EXOSIMS Run and Post Processing Automation

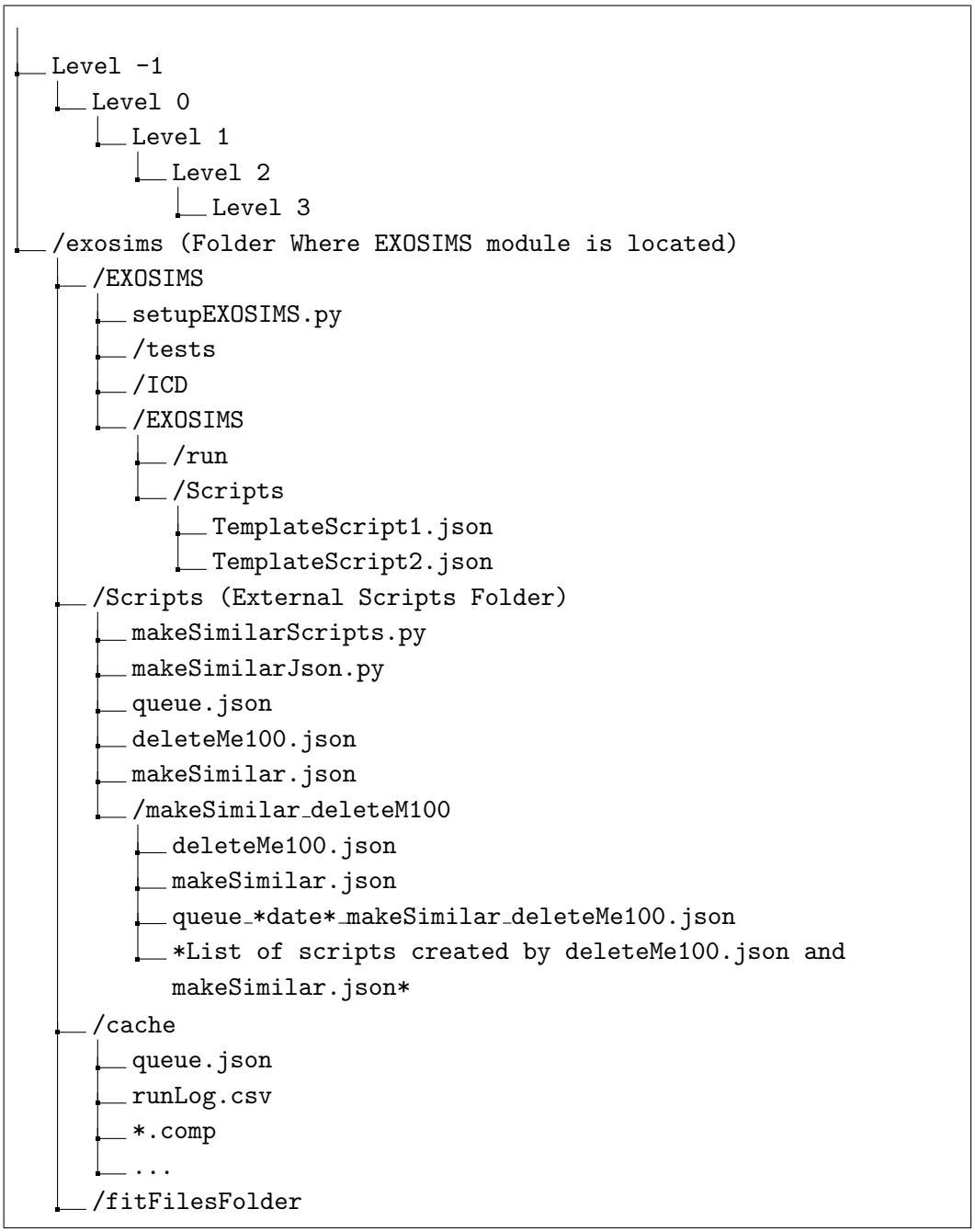
Dean Keithly

October 28, 2018

1 setupEXOSIMS.py

setupEXOSIMS.py is designed to be run from level 1 of the EXOSIMS repository (the same folder where documentation, ICD, tests, and EXOSIMS folders exist). This script creates the "Scripts", "cache", and "fitFilesFolder" at level 0 (one folder level above level1, the EXOSIMS repository). This script copies the makeSimilarScripts.py, singleRunPostProcessing.py, and multiRunPostProcessing.py scripts to the level 0 scripts folder. The script also copies deleteMe100.json (a templeate script), makeSimilar.json, makeSimilar2.json (the makeSimilarScripts.py instruction files) to the Scripts folder which form the base examples of how to use makeSimilarScripts.py. This script uses setuptools.command.easy_install to install the python modules listed in requirements.txt.

2 Directory Tree



2.1 cache

2.2 Scripts

The Scripts folder is where makeSimilarScripts.py is designed to be run from. The Scripts folder should contain all makeSimilarInst.json files. If makeSimilarInst.json file refers to a Template.json file as the template script, then that script should be located in the Scripts folder. A note on operation of makeSimilarScripts.py: If a makeSimilarInst.json file refers to a Template file Template.json and this Template.json file has a field which is a file name, that file will be copied into the subfolder makeSimilarInst_Template with 'auto_makeSimilarInst_Template' prepended to the filename (This ensure that all necessary information required to replicate a given run is contained within the makeSimilarInst_Template folder).

Example: If a makeSimilarInst.json file refers to a Template.json file with a "missionSchedule" dictionary key with value "sampleOB.csv", then sampleOB.csv should reside in the Scripts folder. When makeSimilarScripts.py is run, a copy of sampleOB.csv will be made in makeSimilarInst_Template with filename auto_makeSimilarInst_TemplatesampleOB.csv.

2.3 fitFilesFolder

