Simulation Module (sim)

It should be noted that there are separate versions of the sim module for the qt and pygame GUIs of BlueSky. This document **only discusses the qt version** of the sim module.

# Class Simulation

* This class is in simulation.py
* It is a derived class. Its base class is 'QObject'. Therefore it inherits many methods that are not visible in simulation.py
* It can be used to create a sim object, which in turn contains other objects (such as the traf and stack and scnreenio).

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

Goal: Initializes the simulation object. The simulation object contains variables to control the simulation, like simt (simulation time), as well as objects, like traf and stack. Therefore, the simulation object can easily 'communicate' with the methods/functions of other classes such as Traffic and CommandStack.

Inputs:

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

Ouputs: -

## moveToThread(self, target\_thread)

Goal: In BlueSky, the simulation and the gui are on separate threads. This function specifies to which thread different parts of BlueSky should be moved to. It makes a call to the moveToThread method of the base QObject class. **Best not to modify this function if you don't know anything about multi-threaded programs!**

Inputs:

1. target\_thread(object, instance of the Thread class)

Ouputs: -

## eventTarget(self)

**TO BE WRITTEN**

## doWork(self)

Goal: This is the main method of the Simulation Class. It calls methods from CommandStack and Traffic classes (amongst other classes) in a while loop until the simulation is quit. The methods of the other classes that are called from within the doWork function are, for example, responsible for processing all stack commands and updating the states of the traffic. In this sense, this function contains the main simulation loop (not to be confused with the MainLoop function in BlueSky\_qtgl.py).

Inputs: -

Outputs: -

## stop(self)

Goal: As the name suggests, this function is called to stop the simulation. It changes the mode of the simulation to 'end' (or mode==4) and asks the screenio object to stop the visualization.

Inputs: -

Outputs: -

## start(self)

Goal: As the name suggests, this function is called to start the simulation. It changes the mode of the simulation to 'op' (or mode==1). It is called from several places, including the *doWork* function of the Simulation class (see above), and by various BlueSky commands when they are processed by the *process* function in the CommandStack class.

Inputs: -

Outputs: -

## pause(self)

Goal: As the name suggests, this function is called to pause simulation. It changes the mode of the simulation to 'op' (or mode==2). It is called when the *hold* BlueSky command is processed by the *process* function in the CommandStack class.

Inputs: -

Outputs: -

## reset(self)

Goal: As the name suggests, this function is called to reset simulation. It changes the mode of the simulation to 'init' (or mode==0). It is called when the *IC* BlueSky command is processed by the *process* function in the CommandStack class to load a new scenario file. It also resets the simulation time (simt) to 0 and resets the traffic arrays by calling the traf.reset() function in the Traffic class.

Inputs: -

Outputs: -

## fastforward(self)

Goal: As the name suggests, this function is used to speed up the simulation, i.e., a Fast Time simulation. It is called when the *FF* BlueSky command is processed by the *process* function in the CommandStack class.

Inputs: -

Outputs: -

## datafeed(self,flag)

Goal: This function activates the feeding of external data to blueSky, such as from an ADSB antenna. It is called when the *DATAFEED* BlueSky command is processed by the *process* function in the CommandStack class.

Inputs:

1. flag(string, 'ON' or 'OFF' to turn on or off the data-feed).

Outputs: -

# Class Thread

* This class is in thread.py
* It is a derived class. Its base class is 'QObject'. Therefore it inherits many methods that are not visible in thread.py

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

Goal: Initialize an instance of the Thread class, making use the init function of the base 'QObject' class. It also 'connects' the doWork method of the worker\_object to the simulation thread.

Inputs:

1. worker(object, in the case of BlueSky, the worker\_object is an instance of the Simulation class)

Outputs: -

## start(self, prio)

Goal: As the name suggests, this method starts a simulation thread with the desired priority,

Inputs:

1. prio(int, the priority of the thread that is being started -> for the sim thread, the highest priority is used)

Outputs: -

## quit(self)

**TO BE WRITTEN**

# Class ThreadManager

* This class is in screenio.py
* It is a derived class. Its base class is 'QObject'.
* To do with running multiple scenarios at the same time
* **TO BE WRITTEN**

## instance()

**TO BE WRITTEN**

## currentThreadIsActive()

**TO BE WRITTEN**

## \_\_init\_\_(self, parent=None)

**TO BE WRITTEN**

## getSenderID(self)

**TO BE WRITTEN**

## setActiveNode (self, nodeid)

**TO BE WRITTEN**

## startThread(self, worker\_object, prio=Thread.HighestPriority)

**TO BE WRITTEN**

# Class ScreenIO

* This class is in screenio.py
* It is a derived class. Its base class is 'QObject'.
* This class acts as the interface between the Gui Class and the Simulation Class, and allows the sim object to send/receive data to/from the gui object.
* To this end, it contains many methods (slots and functions). ~~These are not discussed below as the average user of BlueSky is unlikely to modify the GUI.~~
* The function *objappend* can be used to draw simple shapes (such as experiment areas) on the radar widget. Currently it is set up to draw squares and circles (used for defining experiment areas). It can also be used to draw abstract shaped (experiment areas) objects and simple lines.

**TO BE WRITTEN**

# Classes in simevents.py

* Definition of data content to be transferred between GUI and Sim ~~tasks~~ Classes
* These definitions are used on both sides of the communication
* There are many small classes in this script which derived from the *QEvent* base class.
* *It looks like some kind of state-machine setup?*
* Due to small size of the classes in this script, all these classes are described in this chapter

## Class SimStateEvent

**TO BE WRITTEN**

## Class DisplayFlagEvent

**TO BE WRITTEN**

## Class SimInfoEvent

**TO BE WRITTEN**

## Class StackTextEvent

**TO BE WRITTEN**

## Class ShowDialogEvent

**TO BE WRITTEN**

## Class RouteDataEvent

**TO BE WRITTEN**

## Class DisplayShapeEvent

**TO BE WRITTEN**

## Class ACDataEvent

**TO BE WRITTEN**

## Class AMANEvent

**TO BE WRITTEN**

## Class PanZoomEvent

**TO BE WRITTEN**

## Class SimQuitEvent

**TO BE WRITTEN**

# class SimulationManager

* This class is in simmanager.py
* It is a derived class. Its base class is ThreadManager (see above)

## \_\_init\_\_(self, navdb, parent=None)

**TO BE WRITTEN**

## addNode(self)

**TO BE WRITTEN**

## stop(self)

**TO BE WRITTEN**

## getSimObjectList(self)

**TO BE WRITTEN**

## getActiveSimTarget(self)

**TO BE WRITTEN**

## event(self, event)

**TO BE WRITTEN**