

POSEIDON Desktop Application Manual (V0.5.0)

This desktop application is created using Javascript and Python under the Electron and Flask frameworks. It offers a wide range of features designed to help users track and visualize satellite orbits with ease. Powered by Cesium, a powerful library for 3D globes and maps, the app provides real-time 2D and 3D visualizations of satellite orbits, complete with the ability to input Two-Line Element set (TLE) data, search for satellites by name or catalog number, and obtain the latest pass time and telemetry based on user preferences. Additionally, the app has a command window for generating commands, as well as a calendar window for scheduling. For the most up-to-date source code, simply refer to the latest commit on our [GitHub repository](#).

High-Level Feature Overview:

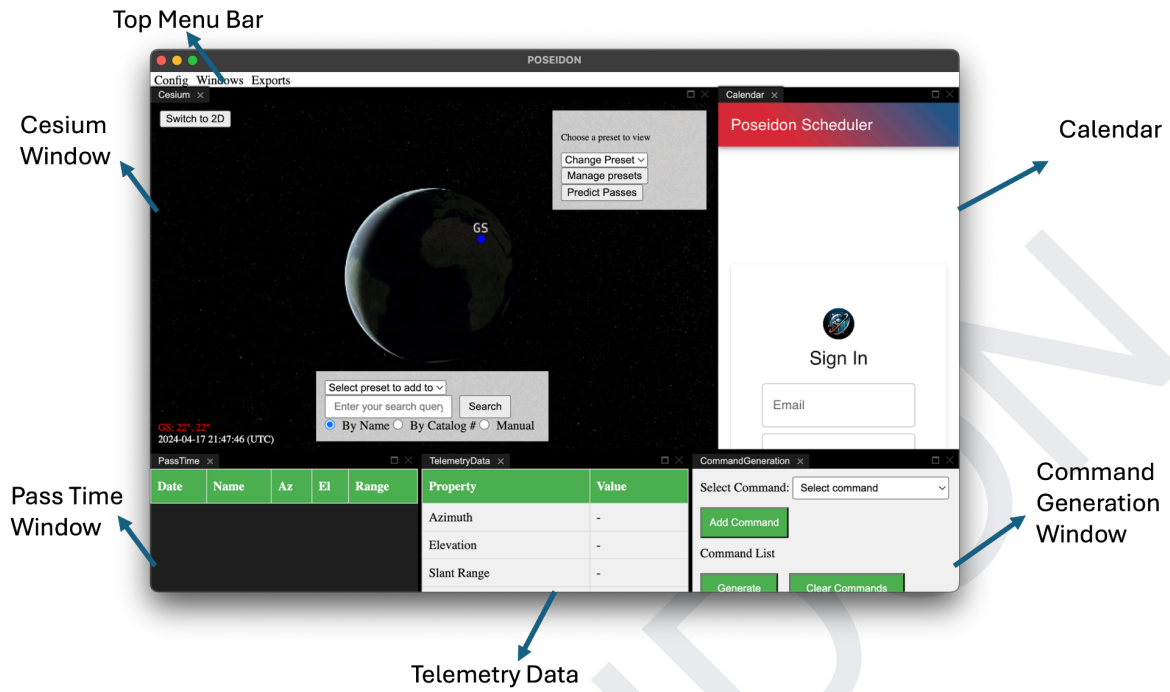
- 3D Orbit Visualization: Real-time satellite orbit visualization with Cesium.
- Telemetry Data Calculation And Visualization: Real-time telemetry data update for corresponding TLE/satellite input.
- Preset list management.
- TLE Data Input: Input box for TLE data with validation functionality.
- Command Input: Chat window for command input and generation.
- Calendar: Calendar window for scheduling.

Installation:

Click the latest version of the release package for installation

*For developers, please check the README or the [Development and Packaging](#) section of this manual for instructions

Contents:



Top Menu Bar:

Config

Ground Station

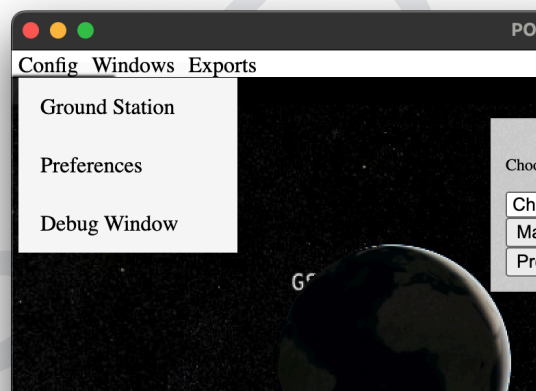
Users can input the latitude and longitude of their ground station, which will be saved in Poseidon until any updates are made. This information calculates pass-time details and will be displayed on the Cesium map visualization.

