Traffic Module (traf)

# Class Navdatabase

This class is in navdb.py

## \_\_init\_\_(self,subfolder)

Goal: Load waypoint, airport and fir data from the specified 'subfolder' input

Inputs:

1. subfolder(string, name of subfolder containing the navigation data files, in this case the ‘global folder’)

Outputs: -

## getwpidx(self,txt,lat=999999.,lon=999999)

Goal: Get waypoint index for the given waypoint name (txt). This makes it possible to access the data of that waypoint from the waypoint data arrays that were loaded by the \_\_init\_\_ function (see above).

Inputs:

1. txt (string, name of waypoint)
2. lat(float, latitude of waypoint, degrees, optional)
3. long(float, longitude of waypoint, degrees, optional)

Outputs:

1. index of requested waypoint

## getapidx(self,txt)

Goal: Get airport index for the given airport name (txt). This makes it possible to access the data of that airport from the airport data arrays loaded by the \_\_init\_\_ function.

Inputs:

1. txt (string, name of airport)

Outputs:

1. index of requested airport

## getinear(self,wlat,wlon,lat,lon)

Goal: Get index that is nearest to lat and lon. Used by getwpinear() and getapinear() functions, see below.

Inputs:

1. wlat(array of floats, latitude of all waypoints/airports, degrees)
2. wlon(array of floats, longitude ofsll waypoints/airports, degrees)
3. lat(float, latitude of point, degrees)
4. long(float, longitude of point, degrees)

Outputs:

1. index of nearest waypoint/airport

## getwpinear(self,lat,lon)

Goal: Find the index of the waypoint that is nearest to the specified lat lon coordinates

Inputs:

1. lat(float, latitude of point, degrees)
2. long(float, longitude of point, degrees)

Outputs:

1. index of nearest waypoint

## getapinear(self,lat,lon)

Goal: Find the index of the airport that is nearest to the specified lat lon coordinates

Inputs:

1. lat(float, latitude of point, degrees)
2. long(float, longitude of point, degrees)

Outputs:

1. index of nearest airport

## getinside(self,wlat,wlon,lat0,lat1,lon0,lon1):

Goal: Get indices inside given box. Used by getwpinside() and getapinside() functions, see below.

Inputs:

1. wlat(array of floats, latitude of all waypoints/airports, degrees)
2. wlon(array of floats, longitude of all waypoints/airports, degrees)
3. lat0(float, latitude of the bottom left coordinate, degrees)
4. lon0(float, longitude of the bottom left coordinate, degrees)
5. lat1(float, latitude of the top right coordinate, degrees)
6. lon1(float, longitude of the top right coordinate, degrees)

Outputs:

1. list containing the indexes of the waypoints/airports that are inside the specified box

## getwpinside(self,lat0,lat1,lon0,lon1):

Goal: Determine the indexes of all the way points inside box

Inputs:

1. lat0(float, latitude of the bottom left coordinate, degrees)
2. lon0(float, longitude of the bottom left coordinate, degrees)
3. lat1(float, latitude of the top right coordinate, degrees)
4. lon1(float, longitude of the top right coordinate, degrees)

Outputs:

1. list containing the indexes of the waypoints that are inside the specified box

## getapinside(self,lat0,lat1,lon0,lon1):

Goal: Determine the indexes of all the airports inside box

Inputs:

1. lat0(float, latitude of the bottom left coordinate, degrees)
2. lon0(float, longitude of the bottom left coordinate, degrees)
3. lat1(float, latitude of the top right coordinate, degrees)
4. lon1(float, longitude of the top right coordinate, degrees)

Outputs:

1. list containing the indexes of the airports that are inside the specified box

# Class Traffic

This class is in traffic.py

## \_\_init\_\_(self, navdb)

Goal: Constructor for the traffic class. It simply calls the *reset* function (see below)

Inputs:

1. navdb(object, instance of the Navdatabase class)

Outputs:-

## reset(self, navdb)

Goal: (Re)Initializes all parameters, lists and arrays related to the traffic class. It is called by this class’s *\_\_init\_\_* function (see above). All the parameters related to traffic are initialized in this function (and there are many). The parameters have been divided into aircraft specific parameters (mostly lists and arrays containing the state/mode of all aircraft in the simulation) and general parameters that apply to all aircraft (e.g. coordinates of the experiment area). Any aircraft specific array/list/value added to the *reset* function should also be added to the *create* function and the *delete* function (see below).

Note: nearly all parameters are in SI units. The only ones that are not necessarily in SI are parameters related to angles (latitude, longitude, heading etc.) which are in degrees.