The fitFilesFolder is where fit files should be stored (EXOSIMS does not require this as of 9/18/2018 and is not anticipated to require this).

3 makeSimilarScripts.py

This script is designed to be run from the 'EXOSIMS/../Scripts' folder. The template script scriptAAA.json must be located in the 'EXOSIMS/../Scripts' folder.

Inputs: makeSimilarInst.json - this file is located in the run directory and encodes the instructions for making a set of similar json input scripts. scriptAAA.json

Outputs: queueXXX.json - this file is a queue containing a list of all

3.1 makeSimilarInst.json

Every configuration of this script has keys 'sweepType' and 'sweepParameters'. The 'sweepType' field must be a string. 'sweepType' has valid input fields of "SweepParameters" and "SweepParametersPercentages". The 'sweepParameters' field must be a list of strings where each string is a valid key of the EXOSIMS json script.

3.1.1 SweepParameters

The 'sweepType' "SweepParameters" is designed to sweep any set of parameters in 'sweepParameters' key. The makeSimilarInst.json must have the 'sweepValues' key. The 'sweepValues' key must have the same length as the 'sweepParameters' key. i.e. 'sweepParameters':[param1, param2, ..., paramn] and 'sweepValues':[vals1, vals2, ..., valsn] where the length of both lists are n Each valsi in 'sweepValues' is a list of values. Each list valsi in 'sweepValues' has the same length. The number of executable EXOSIMS created has length len(valsi).

3.1.2 SweepParametersPercentages

The 'sweepType' "SweepParametersPercentages" is designed to take a set of parameters and set of percentages and run percentage based sweeps on single and combinations of parameters. This 'sweepType' requires a 'sweepPercentages', 'sweepParameters', and 'sweepCombNums'.

The 'sweepCombNums' are a list of sweep combination lengths (i.e. sweepCombNums = [1] means each parameter).

The 'sweepPercentages' are a list of percentages where XX% is 0.XX and -XX% is -0.XX. The list must contain at least 1 element but may contain an arbitrarily large number of elements. No element may contain less than or equal to -100%.

