on that unit.

Output path `awakening_time`

Output type `float`

Output value original value converted to float number (`0.0` for objects from old game versions)

3 Skill level of gunners.

Output path `skill`

Output type complex skills constant

Output value constant (`None` for objects from old game versions)

1.4 Recharge time (in minutes) of anti-aircraft guns of the ship.

Output path `recharge_time`

Output type `float`

Output value original value converted to float number (`0.0` for objects from old game versions)

5.10 Buildings section

Note: Russian version

`BuildingsSectionParser` is responsible for parsing `Buildings` section. Each line of this section describes a single building.

Section example:

```
[Buildings]
  0_bld House$Tent_Pyramid_US 1 43471.34 57962.08 630.00
```

Output example:

```
{
    'buildings': [
        Building(
            id='0_bld',
            belligerent=Belligerents.red,
            code='Tent_Pyramid_US',
```

```
            pos=Point2D(43471.34, 57962.08),
            rotation_angle=270.00,
        ),
    ],
}
```

The result is a `dict` with `buildings` item which contains a list of buildings.

We use `Building` structure to store information about buildings.

Description:

`0_bld` Object ID which is given by full mission editor. Contains `bld` word prefixed by a sequence number.

Output path `id`

Output type `str`

Output value original string value

`House$Tent_Pyramid_US` Building type (`House`) and code name (`Tent_Pyramid_US`). Type is not present in the output because all buildings have type `house`.

Output path `code`

Output type `str`

Output value original string value

`1` Code number of belligerent the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

`43471.34` X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

`57962.08` Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`630.00` Angle of rotation.

Output path `rotation_angle`

Output type `float`

Output value original value converted to float number and taken modulo 360

5.11 Target section

Note: Russian version

`TargetSectionParser` is responsible for parsing `Target` section. Each line of this section describes a single mision target.

Section example:

```
[Target]
  3 1 1 50 500 133978 87574 1150
  3 0 1 40 501 134459 85239 300 0 1_Chief 134360 85346
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.recon,
            'priority': TargetPriorities.secondary,
            'in_sleep_mode': True,
            'delay': 50,
            'requires_landing': False,
            'pos': Point2D(133978.0, 87574.0),
            'radius': 1150,
        },
        {
            'type': TargetTypes.recon,
            'priority': TargetPriorities.primary,
            'in_sleep_mode': True,
            'delay': 40,
            'requires_landing': True,
            'pos': Point2D(134459.0, 85239.0),
            'radius': 300,
            'object': {
                'waypoint': 0,
                'id': '1_Chief',
                'pos': Point2D(134360.0, 85346.0),
            },
        },
    ],
}
```

The output of the parser is a `dict` with `targets` item which contains a list of dictionaries. Each dictionary represents a single target.

There are 8 different types of targets and 3 types of target priorities. Some different types of targets have identical sets of parameters.

- *Destroy*
- *Destroy area*
- *Destroy bridge*
- *Recon*
- *Escort*
- *Cover*
- *Cover area*
- *Cover bridge*

5.11.1 Destroy

Definition example:

```
0 0 0 0 500 90939 91871 0 1 10_Chief 91100 91500
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.destroy,
            'priority': TargetPriorities.primary,
            'in_sleep_mode': False,
            'delay': 0,
            'destruction_level': 50,
            'pos': Point2D(90939.0, 91871.0),
            'object': {
                'waypoint': 1,
                'id': '10_Chief',
                'pos': Point2D(91100.0, 91500.0),
            },
        },
    ],
}
```

0 Target type (destroy).

Output path `type`

Output type complex target type constant

0 Target priority (primary).

Output path `priority`

Output type complex target priority constant

0 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

0 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

500 Destruction level multiplied by 10.

Output path `destruction_level`

Output type `int`

Output value original value converted to integer number and divided by 10

90939 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

91871 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

0 Is not used by targets of this type.

1 Waypoint number of the object which must be destroyed.

Output path `object.waypoint`

Output type `int`

Output value original value converted to integer number

10_Chief ID of the object which must be destroyed.

Output path `object.id`

Output type `str`

Output value original string value

91100 X coordinate of the object which must be destroyed.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

91500 Y coordinate of the object which must be destroyed.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.11.2 Destroy area

Definition example:

```
1 1 1 60 750 133960 87552 1350
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.destroy_area,
            'priority': TargetPriorities.secondary,
            'in_sleep_mode': True,
            'delay': 60,
            'destruction_level': 75,
```

```
            'pos': Point2D(133960.0, 87552.0),
            'radius': 1350,
        },
    ],
}
```

1 Target type (destroy area).

Output path `type`

Output type complex target type constant

1 Target priority (secondary).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

60 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

750 Destruction level multiplied by 10.

Output path `destruction_level`

Output type `int`

Output value original value converted to integer number and divided by 10

133960 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

87552 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

1350 Area radius.

Output path `radius`

Output type `int`

Output value original value converted to integer number

5.11.3 Destroy bridge

Definition example:

```
2 2 1 30 500 135786 84596 0 0  Bridge84 135764 84636
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.destroy_bridge,
            'priority': TargetPriorities.hidden,
            'in_sleep_mode': True,
            'delay': 30,
            'pos': Point2D(135786.0, 84596.0),
            'object': {
                'id': 'Bridge84',
                'pos': Point2D(135764.0, 84636.0),
            },
        },
    ],
}
```

2 Target type (destroy bridge).

Output path `type`

Output type complex target type constant

2 Target priority (hidden).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

30 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

500 Is not used by targets of this type.

133960 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

87552 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

0 Is not used by targets of this type.

0 Is not used by targets of this type.

Bridge84 ID of the bridge which must be destroyed.