Preferences

Users can input the customized JSON file for their command, which will be parsed for the [command generation functionality](#). Users can also input their desired time zone for pass-time calculation.

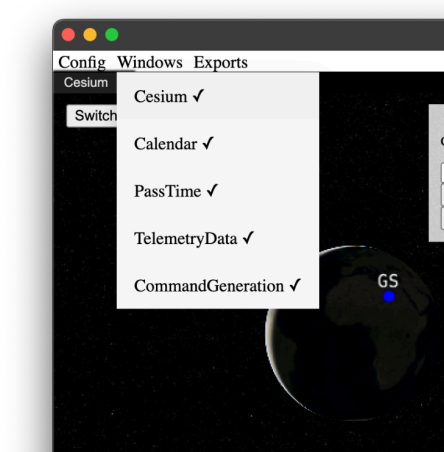
Debug Window

Users can click on the Debug window if they experience any unexpected behavior to check with any error outputs in the console for further debug instructions.



Windows

The application displays all open windows for the user to view. Should the need arise to close or reopen a window, the user can easily achieve this by clicking on the corresponding row. The check mark indicates that the window is actively being displayed in the application.



Exports

Users can export the generated pass time by clicking the button below and saving it to their desired location.

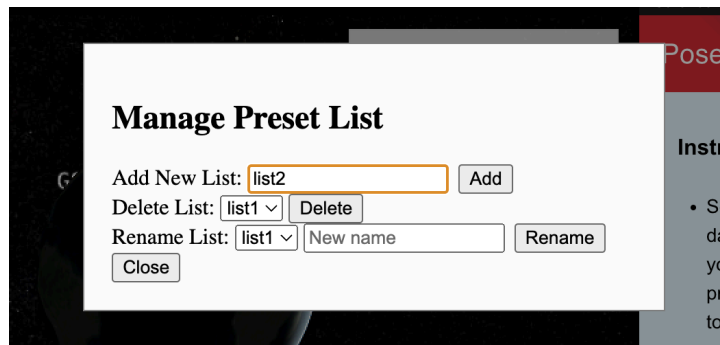


Cesium Window:

Preset List

Manage Presets

The “Manage Presets” window allows you to add, remove, or rename satellite lists. Press close or ‘esc’ to close the window.

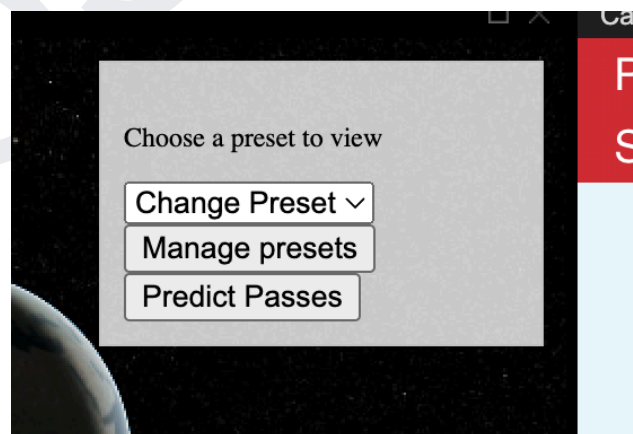


Predict Passes

When this button is clicked, pass times will be generated for all selected satellites and displayed in the pass times window.

Change Preset

Change the Preset list that is displayed. When a preset list is selected, satellites in that preset list are displayed. Satellites on display can be removed from the list (using the “x” button on the right side of the satellite name) or selected (using a checkbox) to predict pass times. If only one satellite is selected (using a checkbox), its TLE will be used to update the data in the telemetry window.



