Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

positional-iss API Reference

Akhil Sadam | as97822
Aerospace Department, University of Texas at Austin

Abstract

An containerized Flask webserver designed for querying ISS sightings and positions on February 13, 2022. Midterm project for COE332. R is used to produce documentation.

Keywords: positional-iss, Docker, Flask, Python3, R.

1. Motivation

While the ISS (International Space Station) positions and sightings are freely available on the National Aeronautics and Space Administration website, finding data for a particular time and place is rather difficult.
For skywatchers, to be able to see the ISS, one will need to predict the location at a particular time, along with the visibility from their location. This project is an example containerized Flask webserver, and is designed to help with that goal, while also serving as an example for Flask REST-API containerization.
We provide easy access to sightings of the ISS from various regions and cities in the USA, particularly the Midwest, and also provide time-based positional data for February 13, 2022. The user can then extrapolate the current and future positions of the ISS from this data.

2. Implementation

This project uses Python3 (in particular Flask), and Docker for containerization. Specific Python3 package requirements can be found here. R and the npm package `@appnest/readme` by Andreas Mehlsen are used for documentation, but are not part of the API and will not be documented.

The source is available here, and a list of important files can be found below. ## Files

- `app/`: The application folder.
- `doc/`: A documentation folder.
- `Dockerfile`: A dockerfile for containerization.
- `Makefile`: A makefile for ease of compilation.
- `requirements.txt`: The list of Python3 requirements.
- `wsgi.py`: The main Python file.

The App/ Directory

- `api/`: Contains API route definitions in Python.
- `static/`: Contains static files for browser use.
- `templates/`: Contains Jinja2 templates for browser use.
- `test`: Contains testfiles in Python.
- `assets.py`: Collects static files for browser use.
- `routes.py`: Collects the API route definitions.
- `log.py`: Defines Python logger.
- `options.py`: Defines global options, like the application url.

2.1. Input Data

- The application queries data from the National Aeronautics and Space Administration (NASA) public website, in particular ISS positional information via the Public Distribution file and regional sighting data for the Midwest via the XMLsightingData_citiesUSA05 file.

Example input data is available at the above links.

3. Installation & Usage

A user can build this project from source, or use the provided Docker container on DockerHub.
A Docker installation is required for source builds, as we build and run a Docker image. The following commands are all terminal commands, and are expected to run on a Ubuntu 20.04 machine with Python3, and are written in that fashion. Mileage may vary for other systems.
We will describe the Docker installation first.

3.2. From Docker:

Install

To install the Docker container, first install Docker.

- `apt-get install docker` (if using an Ubuntu machine, else get Docker from docker.com.)

Next install the containers.

- `docker pull akhilsadam/positional-iss:0.0.2`

Run

To test the code, please run the following in a terminal.

- `docker run -it --rm akhilsadam/positional-iss:0.0.2 testall.py`

To run the code, please run the following in a terminal. The terminal should return a link, which can be viewed via a browser or with the `curl` commands documented in the API reference section.

- `docker run --name "positional-iss" -p 5026:5026 akhilsadam/positional-iss:0.0.2 wsgi.py`

Now we will move to the source installation.

3.3. From Source:

Since this is a Docker build, the requirements need not be installed on the server, as it will automatically be done on the Docker image.
All commands, unless otherwise noted, are to be run in a terminal (in the home directory of the cloned repository).

Build

Again, first install Docker.

- `apt-get install docker` (if using an Ubuntu machine, else get Docker from docker.com.)

Next, clone the repository and change directory into the repository.

- `git clone git@github.com:akhilsadam/positional-iss.git`
- `cd positional-iss`

Now build the image.

- `make build`

Run

To test the code, please run one of the following.

- `make test`
- `pytest`

To run the code, please run the following. The terminal should return a link, which can be viewed via a browser or with the `curl` commands documented in the API reference section.

- `make run`

To run a rebuild of the code, run this command instead. This command will automatically kill, rebuild, and test the code before running.

- `make iterate`

3.4. Usage

As mentioned above, please use a browser or the `curl` utility to view output. All endpoints as mentioned in the REST API section below are valid urls, and navigating to those links will return expected output. The REST API follows, with example input and expected output given.