Output path `object.id`

Output type `str`

Output value original string value

135764 X coordinate of the bridge which must be destroyed.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

84636 Y coordinate of the bridge which must be destroyed.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.11.4 Recon

There are 2 possible definitions:

```
3 1 1 50 500 133978 87574 1150
3 0 1 40 501 134459 85239 300 0 1_Chief 134360 85346
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.recon,
            'priority': TargetPriorities.secondary,
            'in_sleep_mode': True,
            'delay': 50,
            'requires_landing': False,
            'pos': Point2D(133978.0, 87574.0),
            'radius': 1150,
        },
        {
            'type': TargetTypes.recon,
            'priority': TargetPriorities.primary,
            'in_sleep_mode': True,
            'delay': 40,
            'requires_landing': True,
            'pos': Point2D(134459.0, 85239.0),
            'radius': 300,
            'object': {
                'waypoint': 0,
                'id': '1_Chief',
```

```
                'pos': Point2D(134360.0, 85346.0),
            },
        },
    ],
}
```

Let's examine second definition:

3 Target type (recon).

Output path `type`

Output type complex target type constant

0 Target priority (primary).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

40 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

501 Tells whether you need to land near the target to succeed.

Output path `requires_landing`

Output type `bool`

Output value `True` if `501`, `False` otherwise

134459 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

87574 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

300 Maximal distance to target if you need to land.

Output path `radius`

Output type `int`

Output value original value converted to integer number

0 Waypoint number of the object which you need to recon.

Output path `object.waypoint`

Output type `int`

Output value original value converted to integer number

`1_Chief` ID of the object which you need to recon.

Output path `object.id`

Output type `str`

Output value original string value

`134360` X coordinate of the object which you need to recon.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

`85346` Y coordinate of the object which you need to recon.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.11.5 Escort

Definition example:

```
4 0 1 10 750 134183 85468 0 1 r0100 133993 85287
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.escort,
            'priority': TargetPriorities.primary,
            'in_sleep_mode': True,
            'delay': 10,
            'destruction_level': 75,
            'pos': Point2D(134183.0, 85468.0),
            'object': {
                'waypoint': 1,
                'id': 'r0100',
                'pos': Point2D(133993.0, 85287.0),
            },
        },
    ],
}
```

`4` Target type (escort).

Output path `type`

Output type complex target type constant

`0` Target priority (primary).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

10 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

750 Destruction level multiplied by 10.

Output path `destruction_level`

Output type `int`

Output value original value converted to integer number and divided by 10

134183 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

91871 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

0 Is not used by targets of this type.

1 Waypoint number of the flight which must be escorted.

Output path `object.waypoint`

Output type `int`

Output value original value converted to integer number

r0100 ID of the flight which must be escorted.

Output path `object.id`

Output type `str`

Output value original string value

133993 X coordinate of the flight which must be escorted.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

85287 Y coordinate of the flight which must be escorted.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.11.6 Cover

Definition example:

```
5 1 1 20 250 132865 87291 0 1 1_Chief 132866 86905
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.cover,
            'priority': TargetPriorities.secondary,
            'in_sleep_mode': True,
            'delay': 20,
            'destruction_level': 25,
            'pos': Point2D(132865.0, 87291.0),
            'object': {
                'waypoint': 1,
                'id': '1_Chief',
                'pos': Point2D(132866.0, 86905.0),
            },
        },
    ],
}
```

5 Target type (cover).

Output path `type`

Output type complex target type constant

1 Target priority (secondary).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

20 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

250 Destruction level multiplied by 10.

Output path `destruction_level`

Output type `int`

Output value original value converted to integer number and divided by 10

132865 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

87291 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

0 Is not used by targets of this type.

1 Waypoint number of the object which must be covered.

Output path `object.waypoint`

Output type `int`

Output value original value converted to integer number

1_Chief ID of the object which must be covered.

Output path `object.id`

Output type `str`

Output value original string value

132866 X coordinate of the object which must be covered.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

86905 Y coordinate of the object which must be covered.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.11.7 Cover area

Definition example:

```
6 1 1 30 500 134064 88188 1350
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.cover_area,
            'priority': TargetPriorities.secondary,
```

```
            'in_sleep_mode': True,
            'delay': 30,
            'destruction_level': 50,
            'pos': Point2D(134064.0, 88188.0),
            'radius': 1350,
        },
    ],
}
```

6 Target type (cover area).

Output path `type`

Output type complex target type constant

1 Target priority (secondary).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

30 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

500 Destruction level multiplied by 10.

Output path `destruction_level`

Output type `int`

Output value original value converted to integer number and divided by 10

134064 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

88188 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

1350 Area radius.

Output path `radius`

Output type `int`

Output value original value converted to integer number

5.11.8 Cover bridge

Definition example:

```
7 2 1 30 500 135896 84536 0 0  Bridge84 135764 84636
```

Output example:

```
{
    'targets': [
        {
            'type': TargetTypes.cover_bridge,
            'priority': TargetPriorities.hidden,
            'in_sleep_mode': True,
            'delay': 30,
            'pos': Point2D(135896.0, 84536.0),
            'object': {
                'id': 'Bridge84',
                'pos': Point2D(135764.0, 84636.0),
            },
        },
    ],
}
```

7 Target type (cover bridge).

Output path `type`

Output type complex target type constant

2 Target priority (hidden).

Output path `priority`

Output type complex target priority constant

1 Tells whether sleep mode is turned on.

Output path `in_sleep_mode`

Output type `bool`

Output value `True` if `1`, `False` otherwise

30 Delay (in minutes).

Output path `delay`

Output type `int`

Output value original value converted to integer number

500 Is not used by targets of this type.

135896 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

84536 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`0` Is not used by targets of this type.