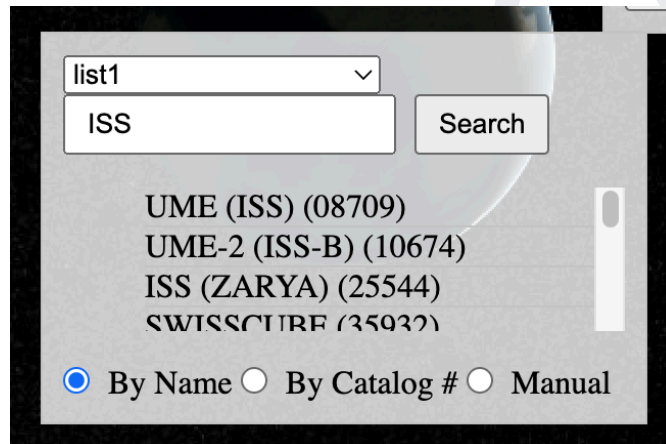
Search

Select Preset

When adding a satellite to a preset list, first select the desired list to add to from this dropdown otherwise, an error message will appear after the user clicks on add.

Search By Name

Upon choosing a preset list to add to, the user may enter a keyword to search for a satellite name. To initiate the search, the user should press the search button. If the keyword matches fewer than 20 results, these results will be displayed for the user to scroll through and select by clicking on the name within the list. However, if the search yields more than 20 results, an error message will appear, prompting the user to refine their keyword search for more precise results.



list1

ISS

Search

UME (ISS) (08709)
UME-2 (ISS-B) (10674)
ISS (ZARYA) (25544)
SWISSCURE (35032)

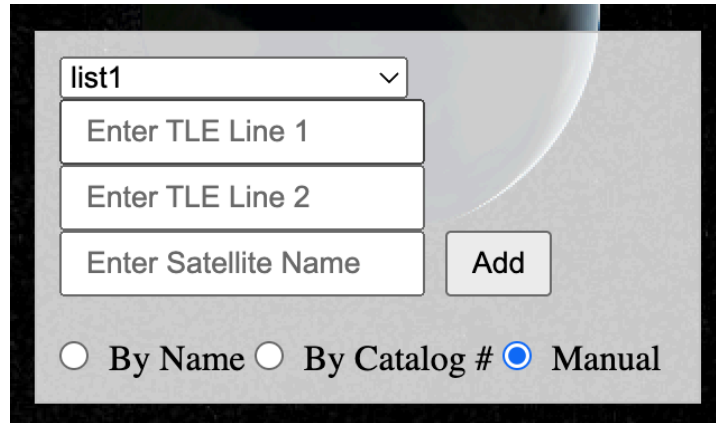
☒ By Name ☐ By Catalog # ☐ Manual

Add By Catalog Number

When selecting a pre-existing list to add to, the user has the option to input the catalog number of the desired satellite. If the catalog number is accurate, upon pressing the "add" button, a notification will appear confirming the satellite's successful addition to the list. Otherwise, an error message will be displayed.

Add Manual TLE

If users need to manually input TLE data in order to add a satellite, they can do so by entering the TLE lines and a name for the satellite. Upon clicking the "add" button, the system will verify that the TLE lines adhere to the required format. If they do, the satellite will be successfully added to the selected list. If not, an error message will appear to notify the user. **(Note manually inputted satellite will not be updated automatically)**



A screenshot of a software interface for entering TLE data. It features a dropdown menu labeled 'list1' with a downward arrow. Below it are three text input fields: 'Enter TLE Line 1', 'Enter TLE Line 2', and 'Enter Satellite Name'. To the right of the 'Enter Satellite Name' field is an 'Add' button. At the bottom, there are three radio buttons: 'By Name', 'By Catalog #', and 'Manual'. The 'Manual' radio button is selected, indicated by a blue dot.

Other

Ground Station, Date-time Text

Displays the ground station longitude and latitude, as well as the current date-time in the local timezone or UTC. Click the date-time text to toggle between Local and UTC, or toggle in the Config window in the Top Menu Bar.

2D/3D Visualization Toggle

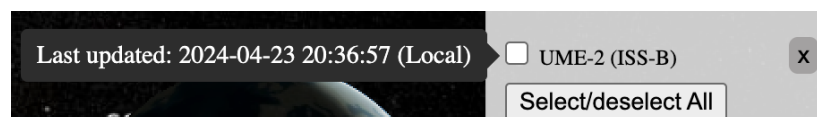
Click to toggle between the 3D or 2D display of the ground station, range circle, and ground track.