4. REST API:

ENDPOINT: /

- Description: Get homepage HTML
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return homepage HTML
- Example: `curl -X GET http://0.0.0.0:5026/ -H "accept: application/json"`

ENDPOINT: `/api/doc`

- Description: Get API HTML
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return API HTML
- Example: `curl -X GET http://0.0.0.0:5026/api/doc -H "accept: application/json"`

ENDPOINT: `/api/save`

- Description: Get API as rendered string
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return rendered API as string
- Example: `curl -X GET http://0.0.0.0:5026/api/save -H "accept: application/json"`

ENDPOINT: `/country`

- Description: Get all possible countries.
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return a list of countries.
- Example: `curl -X GET http://0.0.0.0:5026/country -H "accept: application/json"` yields:

```
[
  "United_States"
]
```

ENDPOINT: `/country/{country}`

- Description: Get data for a single country.
- Parameters:
  - `country` : Value (name) of country to be queried. An example would be : `United_States`
- Responses:
  - A 200 response will : Return all matching (queried country) sightings as json.
- Example: `curl -X GET http://0.0.0.0:5026/country/United_States -H "accept: application/json"` yields:

```
[
  {
    "city": "Olathe",
    "country": "United_States",
    "duration_minutes": "6",
    "enters": "10 above SSW",
    "exits": "10 above ENE",
    "max_elevation": "28",
```

```
"spacecraft": "ISS",
    "utc_date": "Feb 17, 2022",
    "utc_offset": "-6.0",
    "utc_time": "12:13"
  },
....
  {
    "city": "Nantucket",
    "country": "United_States",
    "duration_minutes": "3",
    "enters": "19 above NNW",
    "exits": "10 above NNE",
    "max_elevation": "19",
    "region": "Massachusetts",
    "sighting_date": "Sat Feb 26/04:56 AM",
    "spacecraft": "ISS",
    "utc_date": "Feb 26, 2022",
    "utc_offset": "-5.0",
    "utc_time": "09:56"
  }
]
```


ENDPOINT: /country/{country}/region

- Description: Get data for all regions of a certain country.
- Parameters:
  - `country` : Value (name) of country to be queried. An example would be : `United_States`
- Responses:
  - A 200 response will : Return all matching regions for the queried country as json.
- Example:
  `curl -X GET http://0.0.0.0:5026/country/United_States/region -H "accept: application/json"`
  yields:


```
[
  "Kansas",
  "Kentucky",
  "Louisiana",
  "Maine",
  "Mariana_Islands",
  "Maryland",
  "Massachusetts"
]
```


ENDPOINT: /country/{country}/region/{region}

- Description: Get all data for a specific region of a certain country.
- Parameters:
  - `country` : Value (name) of country to be queried. An example would be : `United_States`
  - `region` : Value (name) of region to be queried. An example would be : `Kansas`
- Responses:
  - A 200 response will : Return all matching results for the queried region as json.
- Example:
  `curl -X GET http://0.0.0.0:5026/country/United_States/region/Kansas -H "accept: application/j`
  yields:

```
[
  {
    "city": "Olathe",
    "country": "United_States",
    "duration_minutes": "6",
    "enters": "10 above SSW",
    "exits": "10 above ENE",
    "max_elevation": "28",
    "region": "Kansas",
    "sighting_date": "Thu Feb 17/06:13 AM",
    "spacecraft": "ISS",
    "utc_date": "Feb 17, 2022",
    "utc_offset": "-6.0",
    "utc_time": "12:13"
  },
....
  {
    "city": "Yates_Center",
    "country": "United_States",
    "duration_minutes": "1",
    "enters": "12 above N",
    "exits": "10 above N",
    "max_elevation": "12",
    "region": "Kansas",
    "sighting_date": "Sat Feb 26/05:29 AM",
    "spacecraft": "ISS",
    "utc_date": "Feb 26, 2022",
    "utc_offset": "-6.0",
    "utc_time": "11:29"
  }
]
```


ENDPOINT: `/country/{country}/region/{region}/city`

- Description: Get all cities for a specific region of a certain country.
- Parameters:
  - `country` : Value (name) of country to be queried. An example would be : `United_States`
  - `region` : Value (name) of region to be queried. An example would be : `Kansas`
- Responses:
  - A 200 response will : Return all matching cities for the queried region and country as json.
- Example:
  `curl -X GET http://0.0.0.0:5026/country/United_States/region/Kansas/city -H "accept: applicat`
  yields:


```
[
  "Olathe",
  "Osborne",
  "Oskaloosa",
  "Oswego",
  "Ottawa",
  "Paola",
  "Phillipsburg",
```

```
"Russell",
  "Saint_Francis",
  "Saint_John",
  "Salina",
  "Scott_City",
....
  "Sublette",
  "Syracuse",
  "Tallgrass_Prairie_National_Preserve",
  "Topeka",
  "Tribune",
  "Troy",
  "Ulysses",
  "WaKeeny",
  "Washington",
  "Wellington",
  "Westmoreland",
  "Wichita",
  "Winfield",
  "Yates_Center"
]
```

