GMAT Automation Software User Manual

Table of Contents

[1 Introduction 2](#_Toc101606669)

[1.1 Runtime Directory Structure 2](#_Toc101606670)

[1.2 System 2](#_Toc101606671)

[1.3 Building GMAT Automation Library 3](#_Toc101606672)

[2 GMAT Model Generation 6](#_Toc101606673)

[2.1 Excel “ConfigSpec” Workbook 6](#_Toc101606674)

[2.2 Points of Variation 8](#_Toc101606675)

[2.3 Generation of the Batch Files 9](#_Toc101606676)

[3 Execute GMAT with Batch Model Files 11](#_Toc101606677)

[4 Report Generation 12](#_Toc101606678)

# Introduction

The Goddard Mission Analysis Tool (GMAT) is a NASA supported open-source astrodynamics, trajectory and orbital analysis tool that is configured and driven by an application specific scripting language. [1]

The normal use case for GMAT is to autoconfigure scripts using a Graphical User Interface (GUI). In this case GMAT is configured and executed for a single mission scenario. Another use case, which GMAT Automation supports, is to execute batches of models with each model instantiating a point of variation, such as a spacecraft orbit or propulsion variation. The GMAT Automation package described in this document supports that use case.

The GMAT Automation package (gmatautomation) is a related set of Python scripts used to generate and run batches of GMAT model files. These model files vary in terms of resource values in accordance with an Excel workbook referred to as a “configspec”.

The Python scripts to generate models are located in sub-package modelgen:

* modelpov.py
* fromconfigsheet.py
* modelgen.py
* gmat\_batcher.py

The GMAT models generated using the modelgen package are generally designed to be batch executed.

The Python script to batch execute these modes is located in the sub-package modelcontrol.

The GMAT Automation package also provides Python scripts to normalize and format GMAT Report Files as Excel workbooks. Each GMAT model system generally has different use cases, so the Report Files vary, thus these data reduction scripts are designed as a simple class inheritance tree, to allow end user specialization of the workbooks.

The python scripts to reduce reports are located in sub-package reportgen.

These scripts will be discussed in the sections below.

## Runtime Directory Structure

“OUTPUT\_PATH” is defined in gmat\_startup\_file.txt located in the bin directory of the installed GMAT application. In Windows 7 - 10 the GMAT directory is under the %APPDATALOCAL% defined directory.

GMAT uses this directory to write its output reports.

This documentation assumes that output reports and log files are written to the directory located under the GMAT OUTPUT\_PATH reference. However, this path may be configured differently by the user in the %APPDATALOCAL%/GMAT/GMAT/bin/gmat\_startup\_file.txt.

The modelgen and modelcontrol packages assumes that the user has created a working directory for scripts where GMAT model scripts are stored. The user should also create a subdirectory called “Batch” under this working directory e.g., “D:\WorkingFolder\ProjectRoot\GMAT\Batch”.

## System

Extensive use of Excel constrains this system to execution on the Windows platform. The environment variable, %APPDATALOCAL%, is used by the python programs to locate GMAT and so is another Windows dependency. This variable was introduced in Windows 7 and subsequent.

Table , GMAT Automation Source Code Hierarchy

|  |  |  |
| --- | --- | --- |
| Figure , top level GMAT-Automation Directory | | |
| Figure , Contents of GMAT-Automation/src | | |
| Figure , /modelcontrol | Figure /modelgen | Figure , reportgen |

Execution of GMAT Model Files is math intensive and requires significant platform resources to execute. numpy libraries are used extensively and versions of python, numpy, SciPy, matplotlib and PyQT must all be compatible. As of gmatautomation version 0.2a the following libraries are installed, with recommended versions of xlwings and xlsxwriter.

* python 3.7.9,
* numpy 1.16
* matplotlib 3.4
* SciPy 1.5.3.
* xlwings-0.26.3
* xlsxwriter-3.0.2
* PyQt5 5.12

The “gmat\_batcher” script executes GMAT in multiple parallel processes using GMAT command line mode. The concurrency model is aimed at maximizing parallel execution of models and uses up to 80% of CPU resources on the below platform.