Select All Button

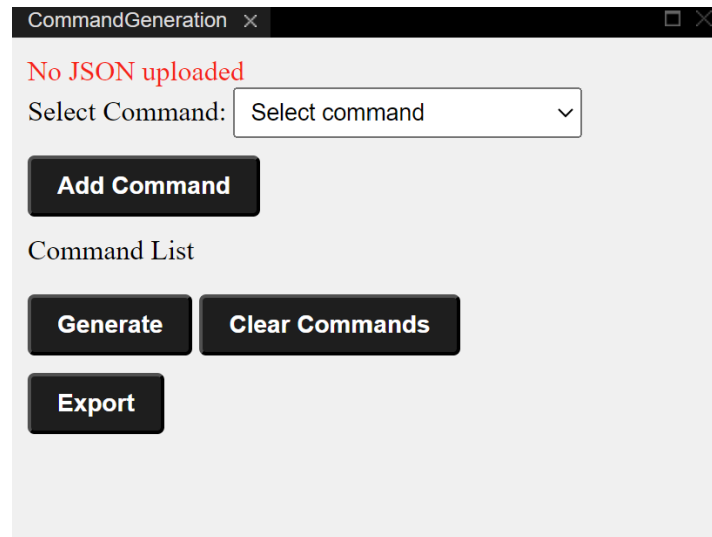
Click this button to select/de-select all the satellites in a preset list.

Last Update Time

All TLE data in the preset list is updated every six hours or on the start of the app. The user can hover over the satellite name to check the last updated time for that satellite's TLE. Manually entered TLE will not be updated.



Command Generation Window:



CommandGeneration x

No JSON uploaded

Select Command:

Add Command

Command List

Generate Clear Commands

Export

1.1 JSON File

The most important aspect of the Command Generation module is the development and implementation of a correctly formatted JSON file. Without it, commands may not populate or generate correctly. The general layout of a JSON file with the necessary fields outlined would look like this:

```
1  [
2    {
3      "ID": "CMD001", ## FOLLOW A SIMILAR NAMING CONVENTION, CMD000 IS RESERVED
4      "Name": "Command Name",
5      "Description": "Command Description",
6      "Family": "Command Family",
7      "Parameters": [
8        {
9          "Name": "Input Name",
10         "Type": "Input Type",
11         "Description": "Input Description",
12         "Enclosure": "Input Enclosure Character"
13       },
14       {
15         "Name": "Dropdown Name",
16         "Type": "Dropdown",
17         "Description": "Dropdown Description",
18         "Options": [
19           {
20             "Name": "Option 1",
21             "Description": "Option 1 Description",
22             "Command": "Option 1 Command Value"
23           },
24           {
25             "Name": "Option 2",
26             "Description": "Option 2 Command Value",
27             "Command": "Option 2 Command Value"
28           }
29           ## ADD ADDITIONAL OPTIONS FOR THIS DROPDOWN HERE
30         ]
31       }
32     } ## ADD ADDITIONAL PARAMETERS HERE (INPUTS, DROPDOWNS)
33   ]
34 }
35 ## ADD ADDITIONAL COMMANDS HERE
36 ]
```