`0` Is not used by targets of this type.

`Bridge84` ID of the bridge which must be covered.

Output path `object.id`

Output type `str`

Output value original string value

`135764` X coordinate of the bridge which must be covered.

Output path `object.pos.x`

Output type `float`

Output value original value converted to float number

`84636` Y coordinate of the bridge which must be covered.

Output path `object.pos.y`

Output type `float`

Output value original value converted to float number

5.12 BornPlace section

Note: Russian version

`BornPlaceSectionParser` is responsible for parsing `BornPlace` section. Each line of this section describes a single homebase.

Section example:

```
[BornPlace]
  1 3000 121601 74883 1 1000 200 0 0 0 5000 50 0 1 1 0 0 3.8 1 0 0 0 0
```

Output example:

```
{
    'home_bases': [
        {
            'range': 3000,
            'belligerent': Belligerents.red,
            'show_default_icon': False,
            'friction': {
                'enabled': False,
                'value': 3.8,
            },
            'spawning': {
                'enabled': True,
                'with_parachutes': True,
                'max_pilots': 0,
```

```
                'in_stationary': {
                    'enabled': False,
                    'return_to_start_position': False,
                },
                'in_air': {
                    'height': 1000,
                    'speed': 200,
                    'heading': 0,
                    'conditions': {
                        'always': False,
                        'if_deck_is_full': False,
                    },
                },
                'aircraft_limitations': {
                    'enabled': True,
                    'consider_lost': True,
                    'consider_stationary': True,
                },
            },
            'radar': {
                'range': 50,
                'min_height': 0,
                'max_height': 5000,
            },
            'pos': Point2D(121601.0, 74883.0),
        },
    ],
}
```

Description:

The output of the parser is a `dict` with `homebases` item which contains a list of of dictionaries. Each dictionary contains information about single homebase.

1 Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

3000 Homebase range (in meters).

Output path `range`

Output type `int`

Output value original value converted to integer number

121601 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

74883 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`1` Tells whether users will have parachutes.

Output path `spawning.with_parachutes`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`1000` Initial height of aircraft (in meters) if it was spawned in the air.

Output path `spawning.in_air.height`

Output type `int`

Output value original value converted to integer number

`200` Initial speed of aircraft (in km/h) if it was spawned in the air.

Output path `spawning.in_air.speed`

Output type `int`

Output value original value converted to integer number

`0` Initial heading of aircraft (in degrees) if it was spawned in the air.

Output path `spawning.in_air.heading`

Output type `int`

Output value original value converted to integer number

`0` Max number of pilots who can take off from this homebase. `0` means unlimited.

Output path `spawning.max_pilots`

Output type `int`

Output value original value converted to integer number

`0` Radar detection min height (in meters).

Output path `radar.min_height`

Output type `int`

Output value original value converted to integer number

`5000` Radar detection max height (in meters).

Output path `radar.max_height`

Output type `int`

Output value original value converted to integer number

`50` Radar detection range (in km).

Output path `radar.range`

Output type `int`

Output value original value converted to integer number

`0` Spawn only in air.

Output path `spawning.in_air.conditions.always`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`1` Enable aircraft limits.

Output path `spawning.aircraft_limitations.enabled`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`1` Homebase looses aircrafts as they get destroyed.

Output path `spawning.aircraft_limitations.consider_lost`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`0` Disable spawning. Output has inverted value.

Output path `spawning.enabled`

Output type `bool`

Output value `True` if `0`, `False` otherwise

`0` Enable friction.

Output path `friction.enabled`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`3.8` Friction value.

Output path `friction.value`

Output type `float`

Output value original value converted to float number

`1` Homebase looses aircrafts as stationary aircrafts get destroyed.

Output path `spawning.aircraft_limitations.consider_stationary`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`0` Render homebase icon at default position.

Output path `show_default_icon`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`0` Spawn in air if deck is full.

Output path `spawning.in_air.conditions.if_deck_is_full`

Output type `bool`

Output value `True` if `1`, `False` otherwise

`0` Spawn in stationary aircrafts.

Output path `spawning.in_stationary.enabled`

Output type `bool`

Output value `True` if `1`, `False` otherwise

0 Return stationary aircraft to start position after landing.

Output path `spawning.in_stationary.return_to_start_position`

Output type `bool`

Output value `True` if `1`, `False` otherwise

5.13 BornPlace aircrafts section

Note: Russian version

`BornPlaceAircraftsSectionParser` is responsible for parsing `BornPlaceN` section, where `N` is sequence number of the home base. This section describes aircrafts which are available on the home base #N.

Each line describes attributes of a single aircraft. Lines, which start with `+`, mark continuation of previous line. Max line length is approximately 210-220 characters.

Section example:

```
[BornPlace1]
  Bf-109F-4 -1 1sc250 4sc50
  Bf-109G-6_Late 0
  Ju-88A-4 10 28xSC50 28xSC50_2xSC250 28xSC50_4xSC250
  + 2xSC1800 2xSC2000
```

Output example:

```
{
    'home_base_aircrafts_1': [
        {
            'code': 'Bf-109F-4',
            'limit': None,
            'weapon_limitations': [
                '1sc250',
                '4sc50',
            ],
        },
        {
            'code': 'Bf-109G-6_Late',
            'limit': 0,
            'weapon_limitations': [],
        },
        {
            'code': 'Ju-88A-4',
            'limit': 10,
            'weapon_limitations': [
                '28xSC50',
                '28xSC50_2xSC250',
                '28xSC50_4xSC250',
                '2xSC1800',
                '2xSC2000',
            ],
        },
    ],
}
```

Description:

The output of the parser is a `dict` with `home_base_aircrafts_N` item, where `N` is original home base number. This item contains a list of dictionaries. Each dictionary stores information about single aircraft.

