pydcs Documentation

Release 0.8

Peinthor Rene

March 26, 2016

Contents

pydcs is a Python(3) framework for creating digital combat simulator missions.

Contents:

CHAPTER 1

Quickstart

Creating a mission is fairly simple, the most important class in this respect is *dcs.mission.Mission*. This class contains all information for running a dcs mission. It is a .zip file that contains several lua data structures and other resources like briefing images, voice overs and other lua scripts.

```
m = dcs.Mission()
m.save('mission.miz')
```

This code is enough to create a mission file without any unit groups in the Caucasus(default) terrain.

To add a A-10C flight group starting from Batumi airport use the following snippet:

```
fg = m.flight_group_from_airport(m.country("USA"), "A-10C Flight Group",
                                 dcs.planes.A-10C, m.terrain.batumi(), group_size=2)
fg.units[0].set_player()
```

This adds a A-10C flight with 2 planes starting cold from a free parking slot. In the next line it also sets the first unit of the flight as player. For more options when adding a flight see *dcs.mission.Mission.flight_group_from_airport()*.

CHAPTER 2

Random mission script

pydcs delivers a small random mission creator script. This script was indented to show features of the framework and is also a little testbed.

It is invoked by the commandline with the following arguments:

```
usage: random_mission.py [-h] [-a {A-10C,Su-25T,M-2000C,Ka-50,MiG-21Bis}]
                         [-p PLAYERCOUNT] [-s {inflight,runway,warm,cold}]
                         [-t {main,CAS,CAP,refuel}]
                         [-d {random,day,night,dusk,dawn,noon}]
                         [-w {dynamic,dyncyclone,dynanti,dynone,clear}] [-u]
                         [--show-stats] [-o OUTPUT]

Random DCS mission generator

optional arguments:
  -h, --help            show this help message and exit
  -a {A-10C,Su-25T,M-2000C,Ka-50,MiG-21Bis}, --aircrafttype {A-10C,Su-25T,M-2000C,Ka-50,MiG-21Bis}
                        Player aircraft type
  -p PLAYERCOUNT, --playercount PLAYERCOUNT
  -s {inflight,runway,warm,cold}, --start {inflight,runway,warm,cold}
  -t {main,CAS,CAP,refuel}, --missiontype {main,CAS,CAP,refuel}
  -d {random,day,night,dusk,dawn,noon}, --daytime {random,day,night,dusk,dawn,noon}
  -w {dynamic,dyncyclone,dynanti,dynone,clear}, --weather {dynamic,dyncyclone,dynanti,dynone,clear}
  -u, --unhide          Show enemy pre mission
  --show-stats          Show generated missions stats
  -o OUTPUT, --output OUTPUT
                        Name and path of the generated mission
```

CHAPTER 3

dcs.mission module

The mission module is the entry point to all pydcs functions.

class dcs.mission.**StartType**
Bases: enum.Enum

Enum class for start types.

Cold = None
Coldstart from ramp.

Warm = None
Warmstart from ramp.

Runway = None
Start from runway.

static from_string(*s: str*)
Returns the StartType enum for a string value.

["cold", "warm", "runway"]

Parameters **s** – string representation of the starttype

Returns the correct StartType.

class dcs.mission.**Mission**(*terrain: typing.Union=None*)
Bases: object

This class represents the whole dcs .miz file.

A .miz file is a zip file containing all needed files to run a mission. example.miz:

- mission
- options
- warehouses
- i10n
  - DEFAULT
  - dictionary
    - mapResource
    - [localized resource files, .wav, .jpg, ...]

Parameters **terrain** – the used terrain for this mission.

season_from_start_time = None
If set to True the mission season will be set by the value of `Mission.start_time`

load_file(*filename: str*)
Load a mission file (.miz) file, replacing all current data.

Parameters **filename** – path to the mission(.miz) file.

Returns True if everything loaded correctly

Return type bool

Raises `RuntimeError` – if an unknown value is encountered

sortie_text() → str
Returns the mission sortie text.

Returns the mission sortie text

set_sortie_text(*text: str*)
Sets the mission sortie text.

Parameters **text** – text to set.

description_text() → str
Returns the mission description text.

Returns the mission description text

set_description_text(*text: str*)
Sets the mission descsription text.

Parameters **text** – text to set.

description_bluetask_text() → str
Returns the blue task description text.

Returns the blue task description text

set_description_bluetask_text(*text: str*)
Sets the red coalitions task description text.

Parameters **text** – text to set.

description_redtask_text() → str
Returns the red task description text.

Returns the red task description text

set_description_redtask_text(*text: str*)
Sets the red coalitions task description text.

Parameters **text** – text to set.

add_picture_red(*filepath: str*) → str
Adds a new briefing picture to the red coalition.

Parameters **filepath** – path to the image, jpg or bmp.

Returns the resource key of the picture

add_picture_blue(*filepath: str*) → str
Adds a new briefing picture to the blue coalition.

Parameters **filepath** – path to the image, jpg or bmp.

Returns the resource key of the picture

next_group_id()
Get the next free group id

Returns a new group id

next_unit_id()
Get the next free unit id

Returns a new unit id

next_dict_id()
Get the next free dictionary id

Returns a new dictionary id

eplrs_for(*group: str*) → typing.Dict
Searches all vehicle eplrs using groups and writes them in a mapping

Parameters **group** – which group to look for eplrs task, ["helicopter", "plane", "vehicle"]

Returns a dict mapping groups to used eplrs id

next_eplrs(*group_type: str*) → int
Get next eplrs for the given group type.

Parameters **group_type** – one of "vehicle", "helicopter" or "plane"

Returns the next eplrs id to use

Return type int

string(*s*, *lang='DEFAULT'*)
Create a new String() object for translation

Parameters

- **s** – string for lang
- **lang** – language for s

Returns A new String() object for string s

Return type *String*

static(*name*, *_type: dcs.unittype.UnitType*) → dcs.unit.Static
Creates a plain static object to be added to a group

Parameters

- **name** – of the static object
- **_type** (StaticType) – type of the static

Returns a new static object

Return type *Static*

static_group(*country*, *name*, *_type: dcs.unittype.UnitType*, *position: dcs.mapping.Point*, *heading=0*, *hidden=False*, *dead=False*)
Add a static group with 1 static object.

Parameters

- **country** (Country) – the object belongs too
- **name** – name of the group
- **_type** – what kind of object

- **position** (dcs.mapping.Point) – where to place the object
- **heading** – of the object
- **hidden** – should the object be hidden on the map
- **dead** – should the object be rendered as dead

Returns the new static group

Return type *StaticGroup*

vehicle(*name*, *_type: dcs.unittype.VehicleType*) → dcs.unit.Vehicle
Creates a plain vehicle unit to be added to a group

Parameters

- **name** – of the vehicle
- **_type** – vehicle type

Returns a new vehicle unit.

Return type *Vehicle*

vehicle_group(*country*, *name*, *_type: dcs.unittype.VehicleType*, *position: dcs.mapping.Point*, *heading=0*, *group_size=1*, *formation=<Formation.Line: 1>*, *move_formation: dcs.point.PointAction=<PointAction.OffRoad: 'Off Road'>*) → dcs.unitgroup.VehicleGroup
Adds a new vehicle group to the given country.

Parameters **country** – which the vehicle group will belong too

Returns the new vehicle group object

Return type *VehicleGroup*

vehicle_group_platoon(*country*, *name*, *types: typing.List*, *position: dcs.mapping.Point*, *heading=0*, *formation=<Formation.Line: 1>*, *move_formation: dcs.point.PointAction=<PointAction.OffRoad: 'Off Road'>*) → dcs.unitgroup.VehicleGroup
Adds a new vehicle group to the given country and given vehicle types.

Parameters **country** – which the vehicle group will belong too

Returns the new vehicle group object

Return type *VehicleGroup*

ship(*name*, *_type: dcs.unittype.ShipType*) → dcs.unit.Ship
Creates a plain ship unit to be added to a group

Parameters

- **name** – of the ship
- **_type** – ship type

Returns a new ship unit.

Return type *Ship*

ship_group(*country*, *name*, *_type: dcs.unittype.ShipType*, *position: dcs.mapping.Point*, *heading=0*, *group_size=1*) → dcs.unitgroup.ShipGroup
Adds a ship group to the given country.

Parameters

- **country** (Country) – which the ship group will belong too
- **name** – of the ship group
- **_type** – which kind of ship to add
- **position** (dcs.mapping.Point) – where the new group will be placed
- **heading** – initial heading of the group, only used if no additional waypoints
- **group_size** – how many ships of _type

Returns the new ship group object

Return type *ShipGroup*

plane_group(*name*) → dcs.unitgroup.PlaneGroup

This creates a plain plane group without any units or starting points.

This method is a advanced interface method not intended for simple use. For adding full featured plane group see

- *flight_group()*
- *flight_group_inflight()*
- *flight_group_from_airport()*

Parameters **name** – Group name

Returns A new *dcs.unitgroup.PlaneGroup*

Return type *PlaneGroup*

plane(*name*, *_type: dcs.planes.PlaneType*, *country: dcs.country.Country*)

Creates a new plane unit.