ENDPOINT: /country/{country}/region/{region}/city/{city}

- Description: Get all information for a specific city of a region of a certain country.
- Parameters:
  - `country` : Value (name) of country to be queried. An example would be : `United_States`
  - `region` : Value (name) of region to be queried. An example would be : `Kansas`
  - `city` : Value (name) of city to be queried. An example would be : `Wichita`
- Responses:
  - A 200 response will : Return all information for the queried city as json.
- Example:
  `curl -X GET http://0.0.0.0:5026/country/United_States/region/Kansas/city/Wichita -H "accept:`
  yields:

```
[
  {
    "city": "Wichita",
    "country": "United_States",
    "duration_minutes": "6",
    "enters": "10 above S",
    "exits": "10 above ENE",
    "max_elevation": "25",
    "region": "Kansas",
    "sighting_date": "Thu Feb 17/06:12 AM",
    "spacecraft": "ISS",
    "utc_date": "Feb 17, 2022",
    "utc_offset": "-6.0",
    "utc_time": "12:12"
  },
....
  {
    "city": "Wichita",
    "country": "United_States",
```

```
"exits": "10 above N",
    "max_elevation": "12",
    "region": "Kansas",
    "sighting_date": "Sat Feb 26/05:29 AM",
    "spacecraft": "ISS",
    "utc_date": "Feb 26, 2022",
    "utc_offset": "-6.0",
    "utc_time": "11:29"
  }
]
```


ENDPOINT: `/data`

- Description: Updates the list of data dictionaries.
- Parameters:
  - N/A
- Responses:
  - A 201 response will : Updated data dictionary list.
- Example: `curl -X POST http://0.0.0.0:5026/data -H "accept: application/json"` yields:

```
"Data updated."
```

ENDPOINT: `/epoch`

- Description: Get all possible epochs.
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return a list of epochs.
- Example: `curl -X GET http://0.0.0.0:5026/epoch -H "accept: application/json"` yields:


```
[
  "2022-042T12:00:00.000Z",
  "2022-042T12:04:00.000Z",
  "2022-042T12:08:00.000Z",
  "2022-042T12:12:00.000Z",
  "2022-042T12:16:00.000Z",
  "2022-042T12:20:00.000Z",
  "2022-042T12:24:00.000Z",
  "2022-042T12:28:00.000Z",
  "2022-042T12:32:00.000Z",
  "2022-042T12:36:00.000Z",
  "2022-042T12:40:00.000Z",
  "2022-042T12:44:00.000Z",
  "2022-042T12:48:00.000Z",
  "2022-042T12:52:00.000Z",
....
  "2022-057T11:08:56.869Z",
  "2022-057T11:12:56.869Z",
  "2022-057T11:16:56.869Z",
  "2022-057T11:20:56.869Z",
  "2022-057T11:24:56.869Z",
```

```
"2022-057T11:36:56.869Z",
  "2022-057T11:40:56.869Z",
  "2022-057T11:44:56.869Z",
  "2022-057T11:48:56.869Z",
  "2022-057T11:52:56.869Z",
  "2022-057T11:56:56.869Z",
  "2022-057T12:00:00.000Z"
]
```


ENDPOINT: /epoch/{name}

- Description: Get data for a single epoch.
- Parameters:
  - `name` : Value of epoch to be queried. An example would be : `2022-042T12:04:00.000Z`
- Responses:
  - A 200 response will : Return epoch information for first matching epoch as json.
- Example:
  `curl -X GET http://0.0.0.0:5026/epoch/2022-042T12:04:00.000Z -H "accept: application/json"`
  yields:


```
{
  "EPOCH": "2022-042T12:04:00.000Z",
  "X": {
    "#text": "-4483.2181885642003",
    "@units": "km"
  },
  "X_DOT": {
    "#text": "2.63479158884966",
    "@units": "km/s"
  },
  "Y": {
    "#text": "-4839.4374260438099",
    "@units": "km"
  },
  "Y_DOT": {
    "#text": "-4.3774148889971602",
    "@units": "km/s"
  },
  "Z": {
    "#text": "-1653.1850590663901",
    "@units": "km"
  },
  "Z_DOT": {
    "#text": "5.7014974180323597",
    "@units": "km/s"
  }
}
```


ENDPOINT: /pdf

- Description: Get writeup HTML
- Parameters:
  - N/A
- Responses:
  - A 200 response will : Return writeup HTML
- Example: `curl -X GET http://0.0.0.0:5026/pdf -H "accept: application/json"`

5. Ethical Considerations

Unfortunately, as with any other application, technology lends itself to misuse. Location information such as that returned by this application is a security and privacy risk. For this particular case, there is no privacy risk as the ISS is an inanimate object, however, there is a security risk. As the ISS is an international research station, there is generally no reason for it to suffer harm, but the information provided here can be used for a targeted attack. So please treat the information provided by this application wisely, and do not use it to cause harm, intentionally or unintentionally.

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi: 10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd