Nebula Net Interactive Feed (NNIF)

Programmer Documentation

Design Team: Jacob Burke, Isabella Cortez, Freddy Lopez, Daniel Willard, Simon Zhao

26FEB2024 v1.8

Document Summary: This document is to archive and document files, methods, and for future developers

Table of Contents

1. Introduction **5**
2. Approved Libraries and Date Approved **5**
3. NNIF Source Files **6**
3.1. Space User Interface (SUI) Module/Component 6
3.1.1.1. Web Stack Code Structure: 6
3.1.1.2. Public/ 6
3.1.1.3. src/ 6
3.1.1.3.1. Components/ 6
3.1.1.3.1.1. Pages/ Contains top-level pages of the website. 6
3.1.1.4. Functionality: 7
3.1.1.5. Technologies Used: 7
3.1.1.6. Programers Notes: 7
3.1.2. moveJSON.py 7
3.1.2.1. Introduction: 7
3.1.2.2. Authorship and Credits: 7
3.1.2.3. Functionality: 7
3.1.2.4. Dependencies: 7
3.1.2.5. Usage: 7
3.1.2.6. Code Documentation: 7
3.1.2.6.1. Function: insert_file_at_line 7
3.2. Photo and Metadata Coalescence - Fits→Notation+PNG (PMC-FNG): 8
3.2.1. Convert.py 8

3.2.1.1. Introduction: 8
3.2.1.2. Authorship and Credits: 8
3.2.1.3. Functionality: 8
3.2.1.4. Dependencies 8
3.2.1.5. Class Structure 9
3.2.1.6. Usage 9
3.2.1.7. Code Documentation 9
3.2.1.7.1. Function: linear_scaling 9
3.2.1.7.2. Function: log_scaling 10
3.2.1.7.3. Function: sqrt_scaling 10
3.2.1.7.4. Function: hist_eq_scaling 10
3.2.1.7.5. Function: asinh_scaling 11
3.2.1.7.6. Function: process_fits 11
3.2.1.7.7. Function: visualize_fits 11
3.2.1.7.8. Function: compare_scaling_methods 12
3.2.1.7.9. Function: append_metadata_to_json 12
3.2.1.7.10. Function: convert_to_png 12
3.2.1.7.11. Function: rename 13
3.2.2. main.py 13
3.2.2.1. Introduction: 13
3.2.2.2. Authorship and Credits 13
3.2.2.3. Functionality 13
3.2.2.4. Dependencies 13
3.2.2.5. Usage 13
3.2.2.6. Code Documentation 13
3.2.2.6.1. Function: main 13
3.2.3. mast_query.py 14
3.2.3.1. Introduction: 14
3.2.3.2. Authorship and Credits: 14
3.2.3.3. Functionality: 14
3.2.3.4. Dependencies: 14
3.2.3.5. Usage: 15
3.2.3.6. Code Documentation 15
3.2.3.6.1. Function: __init 15
3.2.3.6.2. Function: mast_auth 15
3.2.3.6.3. Function: connect_sqlite3 15
3.2.3.6.4. Function: disconnect_from_db 16
3.2.3.6.5. Function: convert_mjd_to_datetime 16
3.2.3.6.6. Function: fetch_from_sql_db 16
3.2.3.6.7. Function: fetch_and_segment_by_week 16
3.2.3.6.8. Function: clean_instrument_name 17

3.2.3.6.9. Function: process_individual_observation 17
3.2.3.6.10. Function: file_exist 17
3.2.3.6.11. Function: convert_numpy 18
3.2.3.6.12. Function: process_weekly_observations 18
3.2.3.6.13. Function: query_mast 18
3.2.3.6.14. Function: select_best_fits 19
3.2.3.6.15. Function: extract_and_store_fits_metadata 19
3.2.3.6.16. Function: process_all_rows_from_db 19
3.2.3.6.17. Function: filter_files 20
3.2.3.6.18. Function: stream_fits_data 20
3.2.3.6.19. Function: get_fits_uris 20
3.2.3.6.20. Function: download_specific_fits 21
3.2.3.6.21. Function: combine 21
3.2.3.6.22. Function: process_all_rows_from_db 21
3.2.3.6.23. Function: filter_files 22
3.2.3.6.24. Function: stream_fits_data 22
3.2.3.6.25. Function: get_fits_uris 22
3.2.3.6.26. Function: download_specific_fits 23
3.2.3.6.27. Function: combine 23
3.3. MAST Validator (MASTv) Module/Component 24
3.3.1. mast_query_test.py 24
3.3.1.1. Introduction: 24
3.3.1.2. Authorship and Credits 24
3.3.1.3. Functionality 24
3.3.1.4. Dependencies 24
3.3.1.5. Usage 24
3.3.1.6. Code Documentation 24
3.3.1.6.1. Function: mast_query 24
3.3.1.6.2. Function: test_mast_auth_success 24
3.3.1.6.3. Function: test_connect_sqlite3_success 25
3.3.1.6.4. Function: test_disconnect_from_db 25
3.3.1.6.5. Function: test_fetch_from_sql_db_success 25
3.3.1.6.6. Function: test_convert_mjd_to_datetime 25
3.3.1.6.7. Progamer Notes & DocStrings 25
3.4. Mission Information Gatherer (MIG) 25
3.4.1. jwstDataFinder.py 25
3.4.1.1. Introduction: 25
3.4.1.2. Authorship and Credits: 26
3.4.1.3. Functionality: 26
3.4.1.4. Dependencies: 26
3.4.1.5. Usage: 26