The gmat\_batcher.py execution was benchmarked with an HP-Z600 workstation as follows:

* Dual Intel® Xeon® E5506 CPUs at 2.13GHz
* 16GB of Installed DDR3 Ram
* 64-bit Windows 10 Pro, version 1803
* NVIDIA GTX 750i GPU

A time-out value of 5 minutes (300 seconds) is used to avoid “hung” threads. If the platform is unable to execute a model within this time, the system will proceed to the next. This event is logged as a warning and the user must check the logs to ensure that each batch file actually completes. To aid the user, the gmat\_batcher also writes time-tagged RunList and ReportList text files for each batch executed.

The logfiles from the automation scripts are stored under the directory where the program is executed.

## Building GMAT Automation Source Distribution

This build instruction assumes setuptools version 40 or later is used to build the package gmatautomation, reference the Python Packaging User Guide. [2]

The build and source directory structure as of v0.2a is shown in Table 1. The top-level folder is the folder above the source folder and above all folders and containing setup.cfg.

Edit the setup.py files to ensure the version tag is current and consistent across all packages. This version tag should match the GIT commit tag for what is being built. Also edit the \_\_init\_\_.py version information and package requirements to be consistent with any new files or packages added (or deleted) during development.

Run a command window as administrator.

Execute the source distribution build command in the top-level Alfano folder (above the src folder).

python setup.py sdist --formats tar

This will produce a tar file in the dist folder Winzip doesn’t handle gz format well, so we use tar internally. For PyPI distribution, follow the Python Packaging User Guide to build and distribute a gz file.

To deploy locally unpack the tar file under /dist, change directory to dist/<packagename>-<version>, where version matches the version in the top level setup.cfg.

Execute the setup command from the dist/<packagename>-<version> folder.

python setup.py install

# References

|  |  |
| --- | --- |
| [1] | GMAT Development Team, "General Mission Analysis Tool," Sourceforge, 2020. [Online]. Available: http://gmat.sourceforge.net/docs/R2020a/html/. [Accessed 23 April 2022]. |
| [2] | J. McNamara, "Creating Excel files with Python and XlsxWriter," Sphinx .18.6, 27 Feb 2022. [Online]. Available: https://xlsxwriter.readthedocs.io/index.html. [Accessed 23 April 2022]. |
| [3] | Zoomer Analytics, "xlwings - Make Excel Fly!," Zoomer Analytics LLC, 11 Apr 2022. [Online]. Available: https://docs.xlwings.org/en/stable/. [Accessed 23 April 2022]. |
| [4] | Python Software Foundation, "Python Packaging User Guide » Tutorials » Packaging Python Projects," Python Software Foundation, 19 April 2022. [Online]. Available: https://packaging.python.org/en/latest/tutorials/packaging-projects/#packaging-python-projects. [Accessed 23 April 2022]. |

# GMAT Model Generation

The script “modelgen.py” is the top-level script which generates the customized batch model files. The architecture of the GMAT model files used in the modelgen automation scheme is composed in three parts; “ModelMissionTemplate.script”, “Include\_StaticDefinitions.script” and “Include\_MissionDefinitions.script”.

ModelMissionTemplate.script is the top-level model definition in the batch system. It will be copied and renamed for each unique instance of a mission by modelgen.py.

The modelgen.py script appends #Include statements referencing the include files to the copy of ModelMissionTemplate.script. Two invariable #Include macros are appended to ModelMissionTemplate.script:

1. Include\_MissionDefinitions.script
2. Include\_StaticDefinitions.script

The bulk of GMAT model resource definitions do not change from model to model and are factored into the invariable macro, “Include\_StaticDefinitions.script”. This invariable macro must be maintained by the user.

The mission and spacecraft configuration are read from an Excel workbook by modelgen.py, which creates and saves mission specific include files in the Batch directory. Each generated include file is uniquely named with configuration, epoch, Julian day and time of generation, and each filename is prefixed with the string "Include\_".