Let's examine the first line.

`Bf-109F-4` Aircraft code name.

Output path `code`

Output type `str`

Output value original string value

`0` Number of available aircrafts. This parameter makes sence only if home base has aircraft limitations turned on.

`-1` means that the number of aircrafts is unlimited.

`0` means that aircraft will not even be present in the list of available aircrafts in briefing.

Output path `limit`

Output type `int`

Output value `None` if `-1`, original value converted to integer number otherwise (always `None` for games of old versions)

`1sc250 4sc50` List of code names of allowed weapons for this aircraft. This part is optional: if it is not present, than all available weapons will be allowed.

Output path `weapon_limits`

Output type `list`

Output value list of strings (list is always empty for games of old versions)

5.14 BornPlace air forces section

Note: Russian version

`BornPlaceAirForcesSectionParser` is responsible for parsing `BornPlaceCountriesN` section, where `N` is sequence number of the homebase. This section defines a list of available airforces for homebase #N.

Each line contains a code name of a single air force.

Section example:

```
[BornPlaceCountries1]
  de
  ru
```

Output example:

```
{
    'home_base_air_forces_1': [
        AirForces.luftwaffe,
        AirForces.vvs_rkka,
    ],
}
```

The output of the parser is a `dict` with `home_base_air_forces_N` item, where `N` is original homebase number. The value is a list of air forces.

5.15 StaticCamera section

Note: Russian version

`StaticCameraSectionParser` is responsible for parsing `StaticCamera` section. Each line of this section describes a single camera.

Section example:

```
[StaticCamera]
  38426 65212 35 2
```

Output example:

```
{
    'cameras': [
        {
            'belligerent': Belligerents.blue,
            'pos': Point3D(38426.0, 65212.0, 35.0),
        },
    ],
}
```

Description:

The output of the parser is a `dict` with a `cameras` item. It contains a list of dictionaries where each dictionary represents a single camera.

38426 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

65212 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

35 Z coordinate.

Output path `pos.z`

Output type `float`

Output value original value converted to float number

2 Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

5.16 FrontMarker section

Note: Russian version

`FrontMarkerSectionParser` is responsible for parsing `FrontMarker` section. Each line of this section describes a single frontline marker.

Section example:

```
[FrontMarker]
  FrontMarker0 7636.65 94683.02 1
```

Output example:

```
{
    'markers': [
        FrontMarker(
            id="FrontMarker0",
            belligerent=Belligerents.red,
            pos=Point2D(7636.65, 94683.02),
        ),
    ],
}
```

Description:

The output of the parser is a `dict` with `markers` item. It contains a list of dictionaries where each dictionary represents a single frontline marker.

`FrontMarker0` Marker ID which is given by full mission editor. Contains `FrontMarker` word suffixed by a sequence number.

Output path `id`

Output type `str`

Output value original string value

`7636.65` X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

`94683.02` Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`1` Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

5.17 Rocket section

Note: Russian version

`RocketSectionParser` is responsible for parsing `Rocket` section. Each line of this section describes a single launchable rocket.

Section example:

```
[Rocket]
  0_Rocket Fi103_V1_ramp 2 84141.38 114216.82 360.00 60.0 10 80.0 83433.91 115445.49
  1_Rocket Fi103_V1_ramp 2 84141.38 114216.82 360.00 60.0 10 80.0
```

Output example:

```
{
    'rockets': [
        Rocket(
            id='0_Rocket',
            code='Fi103_V1_ramp',
            belligerent=Belligerents.blue,
            pos=Point2D(84141.38, 114216.82),
            rotation_angle=0.00,
            delay=60.0,
            count=10,
            period=80.0,
            destination=Point2D(83433.91, 115445.49),
        ),
        Rocket(
            id='1_Rocket',
            code='Fi103_V1_ramp',
            belligerent=Belligerents.blue,
            pos=Point2D(84141.38, 114216.82),
            rotation_angle=0.00,
            delay=60.0,
            count=10,
            period=80.0,
            destination=None,
        ),
    ],
}
```

The output of the parser is a `dict` with `rockets` item. It contains a list of dictionaries where each dictionary represents a single rocket.

A line can have 2 optional parameters: `X` and `Y` destination coordinates.

Let's examine the first line.

`0_Rocket` Rocket ID which is given by full mission editor. Contains `Rocket` word prefixed by a sequence number.

Output path `id`

Output type `str`

Output value original string value

`Fi103_V1_ramp` Code name.

Output path `code`

Output type `str`

Output value original string value

2 Code number of army the object belongs to.

Output path `belligerent`

Output type complex belligerents constant

84141.38 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

114216.82 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

360.00 Angle of rotation.

Output path `rotation_angle`

Output type `float`

Output value original value converted to float number and taken modulo 360

60.0 Delay (in minutes): this parameter tells how much a rocket have to wait until it will be launched.

Output path `delay`

Output type `float`

Output value original value converted to float number

10 Number of rockets to launch.

Output path `count`

Output type `int`

Output value original value converted to integer number

80.0 Period of rocket launch.

Output path `period`

Output type `float`

Output value original value converted to float number

83433.91 Destination X coordinate.

Output path `destination.x`

Output type `float`

Output value original value converted to float number

115445.49 Destination Y coordinate.

Output path `destination.y`

Output type `float`

Output value original value converted to float number

5.18 Wing section

Note: Russian version

`FlightSectionParser` is responsible for parsing `Wing` section. This section contains a list of defined air flights.