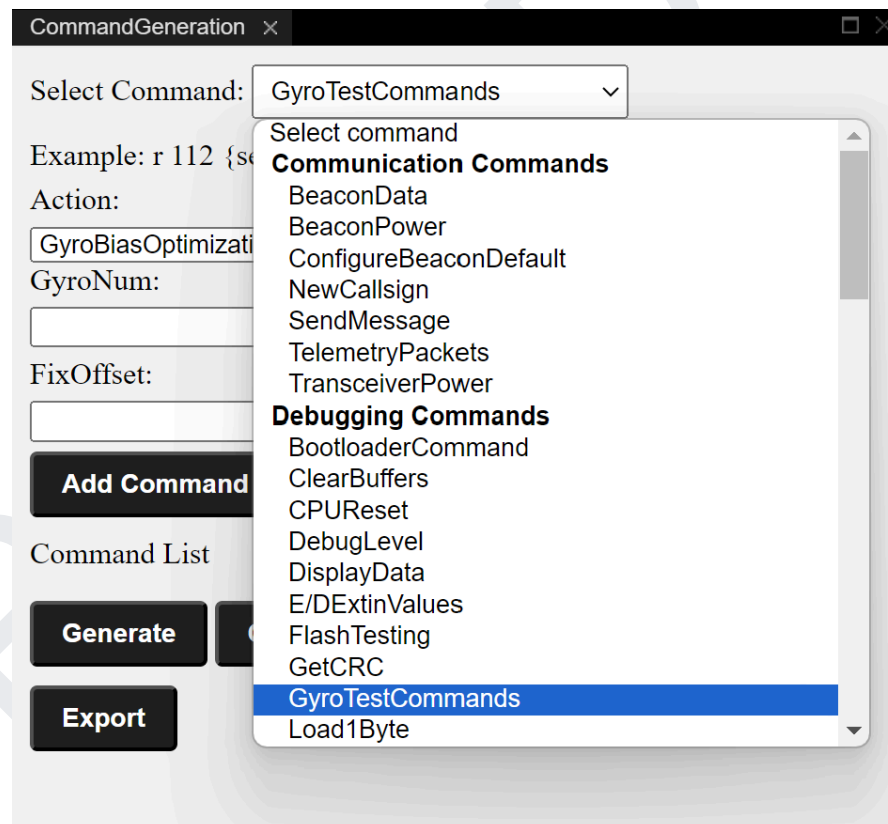

Additional Note: CMD000 is reserved for command options such as “receive” and “forward”. The current command generation code will create long command strings, and will split them by when a CMD000 command is inputted.

To upload the JSON file, in the top left menu bar click Config -> Preferences, and from there you will be able to upload the JSON file. It must be a file ending in “.json”. Once uploaded, it will remain in the system even when you close/open the application again. If you have a different or updated JSON, reupload it and the changes will take place in the Command Generation Window.

If there is no JSON file uploaded, the Command Generation Window will state “no JSON uploaded” in red, and you will not be able to use the command generation function.

1.2 Command Dropdown

The command dropdown is the first dropdown on the page. This displays all of the commands sorted by families and in alphabetical order. From here, you are able to select any command, which will then dynamically load an example of the command and any inputs that you may need to add.



Example

This will give you a basic understanding of what the generated function may look like.



Dropdown/Inputs

These are the fields you will want to enter in order to fully complete your command. Some dropdowns may have more dynamically loaded nested dropdowns/inputs so make sure to pay close attention to what you may need to add. Hovering over the dropdown options or the input box will give you a description of what it is or what data type you will be needing to add. Do not worry about adding enclosures, as those are handled in the JSON file.

Add Command

This will add the command to your command list.

1.3 Command List

This holds all of the commands that you have inputted, with the top being the first command added, and the bottom holding the most recent command. It will show the command ID, along with any dropdown selections or input that you have chosen to add.

The screenshot shows a web application window titled "CommandGeneration". It features a "Select Command:" dropdown menu with "NewCallsign" selected. Below this is an "Example: 159 \"ABC123\"" and a "CallSign:" input field containing "NOT BINGO". There are "Save" and "Add Command" buttons. The "Command List" section displays a list of commands with their IDs and values, each with a delete button (X):

- CMD000 r [X]
- CMD047 "BINGO" [X]
- CMD051 11 Hello World [X]
- CMD000 f [X]
- CMD047 "JOKER" [X]

At the bottom are "Generate" and "Clear Commands" buttons.

Edit a Command

If you click on any command, you are able to edit it as the dropdown/inputs get dynamically repopulated. A "save" button will appear, which you can press to save your changes.



Delete a Command

If you no longer want a command, you will be able to press the "X" button and delete it from your list. There is no undo button, so once something is deleted you will need to add it again.

Clear Commands

Pressing this will completely clear all the commands in the list and you will be able to start from scratch. There is no undo button for this either, so think carefully before clearing.

1.4 Generate Button

This will generate the commands by using the command_generation.js file. It uses the information from the inputted JSON, and will turn all of the items in the command list, into fully generated strings.

Generating the commands will not clear the command list, so you will be free to edit anything you may need.

The screenshot shows a web application window titled "CommandGeneration". It contains a "Command List" section with five entries, each with a delete button (X):

- CMD000 r
- CMD047 "BINGO"
- CMD051 11 Hello World
- CMD000 f
- CMD047 "JOKER"

Below the list are two buttons: "Generate" and "Clear Commands".