This method is a advanced interface method not intended for simple usage. For adding full a featured plane group see

- *flight_group()*
- *flight_group_inflight()*
- *flight_group_from_airport()*

Parameters

- **name** – unit name
- **_type** – type of the plane
- **country** (Country) – the plane belongs, needed for default liveries

Returns A new *dcs.unit.Plane*

Return type *Plane*

helicopter(*name*, *_type: dcs.helicopters.HelicopterType*, *country: dcs.country.Country*)

Creates a new helicopter unit.

This method is a advanced interface method not intended for simple usage. For adding full a featured helicopter group see

- *flight_group()*
- *flight_group_inflight()*

- *flight_group_from_airport()*

Parameters

- **name** – unit name
- **_type** – type of the helicopter
- **country** (Country) – the helicopter belongs, needed for default liveries

Returns A new *dcs.unit.Helicopter*

Return type *Helicopter*

aircraft(*name*, *_type: dcs.unittype.FlyingType*, *country: dcs.country.Country*) → typing.Union

Creates a new plane or helicopter unit, depending on the _type.

This method is a advanced interface method not intended for simple usage. For adding full a featured plane/helicopter group see

- *flight_group()*
- *flight_group_inflight()*
- *flight_group_from_airport()*

Parameters

- **name** – unit name
- **_type** – type of the aircraft
- **country** (Country) – the aircraft belongs, needed for default liveries

Returns A new *dcs.unit.Plane* or *dcs.unit.Helicopter*

Return type *Helicopter*

helicopter_group(*name*) → dcs.unitgroup.HelicopterGroup

Creates a plain helicopter group without any units or starting points.

This method is a advanced interface method not intended for simple usage. For adding a full featured helicopter group see

- *flight_group()*
- *flight_group_inflight()*
- *flight_group_from_airport()*

Parameters **name** – Group name

Returns A new *dcs.unitgroup.HelicopterGroup*

Return type *HelicopterGroup*

flight_group_inflight(*country*, *name: str*, *aircraft_type: dcs.unittype.FlyingType*, *position: dcs.mapping.Point*, *altitude: int*, *speed=None*, *maintask: typing.Union=None*, *group_size: int=1*) → typing.Union

Add a new Plane/Helicopter group inflight.

The type of the resulting group depends on the given aircraft_type.

Parameters

- **country** (Country) – the new group will belong to

- **name** – of the new group
- **aircraft_type** (FlyingType) – type of all units in the group
- **position** (dcs.mapping.Point) – where the new group will be placed
- **altitude** – of the new group
- **speed** – of the new group, if none a default will be picked
- **maintask** (MainTask) – if none the default task for the aircraft_type wil be used
- **group_size** – number of units in the group(maximum 4 or 1 for certain types)

Returns a new *dcs.unitgroup.PlaneGroup* or *dcs.unitgroup.HelicopterGroup*

Return type *FlyingGroup*

flight_group_from_airport(*country: dcs.country.Country*, *name*, *aircraft_type: dcs.unittype.FlyingType*, *airport: dcs.terrain.terrain.Airport*, *maintask: dcs.task.MainTask=None*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *group_size=1*, *parking_slots: typing.List=None*) → typing.Union

Add a new Plane/Helicopter group at the given airport.

Runway, warm/cold start depends on the given start_type.

Parameters

- **country** (Country) – Country object the plane group belongs to
- **name** – Name of the aircraft group
- **maintask** (MainTask) – Task of the aircraft group
- **aircraft_type** (FlyingType) – FlyingType class that describes the aircraft_type
- **airport** (Airport) – Airport object on which to spawn the helicopter
- **start_type** (StartType) – Start from runway, cold or warm parking position
- **parking_slots** – List of parking slots to use for aircrafts
- **group_size** – number of units in the group(maximum 4 or 1 for certain types)

Returns a new *dcs.unitgroup.PlaneGroup* or *dcs.unitgroup.HelicopterGroup*

Return type *FlyingGroup*

flight_group_from_unit(*country: dcs.country.Country*, *name*, *aircraft_type: dcs.unittype.FlyingType*, *carrier_unit: typing.Union*, *maintask: dcs.task.MainTask=None*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *group_size=1*) → typing.Union

Add a new Plane/Helicopter group at the given FARP or carrier unit.

Parameters

- **country** (Country) – Country object the plane group belongs to
- **name** – Name of the aircraft group
- **maintask** (MainTask) – Task of the aircraft group
- **aircraft_type** (FlyingType) – FlyingType class that describes the aircraft_type
- **carrier_unit** (Unit) – Group(Ship, FARP) on which to spawn

- **start_type** (StartType) – Start from runway, cold or warm parking position, ignored for now
- **group_size** – number of units in the group(maximum 4 or 1 for certain types)

Returns a new *dcs.unitgroup.PlaneGroup* or *dcs.unitgroup.HelicopterGroup*

Return type *FlyingGroup*

flight_group(*country: dcs.country.Country*, *name: str*, *aircraft_type: dcs.unittype.FlyingType*, *airport: typing.Union*, *position: typing.Union*, *altitude=3000*, *speed=500*, *maintask: typing.Union=None*, *start_type: dcs.mission.StartType=<StartType.Runway: 3>*, *group_size=1*) → dcs.unitgroup.FlyingGroup

This is wrapper around flight_group_inflight and flight_group_from_airport.

Depending on the airport parameter a flight group will added inflight or on an airport.

Parameters

- **country** (Country) – Country object the plane group belongs to
- **name** – Name of the aircraft group
- **aircraft_type** (FlyingType) – FlyingType class that describes the aircraft_type
- **airport** (Airport) – Airport object on which to spawn the helicopter
- **position** (dcs.mapping.Point) – where the new group will be placed, if inflight
- **altitude** – initial altitude of the group if inflight
- **speed** – initial speed of the group if inflight
- **maintask** (MainTask) – Task of the aircraft group
- **start_type** (StartType) – Start from runway, cold or warm parking position
- **group_size** – number of units in the group(maximum 4 or 1 for certain types)

Returns a new *dcs.unitgroup.PlaneGroup* or *dcs.unitgroup.HelicopterGroup*

Return type *FlyingGroup*

awacs_flight(*country: dcs.country.Country*, *name: str*, *plane_type: dcs.planes.PlaneType*, *airport: typing.Union*, *position: dcs.mapping.Point*, *race_distance=30000*, *heading=90*, *altitude=4500*, *speed=550*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *frequency=140*) → dcs.unitgroup.PlaneGroup

Add an AWACS flight group.

This is simple way to add an AWACS flight group to your mission. It needs an initial orbit point, race distance and heading from this point.

If an airport is given the AWACS flight will start from there otherwise, it will placed 2 km in front of the reference position.

Parameters

- **country** (Country) – Country object the awacs group belongs to
- **name** – of the AWACS flight
- **plane_type** (*PlaneType*) – AWACS plane type. e.g E_3A
- **airport** (Airport) – starting airport, use None if you want it to spawn inflight
- **position** (dcs.mapping.Point) – reference point for the race-track
- **race_distance** – distance for the race-track pattern

- **heading** – direction from the referene position
- **altitude** – of the AWACS race-track
- **speed** – of the AWACS flight
- **start_type** (StartType) – of the flight if starts from airport
- **frequency** – VHF-AM frequencey in mhz

Returns the created AWACS flight group

Return type *PlaneGroup*

refuel_flight(*country*, *name: str*, *plane_type: dcs.planes.PlaneType*, *airport: typing.Union*, *position: dcs.mapping.Point*, *race_distance=30000*, *heading=90*, *altitude=4500*, *speed=407*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *frequency=140*, *tacanchannel='10X'*) → dcs.unitgroup.PlaneGroup

Add an refuel flight group.

This is simple way to add an refuel flight group to your mission. It needs an initial orbit point, race distance and heading from this point.

If an airport is given the refuel flight will start from there otherwise, it will placed 2 km in front of the reference position.

Parameters

- **country** (Country) – Country object the awacs group belongs to
- **name** – of the refuel flight
- **plane_type** (*PlaneType*) – refuel plane type. e.g KC_135
- **airport** (Airport) – starting airport, use None if you want it to spawn inflight
- **position** (dcs.mapping.Point) – reference point for the race-track
- **race_distance** – distance for the race-track pattern
- **heading** – direction from the referene position
- **altitude** – of the refuel race-track
- **speed** – of the refuel flight
- **start_type** (StartType) – of the flight if starts from airport
- **frequency** – VHF-AM frequencey in mhz
- **tacanchannel** – if the PlaneType supports tacan this channel will be set.

Returns the created refuel flight group

Return type *PlaneGroup*

escort_flight(*country*, *name: str*, *escort_type: dcs.planes.PlaneType*, *airport: typing.Union*, *group_to_escort: dcs.unitgroup.FlyingGroup*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *group_size=2*)

Add an escort flight group to the mission.

An escort flight is a flight group that will use the *dcs.task.EscortTaskAction* to escort another flight group.

If no airport is given, the escort flight will spawn near the group to escort.