Each line contains an ID of a single air flight. ID consists of regiment code or default squadron prefix, squadron index and flight index.

Section example:

```
[Wing]
  r0100
  1GvIAP12
  1GvIAP13
```

Output example:

```
{
    'flights': [
        "r0100",
        "1GvIAP12",
        "1GvIAP13",
    ],
}
```

The output of the parser is a `dict` with `flights` item. It contains a list of strings, where a each line represents a single flight ID.

5.19 Flight info section

Note: Russian version

`FlightInfoSectionParser` is responsible for parsing sections which provide information about aircrafts in a single flight. That information includes general data about all aircrafts and it can include data about individual aircrafts.

The output of the parser is a `dict` with a `FLIGHT_ID_info` item, where `FLIGHT_ID` is ID of the flight which is listed in *Wing section*. The item contains a dictionary with information about flight.

5.19.1 Section names

Sections which describe flights use flight IDs as their names. Flight ID consists of code name and two digits at the right side, e.g. `3GvIAP10`.

Code name is a code name of a regiment or a default code of a particular air force. For example, `3GvIAP` is a name of regiment of VVS RKKA and `r01` is default prefix for that air force (see the list of air forces).

Two digits in the flight ID mean squadron and flight indexes respectively. Both of them use zero-based numbering, so `00` means 1st squadron, 1st flight. There can be up to 4 squadrons in a regiment with up to 4 flights in a squadron. Code of the 4th flight in the 4th squadron will be `33`.

Parser's output contains air force, regiment, squadron and flight number:

Output path `air_force`

Output type air force constant

Output path `regiment`

Output type class Regiment

Output value `Regiment` object or `None`

Output path `squadron_index`

Output type `int`

Output value original value converted to integer number

Output path `flight_index`

Output type `int`

Output value original value converted to integer number

5.19.2 General information

Section example:

```
[3GvIAP10]
  Planes 3
  OnlyAI 1
  Parachute 0
  Skill 1
  Class air.A_20C
  Fuel 100
  weapons default
```

Output example:

```
{
    '3GvIAP10_info': {
        'id': '3GvIAP10',
        'air_force': AirForces.vvs_rkka,
        'regiment': <Regiment '3GvIAP'>,
        'squadron_index': 1,
        'flight_index': 0,
        'code': 'A_20C',
        'count': 3,
        'weapons': 'default',
        'fuel': 100,
```

```
        'ai_only': True,
        'with_parachutes': False,
        'aircrafts': [
            {
                'index': 0,
                'has_markings': True,
                'skill': Skills.average,
            },
            {
                'index': 1,
                'has_markings': True,
                'skill': Skills.average,
            },
            {
                'index': 2,
                'has_markings': True,
                'skill': Skills.average,
            },
        ],
    },
}
```

Description:

`Planes` Number of planes in flight. Maximal value is 4.

Input presence always present

Output path `count`

Output type `int`

Output value original value converted to integer number

`OnlyAI` Tells whether users cannot join flight.

Input presence present only if turned off

Output path `ai_only`

Output type `bool`

Output value `True` if `1`, `False` otherwise

Output default `False`

`Parachute` Tells whether crew members of all planes in flight have parachutes.

Input presence present only if turned off

Output path `with_parachutes`

Output type `bool`

Output value `True` if `1`, `False` otherwise

Output default `True`

`Skill` Skill level for all planes in flight.

Input presence present only if all aircrafts in flight have same level of skills

Output path `aircrafts[i].skill`, where `i` is aircraft index. Skills are applied to every aircraft individually (see section below)

Output type complex skills constant

Class Aircraft code name with `air.` prefix.

Input presence always present

Output path `code`

Output type `str`

Output value original string value without `air.` prefix

Fuel Fullness of fuel (in percents).

Input presence always present

Output path `fuel`

Output type `int`

Output value original value converted to integer number

weapons Weapons code name.

Input presence always present

Output path `weapons`

Output type `str`

Output value original string value

5.19.3 Individual skills

Section example:

```
[UN_NN03]
  Planes 2
  Skill0 2
  Skill1 3
  Skill2 1
  Skill3 1
  Class air.B_17G
  Fuel 100
  weapons default
```

Output example:

```
{
    'UN_NN03_info': {
        'air_force': AirForces.usn,
        'regiment': None,
        'squadron_index': 0,
        'flight_index': 3,
        'code': 'B_17G',
        'count': 2,
        'weapons': 'default',
        'fuel': 100,
        'ai_only': False,
        'with_parachutes': True,
        'aircrafts': [
            {
                'index': 0,
                'has_markings': True,
```

```
                'skill': Skills.veteran,
            },
            {
                'index': 1,
                'has_markings': True,
                'skill': Skills.ace,
            },
        ],
    },
}
```

As you can see from the previous section, flight info can contain `Skill` parameter. It defines skill level for all aircrafts in the flight. However, if you need to override skill level even for a single aircraft, `Skill` paramenter will be decomposed into 4 paramenters (even if you have less than 4 aircraft in the flight): `Skill0`, `Skill1`, `Skill2` and `Skill3`.

In our example we have 2 aircrafts in a flight with veteran (`Skill0 2`) and ace (`Skill1 3`) skill levels respectively. Other skill entries (`Skill2 1` and `Skill3 1`) have really no meaning. Their values are equal to default skill level for this flight which was set before it was overridden.