The appended copy of ModelMissionTemplate.script is saved in a Batch directory. It is important that ModelMissionTemplate.script be kept small, because modelgen.py may copy it hundreds of times.

ModelMissionTemplate.script and it’s include files are intended to be executed from the GMAT command line. The GMAT GUI cannot be used to edit these files.

When it is necessary to create a new model, for instance to introduce a different Mission Sequence, proceed with design and test using the GMAT GUI, then cut and paste the static resource definitions into the Include\_StaticDefinitions.script, and cut and paste the mission sequence and user variable definition into the Include\_MissionDefinitions.script.

Resource definitions which change from file to file are generated by modelgen.py from the Excel mission definition. If dynamic resources are added, then code changes will likely be necessary. If the Excel mission definition changes, then code changes likely will be necessary.

The model files created (or changed) in the GUI should be factored and copied to the batch template files located in the GMAT "OUTPUT\_PATH":

Update the configuration workbook any time spacecraft or its hardware is changed, likewise the ModelMissionTemplate.script. Rerun modelgen.py following update of the Excel workbook.

## Excel “ConfigSpec” Workbook

The model generation system is designed for a specific mission called Alfano.

An Interface Agreement between the Python code and an Excel workbook must be enforced.

1. There exists an Excel workbook which contains a sheet named "GMAT" containing a contiguous table starting in cell "A1", which table consists of a first line of parameter names and successive lines of spacecraft properties and relevant hardware configuration.
2. A sheet named “Mission\_Params” also exists, containing named ranges “Inclination”, “Costate”, “Starting\_Epoch”, and “Mission\_Name”.

The first line table headings are usually not the same as GMAT resource names. The associated routine, "modelpov.py" defines a mapping of required GMAT resource names to worksheet table headings, which we refer to as parameter names.

Procedure “modelgen.py” will use this association to generate the correct GMAT resource names.

Procedure “fromconfigsheet.py” will extract only the parameter names defined in modelpov.py. Note that it is possible with this logic to extract NO data from the workbook, in this case the “modelpov.py” file may be edited to include the intended parameter names, or the workbook may be so edited.

Figure 6 shows an example ConfigSpec generated from analysis data using an Excel SQL query.

Variation of orbital elements is independent of hardware configuration. Specifically, inclination cases may be multiple for each given "GMAT" table row and are gleaned from a separate n x 1 table of values in named range, "Inclinations", contained in sheet "Mission\_Params". An associated n x 1 table of negative floating-point values in named range “Costates” has a row-by-row correspondence to the inclination values. The costate values specify the Alfano trajectory for combined altitude and inclination change.

The inclination values should be negative for decreasing inclination and positive for increasing inclination.

|  |
| --- |
| Figure , GMAT ConfigSpec Table in Excel Workbook |

|  |
| --- |
| Figure , Mission Parameters in ConfigSpec Workbook, Starting\_Epoch Highlighted |

Figure 7 shows an example "Mission Params". Note that there are more orbital elements on it than are presently implemented in code.

At the top right in cell B1, the mission name must be defined in a named range called “Mission\_Name”. The mission name is used by the script to name various files and can be any string by which the user can identify model files associated with this set of configurations and mission definitions.

Cases of initial epoch are defined by an n x 4 array of values in named range, "Starting Epoch". Each row contains a UTC formatted time and date value in column 1, e.g., "20 Mar 2020 03:49:00.000 UTC".

For display purposes columns (n,2), (n,3), and (n,4) contain a GMAT viewpoint vector consisting of x, y, and z components of rendering camera position (J2000 ECI coordinate system). These viewpoint vectors are associated with each epoch value. This is under the assumption that the GMAT application is executed with the GUI enabled, which will be a performance burden.

## Points of Variation

The “modelpov.py” script encapsulates the Points of Variation in a Python dictionary structure for GMAT model generation. It is used by "fromconfigsheet.py" and "modelgen.py”.

The implemented points of variation in the model file are listed here. The list form is:

Table Heading: [top model name] [sub-model name].[resource name].