Parameters

- **country** (Country) – the escort flight belongs too

- **name** – of the flight group
- **escort_type** (*PlaneType*) – PlaneType for the escort task
- **airport** (Airport) – starting airport, use None if you want it to spawn inflight
- **group_to_escort** – id of the group to escort
- **start_type** (StartType) – of the flight if starts from airport
- **group_size** – how many planes should be in the escort flight

Returns the created escort group

Return type *PlaneGroup*

patrol_flight(*country*, *name: str*, *patrol_type: dcs.planes.PlaneType*, *airport: typing.Union*, *pos1*, *pos2*, *start_type: dcs.mission.StartType=<StartType.Cold: 1>*, *speed=600*, *altitude=4000*, *max_engage_distance=60000*, *group_size=2*)

Add an patrol flight group to the mission.

A patrol flight is a flight group that will fly a orbit between 2 given points and will engage any incoming air threats within max_engage_distance.

If no airport is given, the patrol flight will spawn near the first patrol point(pos1).

Parameters

- **country** (Country) – the flight belongs too
- **name** – name of the patrol flight
- **patrol_type** (*PlaneType*) – PlaneType for the patrol flight
- **airport** (Airport) – starting airport, use None if you want it to spawn inflight
- **pos1** (dcs.mapping.Point) – first orbit waypoint
- **pos2** (dcs.mapping.Point) – second orbit waypoint
- **start_type** (StartType) – of the flight if starts from airport
- **speed** – orbit speed
- **altitude** – initial altitude and orbit altitude
- **max_engage_distance** – the distance in KM the patrol flight will respond to enemy threats
- **group_size** – how many planes should be in the flight group

Returns the created patrol group

Return type *PlaneGroup*

country(*name*)

Returns the country object for the mission by the given string

Parameters **name** – string representation of the country

Returns the object of the country, None if not found.

Return type *Country*

find_group(*group_name*, *search='exact'*) → typing.Union

Searches a group with the given name.

Parameters

- **group_name** – part or exact name of the group

- **search** – search mode to use
  - 'exact': whole name must match
  - 'match': part of the name must match

Returns the group found, otherwise None

Return type *Group*

is_red(*country: dcs.country.Country*) → bool
Checks if the given country object is part o the red coalition.

Parameters **country** (Country) – object to check

Returns True if it is part of the red coalition, else False.

Return type bool

is_blue(*country: dcs.country.Country*) → bool
Checks if the given country object is part o the blue coalition.

Parameters **country** (Country) – object to check

Returns True if it is part of the blue coalition, else False.

Return type bool

stats() → typing.Dict
Gather some mission stats.

This method counts up the different group types and used units and returns them as easy to print dict.

Returns dict containing various group and unit counts.

print_stats(*d*)
Print the given mission stats to standard output.

Parameters **d** – stats dict to print, *dcs.mission.Mission.stats()*

reload()
Reloads the current loaded file

Raises RuntimeError – if there is currently no file loaded.

save(*filename=None*)
Save the current Mission object to the given file.

Parameters

- **filename** – filepath to save the Mission object
- **show_stats** (*bool*) – if True print mission stats to standard out.

dict()

class dcs.mission.**MapResource**(*mission: dcs.mission.Mission*)
Bases: object

MapResource is responsibly to manage all additional mission resource files.

Mission resource files are briefing images, lua scripts, sounds files.

Parameters **mission** (Mission) – the mission this MapResource belongs too, needed for dictionary ids

load_from_dict(*_dict*, *zipf: zipfile.ZipFile*, *lang='DEFAULT'*)

add_resource_file(*filepath*, *lang='DEFAULT'*, *key=None*)
Adds a file to the mission resource depot.

Parameters

- **filepath** – path to the file to add
- **lang** – language this file belongs too.
- **key** – should None, needed for loading

Returns resource key to use in scripts

store(*zipf: zipfile.ZipFile*, *lang='DEFAULT'*)

class dcs.mission.**Options**
Bases: object

Should be a representation for the mission options file might be removed in the future.

load_from_dict(*d*)

CHAPTER 4

dcs.task module

This module holds all Tasks that are possible to specifiy in dcs.

There are 2 type of tasks, a MainTask and a Task action.

- MainTasks are the flight groups main objective like *CAS*, *CAP*, *SEAD*, ...
- Task actions on the otherhand are specific tasks within a MainTask, these cover things like *AttackGroup*, Orbit, *Follow*, *Escort*, ...

Also options and commands are task actions.

class dcs.task.**Modulation**
Bases: enum.Enum

Enum for VHF frequency bands

AM = None
AM frequency band

FM = None
FM frequency band

class dcs.task.**Task**(*_id*)
Bases: object

Base class for task actions.

classmethod create_from_dict(*d*)

dict()

class dcs.task.**ControlledTask**(*task: dcs.task.Task=None*)
Bases: *dcs.task.Task*

A ControlledTask is a task action with start and stop conditions.

ControlledTask is a wrapper around a normal task action that has special methods to add start/stop conditions.

Parameters **task** – to wrap

Id = 'ControlledTask'

start_after_time(*time: int*)
Start the wrapped task after time seconds.

Parameters **time** – start after x seconds.

start_if_user_flag(*user_flag*, *value: bool*)
Start the wrapped task if user_flag has value.

Parameters

- **user_flag** – id of the userflag
- **value** – bool value of the flag

start_probability(*probability: int*)
Chance that the wrapped task will actually start.

Parameters **probability** – start chance in %

start_if_lua_predicate(*lua_predicate: str*)
Start wrapped task if lua condition is true.

Parameters **lua_predicate** – lua condition as string

stop_after_time(*time: int*)
Stop the wrapped task after time seconds.

Parameters **time** – start after x seconds.

stop_if_user_flag(*user_flag*, *value: bool*)
Stop the wrapped task if user_flag has value.

Parameters

- **user_flag** – id of the userflag
- **value** – bool value of the flag

stop_if_lua_predicate(*lua_predicate: str*)
Stop wrapped task if lua condition is true.

Parameters **lua_predicate** – lua condition as string

stop_after_duration(*duration: int*)
Stop task after duration seconds.

Parameters **duration** – in seconds

class dcs.task.**WeaponType**
Bases: enum.Enum

class dcs.task.**TargetType**
Bases: type

id = None

class dcs.task.**Targets**
Bases: object

class All
Bases: object

id = 'All'

class Air
Bases: object

id = 'Air'

class Planes
Bases: object

id = 'Planes'

class **Fighters**
Bases: object

id = 'Fighters'

class Targets.All.Air.Planes.**Bombers**
Bases: object

id = 'Bombers'

class Targets.All.Air.**Helicopters**
Bases: object

id = 'Helicopters'

class Targets.All.**GroundUnits**
Bases: object

id = 'Ground Units'

class **Infantry**
Bases: object

id = 'Infantry'

class Targets.All.GroundUnits.**Fortifications**
Bases: object

id = 'Fortifications'

class Targets.All.GroundUnits.**GroundVehicles**
Bases: object

id = 'Ground vehicles'

class **ArmoredVehicles**
Bases: object

id = 'Armored vehicles'

class **Tanks**
Bases: object

id = 'Tanks'

class Targets.All.GroundUnits.GroundVehicles.ArmoredVehicles.**IFV**
Bases: object

id = 'IFV'

class Targets.All.GroundUnits.GroundVehicles.ArmoredVehicles.**APC**
Bases: object

id = 'APC'

class Targets.All.GroundUnits.GroundVehicles.**Artillery**
Bases: object

id = 'Artillery'

class Targets.All.GroundUnits.GroundVehicles.**UnarmedVehicles**
Bases: object

id = 'Unarmed vehicles'

class Targets.All.GroundUnits.**AirDefence**
Bases: object

id = 'Air Defence'

class AAA
Bases: object

id = 'AAA'

class SAMRelated
Bases: object

id = 'SAM related'

class SRSAM
Bases: object

id = 'SR SAM'

class Targets.All.GroundUnits.AirDefence.AAA.SAMRelated.**MRSAM**
Bases: object

id = 'MR SAM'

class Targets.All.GroundUnits.AirDefence.AAA.SAMRelated.**LRSAM**
Bases: object

id = 'LR SAM'

class Targets.All.**Naval**
Bases: object

id = 'Naval'

class Ships
Bases: object

id = 'Ships'

class ArmedShips
Bases: object

id = 'Armed ships'

class HeavyArmedShips
Bases: object

id = 'Heavy armed ships'

class AircraftCarriers
Bases: object

id = 'Aircraft Carriers'

class Targets.All.Naval.Ships.ArmedShips.HeavyArmedShips.**Cruisers**
Bases: object

id = 'Cruisers'

class Targets.All.Naval.Ships.ArmedShips.HeavyArmedShips.**Destroyers**
Bases: object

id = 'Destroyers'

class Targets.All.Naval.Ships.ArmedShips.HeavyArmedShips.**Frigates**
Bases: object

id = 'Frigates'

class Targets.All.Naval.Ships.ArmedShips.HeavyArmedShips.**Corvettes**
Bases: object

id = 'Corvettes'

class Targets.All.Naval.Ships.ArmedShips.**LightArmedShips**
Bases: object

id = 'Light armed ships'

class Targets.All.Naval.Ships.**UnarmedShips**
Bases: object

id = 'Unarmed ships'

class Targets.All.Naval.**Submarines**
Bases: object

id = 'Submarines'

class dcs.task.**NoTask**
Bases: *dcs.task.Task*

Id = 'NoTask'

class dcs.task.**AttackGroup**(*group_id=0*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*)
Bases: *dcs.task.Task*

Attack group task action
Parameters

- **group_id** – group id to attack
- **weapon_type** – *WeaponType* to use for the attack

Id = 'AttackGroup'

class dcs.task.**AttackUnit**(*unit_id=0*, *attack_limit: typing.Union=None*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*, *group_attack=False*)
Bases: *dcs.task.Task*

Attack unit task action
Parameters

- **unit_id** – unit id to attack
- **attack_limit** – how much attack runs
- **weapon_type** – *WeaponType* to use for the attack
- **group_attack** – indicate if the unit must be attacked by all aircrafts in the group.