5.19.4 Other individual parameters

Section example:

```
[UN_NN02]
  Planes 1
  Skill 1
  Class air.B_17G
  Fuel 100
  weapons default
  skin0 RRG_N7-B_Damaged.bmp
  noseart0 Angry_Ox.bmp
  pilot0 fi_18.bmp
  numberOn0 0
  spawn0 0_Static
```

Output example:

```
{
    'UN_NN02_info': {
        'air_force': AirForces.usn,
        'regiment': None,
        'squadron_index': 1,
        'flight_index': 3,
        'code': 'B_17G',
        'count': 1,
        'weapons': 'default',
        'fuel': 100,
        'ai_only': False,
        'with_parachutes': True,
        'aircrafts': [
            {
                'index': 0,
                'has_markings': False,
                'skill': Skills.average,
                'aircraft_skin': 'RRG_N7-B_Damaged.bmp',
                'pilot_skin': 'fi_18.bmp',
```

```
                'nose_art': 'Angry_Ox.bmp',
                'spawn_object': '0_Static',
            },
        ],
    },
}
```

As you can see from the previous examples, parsed individual parameters for are stored in `aircrafts` list. Each element of this list is a dictionary with information about a single aircraft.

Aircraft index is accessed by `index` key. Index is a number in range 0-3.

We have discussed individual skills already: skill level is accessed by `skill` key.

Section with information about flight may contain some extra individual parameters which are suffixed by index of the aircraft they are related to:

`skinX` Name of custom skin for aircraft with index `X`.

Input presence present only if non-default skin was selected

Output path `aircraft_skin`

Output type `str`

Output value original string value

`noseartX` Name of used nose art for aircraft with index `X`.

Input presence present only if nose art was selected

Output path `nose_art`

Output type `str`

Output value original string value

`pilotX` Name of custom skin for crew members of aircraft with index `X`.

Input presence present only if non-default skin was selected

Output path `pilot_skin`

Output type `str`

Output value original string value

`numberOnX` Tells whether markings are present for aircraft with index `X`.

Input presence present only if turned off

Output path `has_markings`

Output type `bool`

Output value `True` if `1`, `False` otherwise

Output default `True`

`spawnX` ID of static object which is used for spawning aircraft with index `X`.

Input presence present only if spawn object was set

Output path `spawn_object`

Output type `str`

Output value original string value

5.20 Flight route section

Note: Russian version

`FlightRouteSectionParser` is responsible for parsing sections which provide information about route of a single flight.

Route consists of separate points. Each point is defined on a single line. Lines which start with `TRIGGERS` keyword indicate options for a point which was defined in the previous line.

The output of the parser is a dictionary with a single item. It is accessible by `FLIGHT_ID_route` key, where `FLIGHT_ID` is an ID of the flight which is listed in *Wing section*. The value is a list of dictionaries, where each dictionary represents a single point of route.

Section example:

```
[3GvIAP01_Way]
  TAKEOFF 193373.53 99288.17 0 0 &0
  TRIGGERS 0 10 20 0
  NORMFLY_401 98616.72 78629.31 500.00 300.00 &0 F2
  TRIGGERS 1 1 25 5 500
  NORMFLY 63028.34 42772.13 500.00 300.00 r0100 1 &0
  GATTACK 99737.30 79106.06 500.00 300.00 0_Chief 0 &0
  GATTACK 74338.61 29746.57 500.00 300.00 4_Static 0 &0
  GATTACK 82387.92 51163.75 500.00 300.00 0_Rocket 0 &0
  LANDING_104 185304.27 54570.12 0 0 &1
```

Output example:

```
{
    'flight_route_3GvIAP01': [
        FlightRouteTakeoffPoint(
            type=RoutePointTypes.takeoff_normal,
            pos=Point3D(193373.53, 99288.17, 0.0),
            speed=0.0,
            formation=None,
            radio_silence=False,
            delay=10,
            spacing=20,
        ),
        FlightRoutePatrolPoint(
            type=RoutePointTypes.patrol_triangle,
            pos=Point3D(98616.72, 78629.31, 500.00),
            speed=300.00,
            formation=Formations.echelon_right,
            radio_silence=False,
            patrol_cycles=1,
            patrol_timeout=1,
            pattern_angle=25,
            pattern_side_size=5,
            pattern_altitude_difference=500,
        ),
        FlightRouteAttackPoint(
            type=RoutePointTypes.air_attack,
            pos=Point3D(63028.34, 42772.13, 500.00),
            speed=300.00,
            formation=None,
```

```
            radio_silence=False,
            target_id='r0100',
            target_route_point=1,
        ),
        FlightRouteAttackPoint(
            type=RoutePointTypes.ground_attack,
            pos=Point3D(99737.30, 79106.06, 500.00),
            speed=300.00,
            formation=None,
            radio_silence=False,
            target_id='0_Chief',
            target_route_point=0,
        ),
        FlightRouteAttackPoint(
            type=RoutePointTypes.ground_attack,
            pos=Point3D(74338.61, 29746.57, 500.00),
            speed=300.00,
            formation=None,
            radio_silence=False,
            target_id='4_Static',
            target_route_point=0,
        ),
        FlightRouteAttackPoint(
            type=RoutePointTypes.ground_attack,
            pos=Point3D(82387.92, 51163.75, 500.00),
            speed=300.00,
            formation=None,
            radio_silence=False,
            target_id='0_Rocket',
            target_route_point=0,
        ),
        FlightRoutePoint(
            type=RoutePointTypes.landing_straight,
            pos=Point3D(185304.27, 54570.12, 0.00),
            speed=0.00,
            formation=None,
            radio_silence=True,
        ),
    ]
}
```

There are 4 different types of route points. Each of them has several subtypes. All of them are described as types of route points.

Each point has type, X, Y, and Z coordinates and speed. They also tell about radio silence and can have information about air formation.

- *Take-off*
- *Normal flight*
- *Attack*
- *Landing*

5.20.1 Take-off

Take-off includes taxiing and instant take-off. Aircrafts in take-off can be aligned as `normal`, `pair` or `inline`. The latter two work off as runway take-off; i.e. planes take-off in the direction of the next waypoint.