Points of variation:

* Dry mass: Spacecraft EOTV.DryMass
* Starting Epoch: Spacecraft EOTV.Epoch
* This is a list of epoch values to be executed.
* Inclination: Spacecraft EOTV.INC
* This is a list of inclination values to be executed.
* Costate: The constraint value for final SMA and Inclination change
* This is not actually a model parameter but is used to select a JSON control file.
* Max Thrust Power: ElectricThruster HET1.MaximumUsablePower
* Min Thrust Power: ElectricThruster HET1.MinimumUsablePower
* Efficiency: ElectricThruster HET1.FixedEfficiency
* Isp: ElectricThruster HET1.Isp
* Thrust: ElectricThruster HET1.ConstantThrust
* Available Power: SolarPowerSystem EOTVSolarArrays.InitialMaxPower
* Propellant: ElectricTank RAPTank1.FuelMass
* Output: ReportFile1 ReportFile1.Filename
* Viewpoint: Orbit View DefaultOrbitView.ViewPointVector

In order to cover the range of eclipse conditions the EOTV Epoch is typically varied for the four seasons:

* 20 Mar 2020 03:49 UTC
* 20 Jun 2020 21:43 UTC
* 22 Sep 2020 13:30 UTC
* 21 Dec 2020 10:02 UTC

The Viewpoint is superfluous in most cases, since model execution is intended to be in batch mode for this system. However, if graphic output is desired, the OrbitView viewpoint is varied associated with the Starting Epoch and is documented as part of the Starting Epoch array as follows:

* 20 Mar 2020 03:49 UTC, (80000, 0, 20000)
* 20 Jun 2020 21:43 UTC, (0, 80000, 20000)
* 22 Sep 2020 13:30 UTC, (0, -80000, 20000)
* 21 Dec 2020 10:02 UTC, (-80000, 0, 20000)

The ReportFile1.Filename and the Model name are generated by concatenating the configuration, the Starting Epoch and the Inclination.

## Generation of the Batch Files

Script “modelgen.py” produces GMAT model Include files containing variants of model resource values and parameters. The code supports batch processing of different mission scenarios in which the spacecraft configuration and/or initial orbital elements vary.

The approach utilizes the GMAT 2018a #Include macro, which loads resources and script snippets from external files. The script creates include files whose parameter and resource values vary in accordance with the configspec Excel workbook.

A top level GMAT script template must exist in the GMAT model directory. This script template will be copied and modified by modelgen.py. The script template must contain the GMAT create statements for GMAT resources.

The script template will be appended by three #Include statements referencing include files as follows:

1. a GMAT script file defining the static resources, those that don't change per run.
2. An include file generated by "modelgen.py".
3. An include file containing the GMAT Mission Sequence definition, which doesn’t change per run.

The appended script template will be copied to a "batch" directory under a mission specific filename. An example is:

AlfanoXfer\_64HET8060W\_16000.0kg\_20Jun2020\_28.5\_J039\_172018.script

which references:

Include\_16HET8060W\_16000.0kg\_20Jun2020\_28.5\_J039\_172018.script

There will be a unique copy of the script template for each line of the configspec workbook, elaborated by the number of inclination cases, and elaborated by the number of Starting Epoch cases. The total number of files, template and include, will be:

2 \* [Number of configspec rows] \* [Number of inclination cases] \* [Number of Starting Epoch cases]

Once these files are copied, a list of their filenames will be written out as contents of a batchfile in the GMAT model OUTPUT\_FILE defined path. The model name will be of the form:

[Mission Name] + '\_\_RunList\_[Julian Date-time] + '.batch'

An example is:

"AlfanoXfer\_\_RunList\_J009\_0537.25.batch".

A dictionary is used to drive the actual resources and parameters written to Include macro files.

The dictionary is factored into "modelpov.py" such that additional resources may be added or deleted with minimal change to code in “modelgen.py” and “fromconfigsheet.py”.

Include file 1 is an invariable file copied from the initial model file written by GMAT. This is a large file of Resources and Parameters which do not change from run-to-run.

In the case of a new model or change …

The points of variation probably should be updated in "modelpov.py" a new model is generated.