Id = 'AttackUnit'

class dcs.task.**AttackMapObject**(*position: dcs.mapping.Point=Point(0, 0)*, *attack_limit: typing.Union=None*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*, *group_attack=False*)
Bases: *dcs.task.Task*

Id = 'AttackMapObject'

class dcs.task.**AntishipStrikeTaskAction**
Bases: *dcs.task.Task*

Id = 'EngageTargets'

Key = 'AntiShip'

dict()

class dcs.task.**CASTaskAction**
Bases: *dcs.task.Task*

Id = 'EngageTargets'

Key = 'CAS'

dict()

class dcs.task.**SEADTaskAction**
Bases: *dcs.task.Task*

Id = 'EngageTargets'

Key = 'SEAD'

dict()

class dcs.task.**CAPTaskAction**
Bases: *dcs.task.Task*

Id = 'EngageTargets'

Key = 'CAP'

dict()

class dcs.task.**FighterSweepTaskAction**
Bases: *dcs.task.Task*

Id = 'EngageTargets'

Key = 'FighterSweep'

dict()

class dcs.task.**EscortTaskAction**(*group_id=None*, *engagement_max_dist=60000*, *lastwpt=None*, *targets: typing.List=None*, *position: typing.Dict=None*)
Bases: *dcs.task.Task*

Id = 'Escort'

class dcs.task.**Expend**
Bases: enum.Enum

class dcs.task.**Bombing**(*position: dcs.mapping.Point=Point(0, 0)*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*, *expend: dcs.task.Expend=<Expend.Auto: 'Auto'>*, *attack_qty=1*, *group_attack=False*, *direction: typing.Union=None*, *altitude: typing.Union=None*)
Bases: *dcs.task.Task*

Id = 'Bombing'

class dcs.task.**BombingRunway**(*airport_id: int=0*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*, *expend: dcs.task.Expend=<Expend.Auto: 'Auto'>*, *attack_qty=1*, *group_attack=False*, *direction: typing.Union=None*, *altitude: typing.Union=None*)
Bases: *dcs.task.Task*

Id = 'BombingRunway'

class dcs.task.**EngageTargets**(*max_distance=None*, *targets: typing.List=None*)
Bases: *dcs.task.Task*

Id = 'EngageTargets'

class dcs.task.**EngageTargetsInZone**(*position: dcs.mapping.Point=Point(0, 0)*, *radius=5000*, *targets: typing.List=None*)
Bases: *dcs.task.Task*

Id = 'EngageTargetsInZone'

class dcs.task.**EngageGroup**(*group_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageGroup'

class dcs.task.**EngageUnit**(*unit_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageUnit'

class dcs.task.**FireAtPoint**(*position: dcs.mapping.Point=Point(0, 0)*, *rounds=None*, *radius=0*)
Bases: *dcs.task.Task*

Id = 'FireAtPoint'

class dcs.task.**Hold**
Bases: *dcs.task.Task*

Unit will hold current position

Id = 'Hold'

class dcs.task.**AWACSTaskAction**
Bases: *dcs.task.Task*

Id = 'AWACS'

class dcs.task.**RefuelingTaskAction**
Bases: *dcs.task.Task*

Assigns the aircraft group to refuel at the nearest tanker aircraft.

Id = 'Refueling'

class dcs.task.**Tanker**
Bases: *dcs.task.Task*

Assigns the aircraft to act as an Airborne tanker.

Id = 'Tanker'

class dcs.task.**OrbitAction**(*altitude=4000*, *speed=600*, *pattern: dcs.task.OrbitPattern=<OrbitPattern.RaceTrack: 'Race-Track'>*)
Bases: *dcs.task.Task*

Id = 'Orbit'

class OrbitPattern
Bases: enum.Enum

class dcs.task.**Follow**(*groupid=None*, *position: dcs.mapping.Point=Point(-200, 0)*, *altitude_difference=-200*, *last_wpt=None*)
Bases: *dcs.task.Task*

Id = 'Follow'

class dcs.task.**Aerobatics**
Bases: *dcs.task.Task*

Id = 'Aerobatics'

class dcs.task.**Designation**
Bases: enum.Enum

class dcs.task.**FAC**(*callsign: int=1*, *designation: dcs.task.Designation=<Designation.Auto: 'Auto'>*, *frequency: int=30*, *modulation: dcs.task.Modulation=<Modulation.FM: 1>*, *number: int=1*)
Bases: *dcs.task.Task*

Id = 'FAC'

class dcs.task.**FACEngageGroup**(*group_id: int=0*, *visible=False*, *weapon_type: dcs.task.WeaponType=<WeaponType.Auto: 1073741822>*, *priority: int=0*, *callsign: int=1*, *designation: dcs.task.Designation=<Designation.Auto: 'Auto'>*, *frequency: int=30*, *modulation: dcs.task.Modulation=<Modulation.FM: 1>*, *datalink=True*, *number: int=1*)
Bases: *dcs.task.Task*

Id = 'FAC_EngageGroup'

class dcs.task.**Land**(*position: dcs.mapping.Point=Point(0, 0)*, *duration: int=None*)
Bases: *dcs.task.Task*

Helicopter landing task.

If added to a helicopter group, the group will land at the given coordinates for the given duration.

Parameters

- **position** – *dcs.mapping.Point* where to land
- **duration** – how long the helicopter should stay on ground in seconds.

Id = 'Land'

class dcs.task.**Embarking**(*position: dcs.mapping.Point=Point(0, 0)*, *groupids: typing.List=None*, *distribution: typing.Dict=None*, *duration: int=None*)
Bases: *dcs.task.Task*

Pickup task for helicopters.

Helicopter group will land at the given coordinates and will pickup the given groups.

Parameters

- **position** – *dcs.mapping.Point* where to land and pickup
- **groupids** – list of groups to pickup
- **distribution** – dictionary with heli unit to groups to pickup mapping {heliunit: [grp1, grp2], ..}
- **duration** – how long the helicopter should stay on ground and wait in seconds.

Id = 'Embarking'

class dcs.task.**EmbarkToTransport**(*position: dcs.mapping.Point=Point(0, 0)*, *zone_radius=200*, *concrete_unitid=None*)
Bases: *dcs.task.Task*

Task for ground units that will get picked up by a helicopter

Parameters

- **position** – *dcs.mapping.Point* where to wait to get picked up.
- **zone_radius** – radius around the point where the group will embark.
- **concrete_unitid** – if specified the group will embark to exaclty this unit.

Id = 'EmbarkToTransport'

class dcs.task.**DisembarkFromTransport**(*position: dcs.mapping.Point=Point(0, 0)*, *zone_radius=200*)

Bases: *dcs.task.Task*

Task for ground units that will disembark a transport helicopter

Parameters

- **position** – *dcs.mapping.Point* where the group will disembark
- **zone_radius** – radius around the point where the group will disembark.

Id = 'DisembarkFromTransport'

class dcs.task.**CargoTransportation**(*groupid=None*, *zoneid=None*)

Bases: *dcs.task.Task*

Task for Cargo transportation.

Parameters

- **groupid** – cargo group id
- **zoneid** – zone id to transport to??

Id = 'CargoTransportation'

class dcs.task.**WrappedAction**

Bases: *dcs.task.Task*

Id = 'WrappedAction'

class dcs.task.**EPLRS**(*group_id=1*)

Bases: *dcs.task.WrappedAction*

Key = 'EPLRS'

eplrs

class dcs.task.**ActivateBeaconCommand**(*channel=10*, *modechannel='X'*, *callsign='TKR'*, *bearing=True*)

Bases: *dcs.task.WrappedAction*

Key = 'ActivateBeacon'

static calc_tacan_frequency(*mode_channel*, *channel*, *aa=True*)

class dcs.task.**RunScript**(*script: str=''*)

Bases: *dcs.task.WrappedAction*

Runs a given script string

Parameters **script** – to be executed

Key = 'Script'

class dcs.task.**RunScriptFile**(*resourcekey: str=''*)

Bases: *dcs.task.WrappedAction*

Runs a script attached to the mission

Parameters **resourcekey** – resource key to the script file, see *dcs.mission.MapResource*

Key = 'ScriptFile'

class dcs.task.**TransmitMessage**(*soundfile_reskey: typing.Union=None*, *subtitle_resstring: typing.Union=None*, *loop=False*, *subtitle_duration=5*)

Bases: *dcs.task.WrappedAction*

Transmits a given sound file over the current radio channel.

Parameters

- **soundfile_reskey** – resource key to the sound file to transmit, see *dcs.mission.MapResource*
- **subtitle_resstring** – string resource key to subtitle displayed, see dcs.mission.Mission.translation
- **loop** – True or False if the sound file should be looped.
- **subtitle_duration** – how long the subtitle should be displayed in seconds.