You can also set the distance between planes on the ground. You can also delay the take-off.

If you set normal takeoff, plane position will be snapped to runway as usual if the waypoint is less than 1250 m away from the runway. However, flight will respect any delay that was set.

You can also specify all of those parameters for carrier take-off, but all except the time delay will be ignored.

Definition example:

```
TAKEOFF_003 80156.47 47263.58 0 0 &0
TRIGGERS 0 2 20 0
```

Output example:

```
FlightRouteTakeoffPoint(
    type=RoutePointTypes.takeoff_in_line,
    pos=Point3D(80156.47, 47263.58, 0.0),
    speed=0.0,
    formation=None,
    radio_silence=False,
    delay=2,
    spacing=20,
)
```

Take-off points are defined by `FlightRouteTakeoffPoint`.

Let's examine defined lines:

TAKEOFF_003 Type of route point (inline take-off).

Output path `type`

Output type complex constant route point types

80156.47 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

47263.58 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

0 Z coordinate.

Output path `pos.z`

Output type `float`

Output value original value converted to float number

0 Speed.

Output path `speed`

Output type `float`

Output value original value converted to float number

&0 Tells whether radio silence is enabled for this route point.

Output path `radio_silence`

Output type `bool`

Output value `True` if `&1`, `False` otherwise

Note: `TRIGGERS` line is not present for normal take-off

TRIGGERS Tells that this line contains additional options for previous one.

0 Is not used for take-off.

2 Time delay (in minutes)

Output path `delay`

Output type `int`

Output value original value converted to integer number

20 Distance between aircrafts (in meters).

Output path `spacing`

Output type `int`

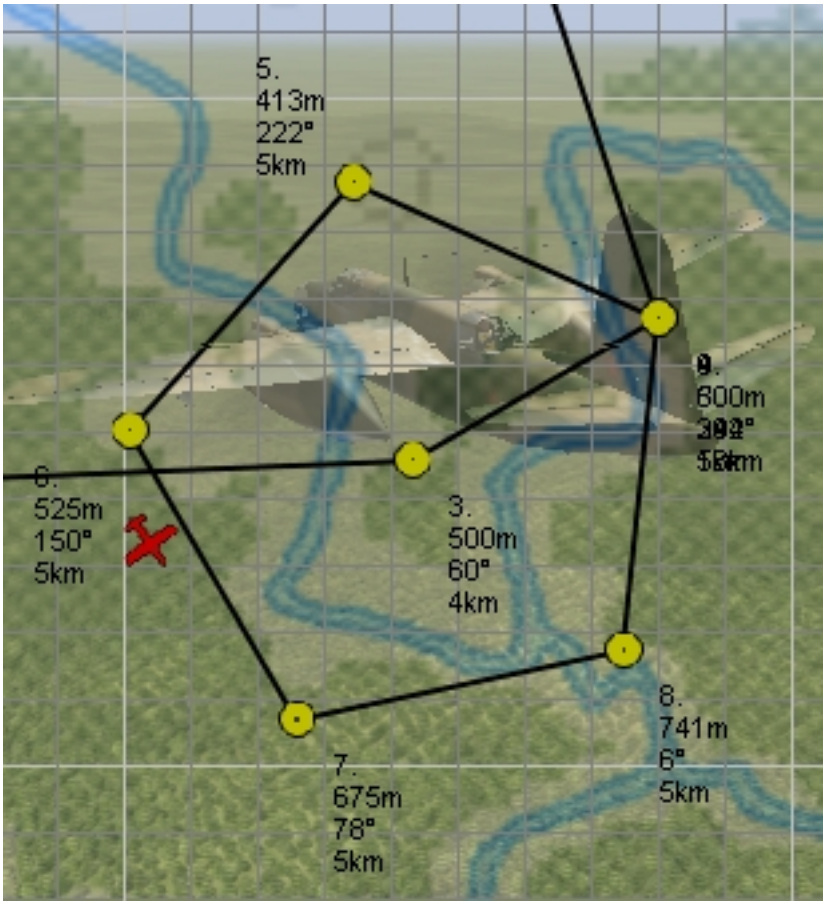
Output value original value converted to integer number

0 Is not used for take-off.

5.20.2 Normal flight

Normal flight mode includes `cruising`, `patrolling`, and `artillery spotter`.

Patrolling will establish circling movement in a particular pattern (triangle, square, etc.). You can adjust orientation of the pattern (direction of first waypoint in the pattern), side size (in km) and altitude difference from waypoint to waypoint (climbing or descending pattern).



If number of cycles or timer are set, they will tell AI when to exit the pattern and continue with subsequent waypoints. They work as OR logic, so whichever comes first will make the AI exit the cycle. Zero value for either of the two parameters means that this trigger is ignored.

Waypoints with type `artillery spotter` have such parameters as: number of cycles, timer, direction and side size. However, they do not have any effect.

Definition example:

```
NORMFLY_401 98616.72 78629.31 500.00 300.00 &0 F2
TRIGGERS 1 1 25 5 500
```

Output example:

```
FlightRoutePatrolPoint(
    type=RoutePointTypes.patrol_triangle,
    pos=Point3D(98616.72, 98616.72, 500.00),
```

```
    speed=300.00,
    formation=Formations.echelon_right,
    radio_silence=False,
    patrol_cycles=1,
    patrol_timeout=1,
    pattern_angle=25,
    pattern_side_size=5,
    pattern_altitude_difference=500,
)
```

Patrol points are defined by `FlightRoutePatrolPoint`. In other cases (normal flight and artillery spotter) `FlightRoutePoint` is used.