The external module "fromconfigsheet.py" is called to read excel worksheet to update the values of the dictionary.

The Alfano trajectory is used in the current model mission, and a user defined parameter, the costate, must be updated in concert with the inclination to execute the Alfano-Edelbaum yaw control law. The costate calculation is out of scope as of this version [TODO]. The costate is specified on the Mission\_Params sheet of the workbook.

Notes:

1. To model the return trip of the reusable vehicle, two include files must be generated, one with payload mass included, one without. This should be handled in the configspec workbook if a return trip is required.
2. Dry mass varies with the vehicle power and thrust.
3. Efficiency, thrust and Isp vary with the selected thruster set-points.
4. In order to cover the range of eclipse conditions the EOTV Epoch is varied for the four seasons:

20 Mar 2020 03:49 UTC

20 Jun 2020 21:43 UTC

22 Sep 2020 13:30 UTC

21 Dec 2020 10:02 UTC

The epoch date is specified in the configsheet workbook.

1. Propellant is given as an initial calculation in the workbook, then the actual value from a run is substituted and the model rerun until convergence. This is presently a manual process.
2. A GMAT Report File is output into the OUTPUT\_PATH/Report directory for each run. This is a .csv file and its name must be unique for each iteration of the model. The ReportFile1.Filename is generated by “modelgen.py” which suffixes the Julian date-time to the filename.
3. The OrbitView viewpoint is superfluous in most cases, since model execution is intended to be in batch mode for this system. However, if graphic output is desired, the convention is to vary the viewpoint with the Starting Epoch
4. The top level GMAT script is be called by the batch facility, “gmat\_batcher.py”.
5. TODO: the initial baseline depends on the exact spacecraft and hardware created in the top-level template. Five of these represent points of variation:

* Create Spacecraft EOTV;
* Create ElectricThruster HET1;
* Create SolarPowerSystem EOTVSolarArrays;
* Create ElectricTank RAPTank1;
* Create ReportFile, multiple files possible, various names

These instance dependencies can be avoided by reading and interpreting ModelMissionTemplate.script where the user will copy his definitions of these resources.

# Execute GMAT with Batch Model Files

Python script “gmat\_batcher.py” reads the batch file created by modelgen.py, and executes GMAT in command line mode for each filename listed in the file, “AlfanoXfer\_RunList\_”[julian time-date]”.batch”. The script provides a file selection GUI to select this RunList batch file.

There are four assumptions for execution:

* Python 3.5 or subsequent is the default python interpreter and libraries are installed per the required library list, Table 1.
* GMAT.exe must be in the executable path.
* The user's platform must be a multi-core platform capable of executing each model file within 5 minutes. (5-minute limit is determined by the “cpto” global variable in gmat\_batcher.py).
* The factored model topology is as created by modelgen.py.

Table 2, Required Library List

|  |
| --- |
| import os  import time  import re  import platform  import logging  import traceback  import getpass  import random  from pathlib import Path  import subprocess as sp  from multiprocessing import Pool  from multiprocessing import cpu\_count  from multiprocessing import Manager  from PyQt5.QtWidgets import(QApplication, QFileDialog) |

Depending on the number of generated model files the execution may be quite lengthy, and the platform should not go into a power time-out or “sleep” while batch execution proceeds. For a session with 1400 model files, the platform as defined above will require approximately 3 hours of execution.

The author uses another script named “Move Mouse”, available in open source form (movemouse.codeplex.com).

The output of the batch scripts is assumed to be GMAT Report Files, which are uniquely named in accordance with the top level script name and are stored in folder named “Reports” under the current working directory for the program.

At the end of each execution GMAT writes to a “personalization file” called “MyGMAT.ini which is located under the GMAT root directory in the /data folder. The parallel execution of GMAT instances randomly causes file contention and an exception may be thrown. The exception will “hang” a process completion and should be cleared as quickly as possible when it appears. Better, the gmat\_startup\_file.txt located under the GMAT root directory in the /bin folder may be modified with the line “WRITE\_PERSONALIZATION\_FILE = OFF”, which will prevent the file contention.

# Report Generation