Key = 'TransmitMessage'

class dcs.task.**StopTransmission**
Bases: *dcs.task.WrappedAction*

Stops any *dcs.task.TransmitMessage* task currently ongoing.

Key = 'StopTransmission'

class dcs.task.**SetFrequencyCommand**(*frequency=133*, *modulation: dcs.task.Modulation=<Modulation.AM: 0>*)
Bases: *dcs.task.WrappedAction*

Set the groups radio frequency.

Parameters

- **frequency** – frequency band in mhz.
- **modulation** – AM or FM, see *dcs.task.Modulation*

Key = 'SetFrequency'

class dcs.task.**SwitchWaypoint**(*from_waypoint=1*, *to_waypoint=2*)
Bases: *dcs.task.WrappedAction*

Switch to a different waypoint.

Parameters

- **from_waypoint** – from which waypoint to switch.??
- **to_waypoint** – new current waypoint

Key = 'SwitchWaypoint'

class dcs.task.**SetInvisibleCommand**(*value=True*)
Bases: *dcs.task.WrappedAction*

Key = 'SetInvisible'

class dcs.task.**SetImmortalCommand**(*value=True*)
Bases: *dcs.task.WrappedAction*

Key = 'SetImmortal'

class dcs.task.**MainTask**
Bases: object

name = None

sub_tasks = []

perform_task = []

map = {'SEAD': <class 'dcs.task.SEAD'>, 'AFAC': <class 'dcs.task.AFAC'>, 'Nothing': <class 'dcs.task.Nothing'>, 'Refu

class dcs.task.**Nothing**
Bases: *dcs.task.MainTask*

id = 15

name = 'Nothing'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Aerobatics'>]

class dcs.task.**AFAC**
Bases: *dcs.task.MainTask*

id = 16

name = 'AFAC'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.AttackGroup'>, <class 'dcs.task

class dcs.task.**AWACS**
Bases: *dcs.task.MainTask*

id = 14

name = 'AWACS'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.RefuelingTaskAction'>]

perform_task = [<class 'dcs.task.AWACSTaskAction'>]

class dcs.task.**AntishipStrike**
Bases: *dcs.task.MainTask*

id = 30

name = 'Antiship Strike'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.AttackGroup'>, <class 'dcs.task

perform_task = [<class 'dcs.task.AntishipStrikeTaskAction'>]

class dcs.task.**CAS**
Bases: *dcs.task.MainTask*

id = 31

name = 'CAS'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.AttackGroup'>, <class 'dcs.task

perform_task = [<class 'dcs.task.CASTaskAction'>]

class EnrouteTasks
Bases: object

class EngageGroup(*group_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageGroup'

class CAS.EnrouteTasks.**EngageTargetsInZone**(*position: dcs.mapping.Point=Point(0, 0)*, *radius=5000*, *targets: typing.List=None*)
Bases: *dcs.task.Task*

Id = 'EngageTargetsInZone'

class CAS.EnrouteTasks.**EngageUnit**(*unit_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageUnit'

class dcs.task.**CAP**
Bases: *dcs.task.MainTask*

id = 11

name = 'CAP'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Aerobatics'>]

perform_task = [<class 'dcs.task.CAPTaskAction'>]

class EnrouteTasks
Bases: object

class EngageTargets(*max_distance=None*, *targets: typing.List=None*)
Bases: *dcs.task.Task*

Id = 'EngageTargets'

class CAP.EnrouteTasks.**EngageTargetsInZone**(*position: dcs.mapping.Point=Point(0, 0)*, *radius=5000*, *targets: typing.List=None*)
Bases: *dcs.task.Task*

Id = 'EngageTargetsInZone'

class CAP.EnrouteTasks.**EngageGroup**(*group_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageGroup'

class CAP.EnrouteTasks.**EngageUnit**(*unit_id=0*, *visible=False*)
Bases: *dcs.task.Task*

Id = 'EngageUnit'

class dcs.task.**Escort**
Bases: *dcs.task.MainTask*

id = 18

name = 'Escort'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.EscortTaskAction'>]

perform_task = [<class 'dcs.task.EscortTaskAction'>]

class dcs.task.**FighterSweep**
Bases: *dcs.task.MainTask*

id = 19

name = 'Fighter Sweep'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Aerobatics'>]

perform_task = [<class 'dcs.task.FighterSweepTaskAction'>]

class dcs.task.**GroundAttack**
Bases: *dcs.task.MainTask*

id = 32

name = 'Ground Attack'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Bombing'>, <class 'dcs.task.Att

class dcs.task.**Intercept**
Bases: *dcs.task.MainTask*

id = 10

name = 'Intercept'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.AttackGroup'>, <class 'dcs.task

class dcs.task.**PinpointStrike**
Bases: *dcs.task.MainTask*

id = 33

name = 'Pinpoint Strike'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Bombing'>, <class 'dcs.task.Att

class dcs.task.**Reconnaissance**
Bases: *dcs.task.MainTask*

id = 17

name = 'Reconnaissance'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Aerobatics'>]

perform_task = []

class dcs.task.**Refueling**
Bases: *dcs.task.MainTask*

id = 13

name = 'Refueling'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>]

perform_task = [<class 'dcs.task.Tanker'>]

class dcs.task.**RunwayAttack**
Bases: *dcs.task.MainTask*

id = 34

name = 'Ground Attack'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Bombing'>, <class 'dcs.task.Bor

perform_task = []

class dcs.task.**SEAD**
Bases: *dcs.task.MainTask*

id = 29

name = 'SEAD'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.AttackGroup'>, <class 'dcs.task

perform_task = [<class 'dcs.task.SEADTaskAction'>]

class dcs.task.**Transport**
Bases: *dcs.task.MainTask*

id = 35

name = 'Transport'

sub_tasks = [<class 'dcs.task.OrbitAction'>, <class 'dcs.task.Follow'>, <class 'dcs.task.Aerobatics'>]

perform_task = []

class dcs.task.**Option**(*value=None*)
Bases: *dcs.task.Task*

Id = 'WrappedAction'

Key = None

class dcs.task.**OptROE**(*value=0*)
Bases: *dcs.task.Option*

Sets the groups rules of engagement. Ultimately defines whether or not the AI group is allowed to attack. This option can override all other tasking. With the 1.5 patch two of the values have different names in the mission editor. However the behavior is still exactly the same as before, its just labeled slightly different. The scripting engine still uses the previous values.

Key = 0

class Values
Bases: object

WeaponFree = 0
AI will engage any enemy group it detects. Target prioritization is based based on the threat of the target.

OpenFireWeaponFree = 1
AI will engage any enemy group it detects, but will prioritize targets specified in the groups tasking.

OpenFire = 2
AI will engage only targets specified in its taskings.

ReturnFire = 3
AI will only engage threats that shoot first.

WeaponHold = 4
AI will hold fire under all circumstances.

class dcs.task.**OptReactOnThreat**(*value=0*)
Bases: *dcs.task.Option*

Defines the allowable action for an airborne group to take in response to a threat. This option can have an effect on other tasking.

Key = 1

class Values
Bases: object

NoReaction = 0
No defensive actions will take place to counter threats

PassiveDefense = 1
AI will use jammers and other countermeasures in an attempt to defeat the threat. AI will not attempt a maneuver to defeat a threat.

EvadeFire = 2
AI will react by performing defensive maneuvers against incoming threats, AI will also use passive defense.

ByPassAndEscape = 3
AI will attempt to avoid enemy threat zones all together. This includes attempting to fly above or around threats.

AllowAbortMission = 4
If a threat is deemed severe enough the AI will abort its mission and return to base.

class dcs.task.**OptDisparseUnderFire**(*value=None*)
Bases: *dcs.task.Option*

Key = 8

CHAPTER 5

dcs

5.1 dcs package

5.1.1 Subpackages

dcs.lua package

Submodules

dcs.lua.parse module

dcs.lua.parse.**loads**(*tablestr*)

dcs.lua.serialize module

dcs.lua.serialize.**dumps**(*value*, *varname=None*, *indent=None*)