Inputs:

1. navdb(object, instance of the Navdatabase class)

Outputs:-

## create(self, acid, actype, aclat, aclon, achdg, acalt, casmach)

Goal: Creates an aircraft. This function is called by the *‘CRE’* command. The inputs to this function are used to define the aircraft being created, i.e., the inputs are appended to the relevant array/list that contains the states/modes of all the aircraft in the simulation. A value for all state arrays/lists needs to be specified in the *create* function to enable vectorized calculations (even if particular a state array is not relevant to the aircraft being created). This is why any aircraft state array/list added to the *reset* function should also be added to the *create* function and the *delete* function. The performance characteristics of the aircraft are also ‘created’ based on the ‘actype’.

Inputs:

1. acid(string, call sign of aircraft being created, default=None)
2. actype(string, type of aircraft being created, used for aircraft performance settings, default=None)
   1. If type is unknown, the default Boeing 747-400 aircraft is used.
3. aclat(float, latitude of aircraft, degrees, default=None)
4. aclon(float, longitude of aircraft, degrees, default=None)
5. achdg(float, heading of aircraft, degrees, default=None)
6. acalt(float, altitude of aircraft, meters, default=None)
   1. Note: The ‘*CRE’* requires altitudes in feet or FL. It is converted to meters when send to the *create* function.
7. casmach(float, speed of the aircraft, kts/Mach number, default=None)
   1. the speed specified is the Calibrated Air Speed (CAS) and can be in kts or in Mach number

Outputs:

1. ‘True’/‘False’ (boolean)
   1. Needed to detect command syntax errors in the *process* function of the *Stack* class. If ‘True’, then there are no command syntax errors.

## delete(self, acid)

Goal: Delete the specified aircraft from all the state arrays that define an aircraft, including performance settings. Don’t forget to add any new aircraft state array to the *delete* function to ensure that it is deleted completely!

Inputs:

1. acid(string, call sign of the aircraft that should be deleted)

Outputs:

1. ‘True’/‘False’ (boolean)
   1. Needed to detect command syntax errors in the *process* function of the *Stack* class. If ‘True’, then there are no command syntax errors.

## deleteall(self)

Goal: Deletes all aircraft. Calls the *delete* function (see above) in a loop, thereby deleting aircraft from the state arrays one-by-one.

Inputs: -

Outputs: -

## update(self, simt, simdt)

**TO BE WRITTEN**

## id2idx(self, acid)

Goal: Determine the index of the specified aircraft in the arrays/lists that are used to define traffic.

Inputs:

1. acid(string, call sign of the desired aircraft)

Outputs:

1. idx(int, index of the specified aircraft in the arrays/lists that are used to define traffic)
   1. ‘-1’ if aircraft call-sign is not known

## changeTrailColor(self, color, idx)

Goal: This function is called by the *‘TRAIL’* command to change the trail color of a particular aircraft.

Inputs:

1. color(string ‘RED’/’BLUE’/’YELLOW’, desired trail color for specified aircraft)
2. idx(int, index of the specified aircraft in the arrays/lists that are used to define traffic)
   1. The *process* function of the *Stack* class uses the *id2idx* function (see above) to determine the aircraft *idx* from its call-sign.

Outputs: -

## setNoise(self, A):

Goal: This function is called by the *‘NOISE’* command to switch on/off simulation noise. There are three types of noise (turbulence, ADSB transmission noise, ADSB truncation). Standard noise settings are hard-coded in this function.

Inputs:

1. A(Boolean, switch on/off noise)
   1. ‘A’ is used to switch on the three noise components separately.

Outputs: -

## engchange(self, acid, engid)

Goal: This function is called by the ‘*ENG’* command to change the engine type of the specified aircraft.

Inputs:

1. acid(string, call sign of the desired aircraft)
2. engid(int, index of the engine id that is printed on the BlueSky console)

Outputs: -

## selhdg(self, idx, hdg)

Goal: This function is called by the *‘HDG’* command to change the heading the autopilot should fly to for the specified aircraft

Inputs:

1. idx(int, index of the specified aircraft in the arrays/lists that are used to define traffic, default=None)
2. hdg(float, desired heading of the specified aircraft, default=None)

Outputs:

1. ‘True’/‘False’ (boolean)
   1. Needed to detect command syntax errors in the *process* function of the *Stack* class. If ‘True’, then there are no command syntax errors.

## selspd(self, idx, spd)

Goal: This function is called by the *‘SPD* command to change the speed the autopilot should fly with for the specified aircraft

Inputs:

1. idx(int, index of the specified aircraft in the arrays/lists that are used to define traffic, default=None)
2. spd(float, desired speed of the specified aircraft, kts/Mach numberdefault=None)

Outputs:

1. ‘True’/‘False’ (boolean)
   1. Needed to detect command syntax errors in the *process* function of the *Stack* class. If ‘True’, then there are no command syntax errors.