At the bottom, there is a text box containing the generated command strings: "r 159 "BINGO" 166 11 Hello World,f 159 "JOKER"".

Below the text box is an "Export" button.

Outputted Text Box

This will hold the copyable generated commands and save them to a safe place. You may also just use this to read the commands you have generated to ensure you have added what you needed.



Export

This will allow the user to save the generated commands to a “.txt” file anywhere on their machine.

POSEIDON



Calendar Window:

Please refer to the [Calendar documentation](#).

POSEIDON



Development and Packaging:

Instructions can also be found in the README on our [GitHub repository](#)

Running the Program (Developers)

Ensure Node.js and npm are installed on your system. Open a terminal or command prompt, navigate to the app's directory, and run:

```
npm install # Install dependencies
cd backend # Get into the backend dir
# Install all the requirements to local env
pip install flask sgp4 scipy requests skyfield waitress setuptools
/path/to/your/desired/python3 -m pip install -e .
```

```
# Option 2: create a virtual env
# ---for windows--- #
py -3.9 -m venv <venv-name>
.\<venv-name>\Scripts\activate
pip install flask sgp4 scipy requests skyfield waitress setuptools
tzdata pyinstaller
pip install -e .
```

```
# ---for linux and mac--- #
python3 -m venv venv
source venv/bin/activate
pip install flask sgp4 scipy requests skyfield waitress setuptools
pip install -e .
```

```
npm start # Start the application
```

****Note for error "'run' can't be opened because Apple cannot check it for malicious software."**

Right-click (or control-click) on the backend/dist/run executable in Finder, and select Open from the context menu. This action brings up a dialog with an option to open the app anyway. This method only needs to be done once, as macOS will remember your choice for this app in the future.

****Note:** Python vers. 3.9 required. Some packages aren't supported in later versions.



Packaging the Application

For packaging the application, electron-builder is preferred. It can easily package and build a ready-for-distribution Electron app for macOS, Windows, and Linux.

First, package the python environment:

```
cd backend
./<venv-name>/Scripts/activate
python setup.py sdist
pip install dist/tle_calculations-0.0.0.tar.gz

# For windows only:
pyinstaller --onedir --add-data "tle_calculations;tle_calculations"
--add-data "./<venv-name>/Lib/site-packages/waitress;waitress"
tle_calculations/run.py

# For mac only:
pyinstaller run.spec
```

Second, install electron-builder as a dev dependency:

```
npm install electron-builder --save-dev
```

Then, run:

```
# For your current platform
npm run dist

# For a specific platform
npm run dist --mac
npm run dist --win
npm run dist --linux
```



Appendix: POSEIDON Source Code Structure:

```

├── ExampleCommandLibraryJSON.json  Example Command JSON format for the users
├── LICENSE
├── POSEIDON User Manual.pdf
├── README.md
├── backend
│   ├── database.ini  SQLite initialization code called when pip install .
│   ├── poseidon.db  SQLite Database
│   ├── requirements.txt
│   ├── run.spec  Packing specs, called when run dist
│   ├── setup.py
│   └── tle_calculations  Directory contains all flask API calls for satellite retrieval and calculation
│       ├── __init__.py
│       ├── database.py
│       ├── orbit.py
│       ├── run.py
│       ├── tle.py
│       └── unitTests
│           └── tle_calculation_test.py
├── command_generation  Directory contains all Command generation functions
│   └── command-generation.js
├── js_component  Directory contains all functions in the front-end separated by functionality
│   ├── cesium.js
│   ├── menu.js
│   ├── orbit.js
│   ├── passtime.js
│   ├── popup.js
│   ├── presetList.js
│   ├── renderer.js
│   ├── search.js
│   └── telemetry.js
├── main.js  Main thread for electron
├── mainWindow.css
├── mainWindow.html  Main front-end HTML setup
├── media_res  Media resource directory
│   ├── icon.png
│   └── splash.png
├── node_modules  Directory for node modules packages
├── package-lock.json  Dependencies requirement file by electron
├── package.json  Electron packaging metadata
├── splash_screen  Splash screen directory
│   └── splash.html

```

*Code-related documentation can be found in each corresponding source file