Module contents

dcs.terrain package

Submodules

dcs.terrain.caucasus module

class dcs.terrain.caucasus.**Anapa**

Bases: *dcs.terrain.terrain.Airport*

id = 12

name = 'Anapa'

position = Point(-5406.2803440839, 243127.2973737)

tacan = None

frequencies = [121000000, 38400000, 250000000, 3750000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**KrasnodarCenter**
Bases: *dcs.terrain.terrain.Airport*

id = 13

name = 'Krasnodar-Center'

position = Point(11692.789495652, 367948.47230953)

tacan = None

frequencies = [122000000, 38600000, 251000000, 3800000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Novorossiysk**
Bases: *dcs.terrain.terrain.Airport*

id = 14

name = 'Novorossiysk'

position = Point(-40915.496728899, 279256.64920952)

tacan = None

frequencies = [123000000, 38800000, 252000000, 3850000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Krymsk**
Bases: *dcs.terrain.terrain.Airport*

id = 15

name = 'Krymsk'

position = Point(-6583.663574989, 294383.98405512)

tacan = None

frequencies = [124000000, 39000000, 253000000, 3900000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Maykop**
Bases: *dcs.terrain.terrain.Airport*

id = 16

name = 'Maykop'

position = Point(-26441.347360305, 458040.61422532)

tacan = None

frequencies = [125000000, 39200000, 254000000, 3950000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Gelendzhik**
Bases: *dcs.terrain.terrain.Airport*

id = 17

name = 'Gelendzhik'

position = Point(-50392.648146355, 298387.43849386)

tacan = None

frequencies = [126000000, 39400000, 255000000, 4000000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Sochi**
Bases: *dcs.terrain.terrain.Airport*

id = 18

name = 'Sochi'

position = Point(-164474.73482633, 462236.21834688)

tacan = None

frequencies = [127000000, 39600000, 256000000, 4050000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**KrasnodarPashkovsky**
Bases: *dcs.terrain.terrain.Airport*

id = 19

name = 'Krasnodar-Pashkovsky'

position = Point(7674.038444859, 385029.5736699)

tacan = None

frequencies = [128000000, 39800000, 257000000, 4100000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Sukhumi**
Bases: *dcs.terrain.terrain.Airport*

id = 20

name = 'Sukhumi'

position = Point(-220531.73642658, 564387.05872916)

tacan = None

frequencies = [129000000, 40000000, 258000000, 4150000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Gudauta**
Bases: *dcs.terrain.terrain.Airport*

id = 21

name = 'Gudauta'

position = Point(-196974.19851241, 516290.23098695)

tacan = None

frequencies = [130000000, 40200000, 259000000, 4200000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Batumi**
Bases: *dcs.terrain.terrain.Airport*

id = 22

name = 'Batumi'

position = Point(-355692.3067714, 617269.96285781)

tacan = '16X'

frequencies = [131000000, 40400000, 260000000, 4250000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Senaki**
Bases: *dcs.terrain.terrain.Airport*

id = 23

name = 'Senaki'

position = Point(-281713.83114196, 647369.87369832)

tacan = '31X'

frequencies = [132000000, 40600000, 261000000, 4300000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Kobuleti**
Bases: *dcs.terrain.terrain.Airport*

id = 24

name = 'Kobuleti'

position = Point(-317948.32727306, 635639.37385346)

tacan = '67X'

frequencies = [133000000, 40800000, 262000000, 4350000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Kutaisi**
Bases: *dcs.terrain.terrain.Airport*

id = 25

name = 'Kutaisi'

position = Point(-284889.06283057, 683853.75717885)

tacan = None

frequencies = [134000000, 41000000, 263000000, 4400000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Mineralnye**
Bases: *dcs.terrain.terrain.Airport*

id = 26

name = 'Mineralnye'

position = Point(-51251.551717591, 705718.47981263)

tacan = None

frequencies = [135000000, 41200000, 264000000, 4450000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Nalchik**
Bases: *dcs.terrain.terrain.Airport*

id = 27

name = 'Nalchik'

position = Point(-124921.90954665, 760428.0733062)

tacan = None

frequencies = [136000000, 41400000, 265000000, 4500000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Mozdok**
Bases: *dcs.terrain.terrain.Airport*

id = 28

name = 'Mozdok'

position = Point(-83454.571428571, 834453.14285714)

tacan = None

frequencies = [137000000, 41600000, 266000000, 4550000]

unit_zones = []

civilian = False

class dcs.terrain.caucasus.**Lochini**
Bases: *dcs.terrain.terrain.Airport*

id = 29

name = 'Lochini'

position = Point(-315478.57142857, 896538.85714286)

tacan = None

frequencies = [138000000, 41800000, 267000000, 4600000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Soganlug**
Bases: *dcs.terrain.terrain.Airport*

id = 30

name = 'Soganlug'

position = Point(-317838.57142857, 895424.57142858)

tacan = None

frequencies = [139000000, 42000000, 268000000, 4650000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Vaziani**
Bases: *dcs.terrain.terrain.Airport*

id = 31

name = 'Vaziani'

position = Point(-319069.063, 903150.625)

tacan = None

frequencies = [140000000, 42200000, 269000000, 4700000]

unit_zones = []

civilian = False

class dcs.terrain.caucasus.**Beslan**
Bases: *dcs.terrain.terrain.Airport*

id = 32

name = 'Beslan'

position = Point(-148810.84954665, 843756.7533062)

tacan = None

frequencies = [141000000, 42400000, 270000000, 4750000]

unit_zones = []

civilian = True

class dcs.terrain.caucasus.**Caucasus**
Bases: *dcs.terrain.terrain.Terrain*

center = {'lat': 45.12945, 'long': 34.26527}

bounds = Rectangle(380000, -560000, -600000, 1130000)

map_view_default = <dcs.terrain.terrain.MapView object>

soganlug() → dcs.terrain.terrain.Airport

senaki() → dcs.terrain.terrain.Airport

sochi() → dcs.terrain.terrain.Airport

batumi() → dcs.terrain.terrain.Airport

nalchik() → dcs.terrain.terrain.Airport

beslan() → dcs.terrain.terrain.Airport

mozdok() → dcs.terrain.terrain.Airport

anapa() → dcs.terrain.terrain.Airport

krasnodar_center() → dcs.terrain.terrain.Airport

krasnodar_pashkovsky() → dcs.terrain.terrain.Airport

novorossiysk() → dcs.terrain.terrain.Airport

krymsk() → dcs.terrain.terrain.Airport

maykop() → dcs.terrain.terrain.Airport

gelendzhik() → dcs.terrain.terrain.Airport

mineralnye() → dcs.terrain.terrain.Airport

gudauta() → dcs.terrain.terrain.Airport

vaziani() → dcs.terrain.terrain.Airport

lochini() → dcs.terrain.terrain.Airport

kobuleti() → dcs.terrain.terrain.Airport

kutaisi() → dcs.terrain.terrain.Airport

sukhumi() → dcs.terrain.terrain.Airport

default_red_airports() → typing.List

default_blue_airports() → typing.List

dcs.terrain.nevada module

class dcs.terrain.nevada.**Creech**

Bases: *dcs.terrain.terrain.Airport*

id = 1

name = 'Creech'

position = Point(-359732, -74970.9)

tacan = None

frequencies = [122000000, 38600000, 251000000]

unit_zones = []

civilian = False

slot_version = 2

class dcs.terrain.nevada.**Groom**
Bases: *dcs.terrain.terrain.Airport*

id = 2

name = 'Groom'

position = Point(-288694, -87414.2)

tacan = None

frequencies = [123000000, 38800000, 252000000]

unit_zones = []

civilian = False

slot_version = 2

class dcs.terrain.nevada.**McCarran**
Bases: *dcs.terrain.terrain.Airport*

id = 3

name = 'McCarran'

position = Point(-416083, -27121.1)

tacan = None

frequencies = [124000000, 39000000, 253000000]

unit_zones = []

civilian = False

slot_version = 2

class dcs.terrain.nevada.**Nellis**
Bases: *dcs.terrain.terrain.Airport*

id = 4

name = 'Nellis'

position = Point(-397971, -17639.5)

tacan = None

frequencies = [125000000, 39200000, 254000000]

unit_zones = []

civilian = False

slot_version = 2

class dcs.terrain.nevada.**Nevada**
Bases: *dcs.terrain.terrain.Terrain*

center = {'lat': 39.81806, 'long': -114.73333}

bounds = Rectangle(-497177.65625, -329334.875, -166934.953125, 209836.890625)

map_view_default = <dcs.terrain.terrain.MapView object>

creech() → dcs.terrain.terrain.Airport

groom() → dcs.terrain.terrain.Airport

mccarran() → dcs.terrain.terrain.Airport

nellis() → dcs.terrain.terrain.Airport

dcs.terrain.terrain module

class dcs.terrain.terrain.**ParkingSlot**(*crossroad_idx*, *position: dcs.mapping.Point*, *large=False*, *slot_name=None*, *heli=False*, *airplanes=True*, *length=None*, *width=None*, *height=None*, *shelter=False*)

Bases: object

class dcs.terrain.terrain.**Runway**(*heading*, *ils=None*, *leftright=0*)

Bases: object

class dcs.terrain.terrain.**Airport**

Bases: object

id = None

name = None

position = None

tacan = None

frequencies = []

unit_zones = []

civilian = True

slot_version = 1

load_from_dict(*d*)

set_blue()

set_red()

set_neutral()

set_coalition(*side*)

is_red()

is_blue()

random_unit_zone() → dcs.mapping.Rectangle

free_parking_slots(*aircraft_type: dcs.unittype.FlyingType*) → typing.List

free_parking_slot(*aircraft_type: dcs.unittype.FlyingType*) → typing.Union

dict()

class dcs.terrain.terrain.**MapView**(*center: dcs.mapping.Point*, *zoom=1000000*)

Bases: object

load_from_dict(*d*)

dict()

class dcs.terrain.terrain.**Terrain**(*name: str*)

Bases: object

bounds = None

map_view_default = None

airport_by_id(*id: int*) → dcs.terrain.terrain.Airport

airport_list() → typing.List

class dcs.terrain.terrain.**Warehouses**(*terrain: dcs.terrain.terrain.Terrain*)
Bases: object

load_dict(*data*)

