Making Multiwavelength Coverage Summaries with `MWLGen`

Joseph R. Farah

June 28, 2018

Abstract

This note will provide documentation for the MWL coverage summary generator `MWLGen`. For questions, comments, or suggestions on how the software or the documentation could be improved, please email me at `joseph.farah@cfa.harvard.edu`. This documentation will be continuously updated at my directory.

1 Terminology

This section will expand upon some of the shorthand that MWLGen uses. This naming scheme is also maintained in the code.

1. "Hub" or $HUB: the folder where all data files (**excluding** EHT and SWIFT) are stored. This folder should only contain station coverage summaries for a **single source**. For details on how to construct these data files, see section (2.1) Ex: `sample_data/SGRA_HUB` or `sample_data/M87_HUB`.
2. "Sum" or $SUM: the folder where all relevant EHT schedules are stored. This folder is **not** source dependent– just throw all the EHT schedules for a run in the folder. They can be obtained from EHT members and require no editing. Ex: `sample_data/EHT_Schedules`
3. "SWIFT" or $SWIFT: the filepath where the SWIFT coverage summary for a particular source is stored. They are taken directly from the array's website and require minimal editing. For details on how to construct these data files, see section (2.2).
4. "Src" or $SRC: the source in question. For now, MWLGen only supports SGRA and M87, but in the near future, it will be able to intelligently determine available sources from the EHT summary files.

2 Data file construction

This will detail how the data files are to be constructed to be easily read into the software. At most, the work required by participating stations is a few minutes in a spreadsheet. The goal is to impose strict but intuitive requirements on the data consolidation to prevent the problems that plagued the old coverage summary generation.

2.1 General construction (excluding SWIFT and EHT)

Each individual station should record time start/end stamps in the following format: [day of year]+[(hour+(min/60)/24)]. For example, 12:30pm on April 18th, 2018 (the 108th day of the year) becomes:

$$\text{[day of year]+[(hour+(min/60)/24)]} = [108] + [(12+(30/60))/24] = [108] + [(12+0.5)/24] = 108.52083$$

Files should be saved with the station name in all caps as a .csv file (ex. Ex: `sample_data/SGRA_HUB/CHANDRA.csv`). The files should be constructed with the first line reading "Start,End" and all subsequent lines formatted with the start and end times separated by a comma. For an example, look at any of the files in the $HUB directories in `sample_data/`. Group all the files into a single folder and name it something relevant.

2.2 SWIFT file construction

SWIFT file construction requires minimal work. The SWIFT data for a source can be obtained from the array site (ex. https://www.swift.psu.edu/operations/obsSchedule.php?t=94007). Note the `php?t=94007`. The number is an ID for the source. Some trivial Googling should be sufficient to provide you with any source ID you need. Once you obtain it, substitute it into the URL and you will have access to the SWIFT coverage data.

Once on the page, use the cursor to select the entire table, copy and then paste into a spreadsheet of your choice. Delete everything except the first two columns, and save the file something relevant, preferably with the source in the name and with a .csv extension. At this point the data is ready to be used.

2.3 EHT file construction

EHT file construction is the easiest of the bunch. Once you have obtained all the summary files for a run (or the year, if you choose), put them all in the folder and remember where it is. That's all you have to do.

3 Generating a MWL coverage summary

Once you have the data set up, download the software and extract it from the .zip. Navigate to the folder and run MWLGen with:

```
python GUI_mkgraph.py
```

Two windows will pop up. The smaller one is a short agreement, which requires you to credit the software and creator on any figures you use generated by MWLGen. To agree to this term and continue using this software, hit the big green "CLICK TO ACKNOWLEDGE THE ABOVE" button.

The larger window is the main interface. The UI is not complicated. On the left, not three buttons and dropdown menu. They are for selecting the hub directory, EHT sum file directory, and SWIFT files. These buttons will pull up a folder/file dialog, and invite you to identify which files you would like to use. The dropdown allows you to choose a source. Immediately to the right is a status indicator. When you submit a directory/file to the MWLGen, the backend will run a rudimentary scan to ensure that the data is formatted properly. If it is satisfied, the status indicator will shift to a bright green and loudly announce that it is ready. If the backend is not convinced by your formatting, it will not change the status of the indicator, and instead let you know which file its having trouble swallowing in the black box below.

Finally, furthest to the right is the "MAKE GRAPH" button. Once all four indicators are green, you are clear to make the graph by hitting the button and waiting 1-5 seconds.

4 Messages

This section will cover the different types of messages the software will return. The backend makes a strong effort to handle all errors internally without crashing, and if it cannot resolve an error on it's own it will try to direct you in solving the problem. A message labeled —UI— can be found in the black text box within the MWLGen UI, whereas a message labeled —T— can be found in the terminal.

4.1 Good messages

The following are good things MWLGen will display for you.

1. —UI—"READY/NOT-READY" indicator flashing green: this indicates the backend is satisfied with the data. This does NOT indicate there are no problems with it. In the future, I will be improving the ability of the backend to detect data problems.
2. —UI—"Examining data file:[data/file/path]; $VAR True; $VAR (excluding SWIFT and EHT) is ready to go": this indicates that the backend is satisfied with the $VAR directory you have provided.

4.2 Bad messages (error messages)

The following are some of the errors you will see.

1. —UI—"Found an improperly formatted file! Problem is with: [PROBLEM TYPE] in file /file/path": the backend has detected a problem with one of your data files. It will attempt to tell you which file the error is in and where in the file the source of the problem is (ex: "Problem is with: [column headers]"). If it can, it will even show you the line in the file that caused it to choke.
2. —UI—"There are no SUM files in this folder! sum False": this tells you that the $SUM folder you provided had zero summary files in it. The line before it will tell you what filepath you added–are you sure that is the one you intended?
3. —T— `"CalledProcessError: Command 'python src/read_swift_script.py /home/joseph/Documents/mwlgen/MWLGen/sample_data/SGRA_HUB /home/joseph/Documents/mwlgen/MWLGen/sample_data/SGRA_HUB/VERITAS.csv' returned non-zero exit status 1"`: this tells you that the graph generation failed. It will tell you the script that failed and the arguments provided. A good first debugging step is to copy and run the exact command in the terminal to see what caused the script to fail. Ex: here you would run this in the terminal:

```
python src/read_swift_script.py /home/joseph/Documents/mwlgen/MWLGen/sample_data/SGRA_HUB
        /home/joseph/Documents/mwlgen/MWLGen/sample_data/SGRA_HUB/VERITAS.csv
```

5 Acknowledgements

Special thanks to Michael Johnson, Sera Markoff, and Mislav Balokovic.