Let's examine defined lines:

`NORMFLY_401` Type of route point (patrolling using triangle pattern).

Output path `type`

Output type complex constant route point types

`98616.72` X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

`98616.72` Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`500.00` Z coordinate.

Output path `pos.z`

Output type `float`

Output value original value converted to float number

`300.00` Speed.

Output path `speed`

Output type `float`

Output value original value converted to float number

`&0` Tells whether radio silence is enabled for this route point.

Output path `radio_silence`

Output type `bool`

Output value `True` if `&1`, `False` otherwise

`F2` Type of air formation (echelon right).

Output path `formation`

Output type complex constant air formations or `None`

Note: `TRIGGERS` line is not present for normal flight

`TRIGGERS` Tells that this line contains additional options for previous one.

`1`[1] Number of cycles to repeat.

Output path `patrol_cycles`

Output type `int`

Output value original value converted to integer number

`2`[1] Timeout (in minutes).

Output path `patrol_timeout`

Output type `int`

Output value original value converted to integer number

`25`[1] Angle of pattern (in degrees).

Output path `pattern_angle`

Output type `int`

Output value original value converted to integer number

`5`[1] Size of pattern's side (in km).

Output path `pattern_side_size`

Output type `int`

Output value original value converted to integer number

`500`[1] Altitude difference (in meters).

Output path `pattern_altitude_difference`

Output type `int`

Output value original value converted to integer number

5.20.3 Attack

There are 2 kinds of way points which tell AI to attack other units: attack ground units and attack air units. Both of them have same parameters, but different types. Former one is defined as `GATTACK` and the latter as `NORMFLY`.

Note: Yes, waypoints which tell AI to attack air units has type `NORMFLY`, just if it is a normal flight point. This is misleading, so route point types define this type as `X_AIR_ATTACK`, where `X` tells that this is a fake type.

A target is any destroyable object: aircraft, moving vehicle, artillery, rocket, static object, etc.

Definition example:

```
NORMFLY 63028.34 42772.13 500.00 300.00 r0100 1 &0
GATTACK 99737.30 79106.06 500.00 300.00 0_Chief 0 &0
```

Output example:

[1] For patrol points only.

```
[
    FlightRouteAttackPoint(
        type=RoutePointTypes.air_attack,
        pos=Point3D(63028.34, 42772.13, 500.00),
        speed=300.00,
        formation=None,
        radio_silence=False,
        target_id='r0100',
        target_route_point=1,
    ),
    FlightRouteAttackPoint(
        type=RoutePointTypes.ground_attack,
        pos=Point3D(99737.30, 79106.06, 500.00),
        speed=300.00,
        formation=None,
        radio_silence=False,
        target_id='0_Chief',
        target_route_point=0,
    ),
]
```

Attack points are defined by `FlightRouteAttackPoint`.

Let's examine the second line:

GATTACK Type of route point (attack ground unit).

Output path `type`

Output type complex constant route point types

99737.30 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

79106.06 Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

500.00 Z coordinate.

Output path `pos.z`

Output type `float`

Output value original value converted to float number

300.00 Speed.

Output path `speed`

Output type `float`

Output value original value converted to float number

0_Chief ID of the unit to attack.

Output path `target_id`

Output type `str`

Output value original string value

0 Waypoint number of the unit to attack (not relevant for static objects).

Output path `target_route_point`

Output type `int`

Output value original value converted to integer number

&0 Tells whether radio silence is enabled for this route point.

Output path `radio_silence`

Output type `bool`

Output value `True` if `&1`, `False` otherwise

5.20.4 Landing

For landing you can choose one of the 5 landing patterns:

- right;
- left;
- short right;
- short left;
- straight in.

`Left` pattern is the default pattern used in versions of the game before 4.12. The `straight in` landing is rather tricky to get correct and can cause planes to crash into each other. You can set several flights with different pattern to land on the same airfield. AI seems to handle this fairly well, but there are no guarantees that they will not collide. All settings are ignored if the flight is landing on a carrier (i.e. they use default `left` pattern).

Definition example:

```
LANDING_104 185304.27 54570.12 0 0 &1
```

Output example:

```
FlightRoutePoint(
    type=RoutePointTypes.landing_straight,
    pos=Point3D(185304.27, 54570.12, 0.00),
    speed=0.00,
    formation=None,
    radio_silence=True,
)
```

Landing points do not have special parameters and they are defined by `FlightRoutePoint`.

Description:

LANDING_104 Type of route point (landing using `straight` pattern).

Output path `type`

Output type complex constant route point types

185304.27 X coordinate.

Output path `pos.x`

Output type `float`

Output value original value converted to float number

`54570.12` Y coordinate.

Output path `pos.y`

Output type `float`

Output value original value converted to float number

`0` Z coordinate.

Output path `pos.z`

Output type `float`

Output value original value converted to float number

`0` Speed.

Output path `speed`

Output type `float`

Output value original value converted to float number

`&1` Tells whether radio silence is enabled for this route point.

Output path `radio_silence`

Output type `bool`

Output value `True` if `&1`, `False` otherwise

Footnotes:

CHAPTER 6

Demo

Note: Russian version

Structure and contents of parser's result are hard to desribe in simple words. That's why project's demo page was created.

Demo address: http://il2horusteam.github.io/il2fb-mission-parser/

You can use the demo to check whether the parser can process some mission file. Bug reports are created and reported automatically if some error occurs. Reports contain an error message and a link to the mission which caused the error. You can see current open issues here.

Project's repository:

https://github.com/IL2HorusTeam/il2fb-mission-parser

CHAPTER 7

Indices and tables

- genindex
- modindex
- search