Module contents

5.1.2 Submodules

5.1.3 dcs.country module

dcs.country.**find_exact**(*group_name*, *find_name*)

dcs.country.**find_match**(*group_name*, *find_name*)

class dcs.country.**Country**(*_id*, *name*)
Bases: object

callsign = {}

planes = []

helicopters = []

add_vehicle_group(*vgroup*)

add_ship_group(*sgroup*)

add_plane_group(*pgroup*)

add_helicopter_group(*hgroup*)

add_aircraft_group(*group: dcs.unitgroup.FlyingGroup*)

add_static_group(*sgroup*)

find_group(*group_name*, *search='exact'*)

find_vehicle_group(*name: str*, *search='exact'*)

find_ship_group(*name: str*, *search='exact'*)

find_plane_group(*name: str*, *search='exact'*)

find_helicopter_group(*name: str*, *search='exact'*)

find_static_group(*name: str*, *search='exact'*)

next_callsign_id()

next_callsign_category(*category*)

dict()

5.1.4 dcs.forcedoptions module

class dcs.forcedoptions.**ForcedOptions**
Bases: object

class Views
Bases: enum.Enum

class ForcedOptions.**CivilTraffic**
Bases: enum.Enum

class ForcedOptions.**GEffect**
Bases: enum.Enum

ForcedOptions.**load_from_dict**(*d*)

ForcedOptions.**dict**()

5.1.5 dcs.goals module

class dcs.goals.**GoalRule**(*predicate*)
Bases: object

dict()

class dcs.goals.**AllOfCoalitionInZone**(*coalitionlist*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_all_of_coalition_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**AllOfCoalitionOutsideZone**(*coalitionlist*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_all_of_coalition_out_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**AllOfGroupInZone**(*group*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_all_of_group_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**AllOfGroupOutsideZone**(*group*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_all_of_group_out_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**ArgumentInRange**(*argument*, *_min*, *_max*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_argument_in_range'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**BombInZone**(*typebomb*, *numbombs*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_bomb_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CargoUnhookedInZone**(*cargo*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_cargo_unhooked_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CoalitionHasAirdrome**(*coalitionlist*, *airdromelist*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_coalition_has_airdrome'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CoalitionHasHelipad**(*coalitionlist*, *helipadlist*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_coalition_has_helipad'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CockpitHighlightVisible**(*highlight_id*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_cockpit_highlight_visible'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CockpitParamEqual**(*cockpit_param*, *value_text*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_cockpit_param_equal_to'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CockpitParamEqualToAnother**(*cockpit_param*, *cockpit_param2*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_cockpit_param_is_equal_to_another'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**CockpitParamInRange**(*cockpit_param*, *_min2*, *_max2*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_cockpit_param_in_range'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagEquals**(*flag=1*, *value=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_equals'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagEqualsFlag**(*flag=1*, *flag2=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_equals_flag'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagIsFalse**(*flag=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_is_false'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagIsLess**(*flag=1*, *value=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_less'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagIsLessThanFlag**(*flag=1*, *flag2=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_less_flag'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagIsMore**(*flag=1*, *value=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_more'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**FlagIsTrue**(*flag=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_flag_is_true'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**GroupAlive**(*group*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_group_alive'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**GroupDead**(*group*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_group_dead'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**GroupLifeLess**(*group*, *percent=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_group_life_less'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**IndicationTextEqual**(*indicator_id*, *element_name*, *element_value*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_indication_txt_equal_to'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**MissionScoreHigher**(*coalitionlist*, *score=50*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_mission_score_higher'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**MissionScoreLower**(*coalitionlist*, *score=50*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_mission_score_lower'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**Or**
Bases: *dcs.goals.GoalRule*

predicate = 'or'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PartOfCoalitionInZone**(*coalitionlist*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_part_of_coalition_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PartOfCoalitionOutsideZone**(*coalitionlist*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_part_of_coalition_out_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PartOfGroupInZone**(*group*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_part_of_group_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PartOfGroupOutsideZone**(*group*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_part_of_group_out_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PlayerScoreLess**(*scores=100*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_player_score_less'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**PlayerScoreMore**(*scores=100*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_player_score_more'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**Predicate**(*text*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_predicate'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**Random**(*percent=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_random_less'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**SignalFlareInZone**(*flare_color*, *numflares*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_signal_flare_in_zone'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**TimeAfter**(*seconds=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_time_after'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**TimeBefore**(*seconds=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_time_before'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**TimeSinceFlag**(*flag=1*, *seconds=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_time_since_flag'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitAlive**(*unit*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_alive'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitAltitudeHigher**(*unit*, *altitude=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_altitude_higher'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitAltitudeLower**(*unit*, *altitude=1*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_altitude_lower'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitBankWithin**(*unit*, *min_unit_bank=-180*, *max_unit_bank=180*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_bank'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitDamaged**(*unit*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_damaged'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitDead**(*unit*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_dead'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitHeadingWithin**(*unit*, *min_unit_heading*, *max_unit_heading=360*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_heading'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitInMovingZone**(*unit*, *zone*, *zoneunit*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_in_zone_unit'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitInZone**(*unit*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_in_zone'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitLifeLess**(*unit*, *percent=10*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_life_less'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitOutsideMovingZone**(*unit*, *zone*, *zoneunit*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_out_zone_unit'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitOutsideZone**(*unit*, *zone*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_out_zone'

classmethod **create_from_dict**(*d*)

dict()

class dcs.goals.**UnitPitchWithin**(*unit*, *min_unit_pitch=-180*, *max_unit_pitch=180*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_pitch'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitSpeedHigher**(*unit*, *speed=100*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_speed_higher'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitSpeedLower**(*unit*, *speed=100*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_speed_lower'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**UnitVerticalSpeedWithin**(*unit*, *min_unit_vertical_speed=-300*, *max_unit_vertical_speed=300*)
Bases: *dcs.goals.GoalRule*

predicate = 'c_unit_vertical_speed'

classmethod create_from_dict(*d*)

dict()

class dcs.goals.**Goal**(*comment=''*, *score=100*)
Bases: object

goalrule_map = {'c_unit_heading': <class 'dcs.goals.UnitHeadingWithin'>, 'c_coalition_has_helipad': <class 'dcs.goal

load_from_dict(*data*)

conditions()

dict()

class dcs.goals.**Goals**
Bases: object

load_from_dict(*data*)

add_red(*g: dcs.goals.Goal*)

add_blue(*g: dcs.goals.Goal*)

add_offline(*g: dcs.goals.Goal*)

generate_result()

dict()

5.1.6 dcs.groundcontrol module

class dcs.groundcontrol.**GroundControl**
Bases: object

load_from_dict(*d*)

dict()

5.1.7 dcs.mapping module

dcs.mapping.**point_from_heading**(*_x*, *_y*, *heading*, *distance*)
Calculates a point from a given point, heading and distance.

Parameters

- **_x** – source point x
- **_y** – source point y
- **heading** – heading in degrees from source point
- **distance** – distance from source point

Returns returns a tuple (x, y) of the calculated point

dcs.mapping.**distance**(*x1*, *y1*, *x2*, *y2*)
Returns the distance between 2 points

Parameters

- **x1** – x coordinate of point 1
- **y1** – y coordinate of point 1
- **x2** – x coordinate of point 2
- **y2** – y coordinate of point 2

Returns distance in point units(m)

dcs.mapping.**heading_between_points**(*x1*, *y1*, *x2*, *y2*)
Returns the angle between 2 points in degrees.

Parameters

- **x1** – x coordinate of point 1
- **y1** – y coordinate of point 1
- **x2** – x coordinate of point 2
- **y2** – y coordinate of point 2

Returns angle in degrees

class dcs.mapping.**Point**(*x*, *y*)
Bases: object

point_from_heading(*heading*, *distance*)

heading_between_point(*point*)

distance_to_point(*point*)

class dcs.mapping.**Triangle**(*points: typing.Union*)
Bases: object

area()

random_point() → dcs.mapping.Point

class dcs.mapping.**Rectangle**(*top*, *left*, *bottom*, *right*)
Bases: object

static from_point(*point: dcs.mapping.Point*, *side_length*)

point_in_rect(*point: dcs.mapping.Point*)

height()

width()

center() → dcs.mapping.Point

resize(*percentage: float*)

random_point() → dcs.mapping.Point

random_distant_points(*distance*) → typing.Tuple

class dcs.mapping.**Polygon**(*points: typing.List=None*)
Bases: object

point_in_poly(*point: dcs.mapping.Point*)
Checks if the given point is within the polygon.

