Stack Module

# Class CommandStack

* This class is in stack.py
* It contains methods that are used to read, schedule and process commands, as well a method to create a scenario file.

## \_\_init\_\_(self, sim, traf, scr)

Goal: Initializes an instance of the CommandStack object. Several dictionaries and lists are initialized. Some of these are discussed below:

* *cmddict* : This dictionary is used to specify the arguments, the argument types (e.g. txt for acid) and corresponding triggering function for each command. The command syntax are the keys in the dictionary. Note this is the new way to introduce additional commands to BlueSky. When a command is to be processed, the *process* method (see below) will first look inside *cmddict* to see if the requested command is inside this dictionary. If so, it will uses predefined logic per argument *type* (e.g. ALT, ACID) to process the command. Otherwise, it will use the explicit logic for that particular command as defined in the large *elif* tree in the *process* method.
* *cmdsynon:* This dictionary contains synonym syntax for some commands, e.g. *create* is the same as *cre.* If a user enters *create*, it will be converted to *cre* in the *process* method.
* *cmdstack:* This list is very important list. It contains all the commands, including their arguments, that have to be processed and executed at each BlueSky update cycle (in other words, in each iteration of the doWork method of the Simulation class).
* *scencmd:*  This list contains commands, including their arguments, that have been read from a scenario file
* *scentime*: This list contains the time stamps of the commands that have been read in from a scenario file. It is in the same order as the *scencmd* list.
* *extracmdmodules*: This dictionary contains the file names of ‘extra’ command processing scripts that are located outside stack.py in separate files. Each of these scripts has its own *process* function that can process specific command syntax related to:
  1. *synthetic scenarios*
  2. *ASAS (Airborne Separation Assurance system)*

Inputs:

1. sim(object, instance of the Simulation class)
2. traf(object, instance of the Traffic class)
3. scr(object, instance of the ScreenIO class)

Outputs: -

## stack(self, cmdline)

Goal: Commands are appended to the *cmdstack* list by the *stack* method. Normally, commands are added to the *cmdstack* list one-by-one. However, multiple commands can also be added to the *cmdstack* listif the commands are separated with a semicolon *';'.*

Inputs:

1. cmdline (string, a line containing the command and its arguments)

Outputs: -

## openfile(self, scenname)

Goal: This method opens and reads all the lines in a scenario file. It is called when the *IC* command is processed by the *process* method (see below). Comment lines in scenario files are ignored. The times in the scenario file are separated from the command syntax. The times are stored into the *scentime* list and the commands into the  *scencmd* list. Before the lines are read in from a scenario file, the *scnetime* and *scencmd* lists are cleared.

Inputs:

1. scenname (string, file name of the scenario-does not need to contain the file extension)

## checkfile(self, simt)

Goal: This method is called every BlueSky update cycle from the doWork method of the Simulation class. If there are commands from a scenario file that need to be processed, the *checkfile* method will send the commands that need to be executed at each moment to the *stack* method, at scenario specified simulation times.

Inputs:

1. simt(float, current simulation time)

Outputs: -

## saveic(self, fname, sim, traf)

Goal: Creates a scenario file based on the traffic situation that is currently displayed on the radar gui. The resulting scenario files will contain the following commands, if appropriate: CRE, VS, ALT, HDG, SPD, DEST, ORIG. The time for all commands will be 00:00:00.00. The scenario will be saved into the scenario folder.

Inputs:

1. fname(string, name of the desired output scenario file-does not to have any extension)
2. sim(object, instance of the Simulation class)
3. traf(object, instance of the Traffic class)

Outputs: -

## process(self, sim, traf, scr)

Goal: This method processes all the commands in the *cmdstack* list, and it is called during each BlueSky update cycle (in other words, in each iteration of the *doWork* method of the Simulation class). It should be noted that all the commands in the cmdstack are processed one-by-one during each BlueSky update cycle, and at the end of the method, the *cmdstack* list is emptied.

At first, the *process* method looks for command synonyms in the *cmdsynon* dictionary. If a synonym is found, it is replaced with the standard syntax for that command. Then the *cmddict* is checked. If a command is in the *cmddict*, then predefined logic relating the command’s argument types are used to process that command (thus the same code is used to process many different commands). If a command is not is the *cmddict,* then explict logic for each command is contained in a very large elif tree.

If the command is not in the *cmddict*, or in the large elif tree, then the additional command processing scripts specified in the *extracmdmodules* dictionary is checked. If so, the *process* function of these extra command scripts are called. The commands processed by these separate scripts are all interrelated, for example, the CD&R/ASAS or synthetic scenarios. That’s why they are located in separate files.

For each command, it is checked if the syntax is correct in terms of the number of arguments. If so, the appropriate method of the traf, sim or scr objects are called using the user specified command arguments. If a command is not located in any of these three possible locations, or if there is mistake in the command arguments, then an error message is printed on the console gui.

Inputs:

1. sim(object, instance of the Simulation class)
2. traf(object, instance of the Traffic class)
3. scr(object, instance of the ScreenIO class)

Outputs: -