The 'sweepParameters' can be any number of parameters, but all parameters will be varied by the same percentage.

4 runQueue

Inputs: outpath (string) - this is the full filepath to the "output directory". This "output directory" is the location that each "job" directory will be saved. A "job" is a specific json script which is executed over a number of runs (numRuns). queue.json (file) - see 5 (FUTURE 9/25/2018) *IF queue.json contains "outpath" and outpath is not specified as an input, then it will use the value of "outpath" presuming it is a valid filepath.

Outputs:

5 queue.json

This file is located in the "EXOSIMS/../cache" directory. This json file encodes the "scriptNames", "numRuns", "singleRunPostProcessing", "multiRunPostProcessing", and "outpath".

"scriptNames" is a list of strings where each string is a filename of a file that exists in the "EXOSIMS/../Scripts" folder.

"numRuns" is a list of integers with same length as "scriptNames" where each number of runs is the number of runs to execute for the corresponding script name.

"singleRunPostProcessing" is a dictionary where each dictionary key is the name of a post processing script to call and the value of each dictionary key represent the arguments of that script(EXACT FORMATTING OF ARGUMENTS TBD). Each script-key in "singleRunPostProcessing" will be run on all of the "Run Types" in the folder specified by the "outpath" key.

"singleRunPostProcessing" is a dictionary where each dictionary key is the name of a post processing script to call and the value of each dictionary key represent the arguments of that script (EXACT FORMATTING OF ARGUMENTS TBD). Each script-key in "singleRunPostProcessing" will be run on all of the "Run Types" in the folder specified by the "outpath" key.

"outpath" specifies the path to place each of the "Run Type" folders. NOTE: outpath is overridden by a manual input to runQueue.py.

"PPoutpath" is where each of the runs single run and multi-run post processing will be run on.

6 createOBdurArray.py

This script is designed to create observing blocks start and end times to execute missions over. The top, we input exoplanetObsTime and maxNum Years. We calculate maximum number of days in a year. We calculate the mission Portion. This should be constrained for what I am trying to do. We create OBdur2, a list of logarithmically spaced Observing Block Durations (50 different durations from a duration of 0.1 to 365days). We also create 10 additional OB durations from 2.5 to 10.5 with spacing of 1 day in between each (verify this against simulations I actually ran. Not 100% on this). HarmonicDistOB[OBdur2][0-startTimes, 1-endTimes][observing blocks]

7 runPostProcessing.py

runPostProcessing is run by running %run runPostProcessing.py –queueFileFolder "/full path to json instruction file or json instruction folder/"

Executing runPostProcessing as main calls the queuePostProcessing method if any queueFileFolder input is provided.

Logic is implemented in the queuePostProcessing method to determine whether the queueFileFolder information is for a specific file or a folder to search for a queue* file in (a valid file or folder containing a valid file must be passed).

This queue file must be in json format. The queue file CAN but is not required to have the following dictionary keys outpath, PPoutpath, scriptNames, singleRunPostProcessing, multiRunPostProcessing. outpath is the path to the folder containing folders to analyze (this is the outpath specified in runQueue). PPoutpath is the Full Path to the directory the multi-run analysis outputs to. scriptNames is a list of Names of Run Type Folders containing runs to analyze. singleRunPostProcessing is a list of dictionaries which must contain key 'analysisName' and can contain 'args' key which

contains the arguments for the post processing script. For singleRunPost-Processing, each 'analysisName' will be run over each run in 'scriptNames'. Behavior and requirements for multiRunPostProcessing are the same. In multiRunPostProcessing, each 'analysisName' will be run over the set of 'scriptNames'. singleRunPostProcessing are run first, then multiRunPost-Processing. MOTIVATION with this architecture, we are able to use the same queue.json file used to start a runQueue process to start creating plots for each individual run and process the set of runs.

8