Parameters **point** – Point to test

Returns True if point is within the polygon else False

random_point() → dcs.mapping.Point
Returns a random point within this polygon object

Returns a random point

static is_convex(*a: dcs.mapping.Point*, *b: dcs.mapping.Point*, *c: dcs.mapping.Point*)

static in_triangle(*a*, *b*, *c*, *p*)

is_clockwise()

static get_ear(*poly*)

triangulate()

outbound_rectangle() → dcs.mapping.Rectangle

5.1.8 dcs.point module

class dcs.point.**PointAction**
Bases: enum.Enum

class dcs.point.**StaticPoint**
Bases: object

load_from_dict(*d*, *translation*)

dict()

class dcs.point.**VNav**
Bases: enum.Enum

class dcs.point.**Scale**
Bases: enum.Enum

class dcs.point.**Steer**
Bases: enum.Enum

class dcs.point.**PointProperties**(*vnav: dcs.point.VNav=<VNav.V2D: 0>, scale: dcs.point.Scale=<Scale.Enroute: 0>, steer: dcs.point.Steer=<Steer.ToTo: 2>, angle=None*)
Bases: object

load_from_dict(*d*)

dict()

class dcs.point.**MovingPoint**
Bases: *dcs.point.StaticPoint*

load_from_dict(*d*, *translation*)

find_task(*task_type*)
Searches tasks in this point for the given task class

Parameters **task_type** – task class to search, *dcs.task*

Returns task instance if found, else None

dict()

5.1.9 dcs.templates module

class dcs.templates.**VehicleTemplate**
Bases: object

class Russia
Bases: object

static sa10_site(*mission: dcs.mission.Mission, position: dcs.mapping.Point, heading, prefix=''', skill=<Skill.Average: 'Average'>*)

class VehicleTemplate.**USA**
Bases: object

static patriot_site(*mission: dcs.mission.Mission, position, heading, prefix=''', skill=<Skill.Average: 'Average'>*)

static hawk_site(*mission: dcs.mission.Mission, position, heading, prefix=''', skill=<Skill.Average: 'Average'>*)

5.1.10 dcs.translation module

class dcs.translation.**String**(*_id=''', translation=None*)
Bases: object

set(*text*, *lang='DEFAULT'*)

str(*lang='DEFAULT'*)

class dcs.translation.**Translation**(*_mission*)
Bases: object

set_string(*_id*, *string*, *lang='DEFAULT'*)

get_string(*_id*)

create_string(*s*, *lang='DEFAULT'*)

delete_string(*_id*)

languages() → [<class 'str'>]

dict(*lang='DEFAULT'*)

5.1.11 dcs.unit module

class dcs.unit.**Skill**
Bases: enum.Enum

class dcs.unit.**Unit**(*_id*, *name=None*, *type=''*)
Bases: object

load_from_dict(*d*)

clone(*_id*)

dict()

class dcs.unit.**FlyingUnit**(*_id=None*, *name=None*, *_type: dcs.unittype.FlyingType=None*, *_country=None*)
Bases: *dcs.unit.Unit*

load_from_dict(*d*)

set_parking(*parking_slot: dcs.terrain.terrain.ParkingSlot*)

load_pylon(*weapon*, *pylon=None*)

store_loadout(*filename*)

load_loadout(*filename*)

reset_loadout()

set_default_preset_channel(*freq*)

set_radio_preset()

set_player()

set_client()

dict()

class dcs.unit.**Plane**(*_id=None*, *name=None*, *_type: dcs.planes.PlaneType=<class 'dcs.planes.A_10C'>*, *_country=None*)
Bases: *dcs.unit.FlyingUnit*

class dcs.unit.**Helicopter**(*_id=None*, *name=None*, *_type: dcs.helicopters.HelicopterType=<class 'dcs.helicopters.Ka_50'>*, *_country=None*)
Bases: *dcs.unit.FlyingUnit*

load_from_dict(*d*)

dict()

class dcs.unit.**Vehicle**(*id=None*, *name=None*, *_type='Sandbox'*)
Bases: *dcs.unit.Unit*

load_from_dict(*d*)

dict()

class dcs.unit.**Ship**(*id=None*, *name=None*, *_type=None*)
Bases: *dcs.unit.Unit*

load_from_dict(*d*)

dict()

class dcs.unit.**Static**(*unit_id=None*, *name=None*, *_type: dcs.unittype.UnitType=None*)
Bases: *dcs.unit.Unit*

load_from_dict(*d*)

dict()

5.1.12 dcs.unitgroup module

class dcs.unitgroup.**Group**(*_id: int*, *name=None*)
Bases: object

class Formation
Bases: enum.Enum

Group.**load_from_dict**(*d*)

Group.**add_unit**(*unit: dcs.unit.Unit*)

Group.**add_point**(*point: dcs.point.StaticPoint*)

Group.**x**

Group.**y**

Group.**position**

Group.**formation_line**(*heading*, *distance=20*)

Group.**formation_star**(*heading*, *distance=20*)

Group.**formation_rectangle**(*heading*, *distance=20*)

Group.**formation_scattered**(*heading=0*, *max_radius=None*)

Group.**formation**(*_type=<Formation.Line: 1>*, *heading=0*)

Group.**set_skill**(*skill: dcs.unit.Skill*)

Group.**dict**()

class dcs.unitgroup.**MovingGroup**(*_id*, *name=None*, *start_time=0*)
Bases: *dcs.unitgroup.Group*

load_from_dict(*d*)

dict()

class dcs.unitgroup.**VehicleGroup**(*_id*, *name=None*, *start_time=0*)
Bases: *dcs.unitgroup.MovingGroup*

load_from_dict(*d*)

add_span(*position: dcs.mapping.Point*)

add_waypoint(*position: dcs.mapping.Point*, *move_formation: dcs.point.PointAction=<PointAction.OffRoad: 'Off Road'>*, *speed=32*) → dcs.point.MovingPoint

dict()

class dcs.unitgroup.**FlyingGroup**(*_id*, *name=None*, *start_time=0*)
Bases: *dcs.unitgroup.MovingGroup*

starts_from_airport() → bool

load_from_dict(*d*)

add_waypoint(*pos: dcs.mapping.Point*, *altitude*, *speed=600*, *name: dcs.translation.String=None*) → dcs.point.MovingPoint

add_runway_waypoint(*airport: dcs.terrain.terrain.Airport*, *runway: dcs.terrain.terrain.Runway=None*, *distance=6400*) → dcs.point.MovingPoint

Adds a waypoint parallel to the given runway heading, for start or approach.

Parameters

- **airport** – start airport object
- **runway** – runway for heading direction, if None first(default) airport runway will be used.
- **distance** – distance of the waypoint from the airport

Returns MovePoint object describing the waypoint

land_at(*airport: dcs.terrain.terrain.Airport*) → dcs.point.MovingPoint

load_task_default_loadout(*task*)

load_loadout(*name*)

load_pylon(*weapon*, *pylon=None*)

load_loadout_from_file(*filename*)

set_client()

reset_loadout()

set_frequency(*frequency*)

dict()

class dcs.unitgroup.**PlaneGroup**(*_id*, *name=None*, *start_time=0*)

Bases: *dcs.unitgroup.FlyingGroup*

add_unit(*unit: dcs.unit.Plane*)

class dcs.unitgroup.**HelicopterGroup**(*_id*, *name=None*, *start_time=0*)

Bases: *dcs.unitgroup.FlyingGroup*

add_unit(*unit: dcs.unit.Helicopter*)

class dcs.unitgroup.**ShipGroup**(*_id*, *name=None*, *start_time=0*)

Bases: *dcs.unitgroup.MovingGroup*

add_waypoint(*position: dcs.mapping.Point*, *speed=20*) → dcs.point.MovingPoint

dict()

class dcs.unitgroup.**StaticGroup**(*_id*, *name=None*)

Bases: *dcs.unitgroup.Group*

load_from_dict(*d*)

dict()

5.1.13 dcs.unittype module

class dcs.unittype.**UnitType**
Bases: object

id = None

name = None

class dcs.unittype.**VehicleType**
Bases: *dcs.unittype.UnitType*

eplrs = False

class dcs.unittype.**ShipType**
Bases: *dcs.unittype.UnitType*

helicopter_num = 0

plane_num = 0

parking = 0

class dcs.unittype.**StaticType**
Bases: *dcs.unittype.UnitType*

shape_name = None

rate = 0

category = 'Fortifications'

sea_object = False

can_cargo = False

mass = None

class dcs.unittype.**FlyingType**
Bases: *dcs.unittype.UnitType*

flyable = False

group_size_max = 4

large_parking_slot = False

helicopter = False

fuel_max = 0

max_speed = 500

height = 0

width = 0

length = 0

ammo_type = None

chaff = 0

flare = 0

charge_total = 0

chaff_charge_size = 1

flare_charge_size = 2

category = 'Air'

tacan = False

eplrs = False

radio_frequency = 251

panel_radio = None

pylons = {}

payloads = None

payload_dirs = ['C:\\Program Files\\Eagle Dynamics\\DCS World\\MissionEditor\\data\\scripts\\UnitPayloads', 'C:\\P

tasks = ['Nothing']

task_default = None

classmethod scan_payload_dir()

classmethod load_payloads()

classmethod loadout(*_task*)

classmethod loadout_by_name(*loadout_name*)

classmethod default_livery(*country_name*) → str

5.1.14 dcs.weather module

class dcs.weather.**Wind**(*direction=0*, *speed=0*)
Bases: object

dict()

class dcs.weather.**Cyclone**
Bases: object

dict()

class dcs.weather.**Weather**(*terrain*)
Bases: object

class Season
Bases: object

Summer = 1

Winter = 2

Spring = 3

Fall = 4

class Weather.**BaricSystem**
Bases: enum.Enum

Weather.**load_from_dict**(*d*)

Weather.**set_season_from_datetime**(*dt: datetime.datetime*)

Weather.**dynamic_weather**(*system: dcs.weather.BaricSystem*, *cyclones: int=1*)

Weather.**dict**()

5.1.15 Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

d

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T