3.4.1.6. Code Documentation: 26
3.4.1.6.1. Function: create_session() 26
3.4.1.6.2. Function: scrape_jwst_data(session, u) 26
3.4.1.6.3. Function: write_to_txt(jwst_data_list, output_file='jwst_data.txt') 27
3.4.1.6.4. Function: return_jwst_data(session) 27
3.4.1.6.5. Function: get_jwst_as_py_list() 28
3.4.2. jwstDatabase.py 28
3.4.2.1. Introduction: 28
3.4.2.2. Authorship and Credits: 28
3.4.2.3. Functionality: 28
3.4.2.4. Dependencies: 28
3.4.2.5. Usage: 29
3.4.2.6. Code Documentation: 29
3.4.2.6.1. Function: with open('jwst_data.json') as file: 29
3.4.2.6.2. Function: sqlite3.connect('jwstDatabaseFile.sqlite') 29
3.4.2.6.3. Function: CREATE TABLE IF NOT EXISTS jwst_data (...) 29
3.4.2.6.4. Function: INSERT INTO jwst_data VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?) 30
3.4.2.6.5. Function: conn.commit() 30
3.4.2.6.6. Function: conn.close() 30
3.4.3. jwstJason.py 30
3.4.3.1. Introduction: 30
3.4.3.2. Authorship and Credits: 30
3.4.3.3. Functionality: 31
3.4.3.4. Dependencies: 31
3.4.3.5. Usage 31
3.4.3.6. Code Documentation 31
3.4.3.6.1. Function: parse_txt_to_json(txt_file) 31
3.4.3.6.2. Function: json.dump(visit_info_list, json_file, indent=2) 32
3.5. Web Hosting Component (University of Oregon Server Static Hosting) 32
3.5.1. Install_script_linux.py 32
3.5.1.1. Introduction: 32
3.5.1.3. Functionality: 32
3.5.1.4. Dependanieces: 32
3.5.1.5. Usage: 33
3.5.1.6. Code Documentation: 33
3.5.2. Run_script_linux.py 33
3.5.2.1. Introduction: 33
3.5.2.2. Authorship and Credits: 34
3.5.2.3. Functionality: 34
3.5.2.4. Dependencies: 34
3.5.2.5. Usage: 34

3.5.2.6. Code Documentation: 34

1. Introduction

- This document is the comprehensive programmer documentation for Nebula Net Interactive Feed (NNIF), a cutting-edge web application meticulously crafted to curate and display captivating images captured by the James Webb Space Telescope (JWST). This meticulously curated documentation serves as a definitive resource for developers, providing detailed insights into the architecture, implementation, and functionality of the NNIF project. With comprehensive descriptions of files, functions, and dependencies, this document aims to streamline development workflows, facilitate collaboration, and ensure the project's longevity and maintainability.

2. Approved Libraries and Date Approved

2.1. To uphold the highest standards of quality and performance, NNIF relies on a vetted selection of libraries and dependencies. The project proudly embraces the following technologies:

2.1.1. **15FEB2024:**

- Python Standard Libraries

2.1.2. **22FEB2024**:

- **MAST AP**I: This API provides access to the NASA James Webb Space Telescope database, facilitating the retrieval of photos and related data.
- **Astro query / Astro py**: Essential for handling .fits (Flexible Image Transport System) files commonly used in astronomy. These libraries enable the conversion of .fits files to the more accessible .png (Portable Network Graphic) format for display purposes.
- **MatPlotLib**: Utilized for transforming .fit files into 2D arrays, subsequently facilitating the conversion process to .png files. Matplotlib ensures scalability and enables zoomable image displays.
- **BeautifulSoup**: This library plays a crucial role in scraping photo data from NASA's website, enabling the acquisition of comprehensive photo

sets from past missions and up-to-date information on the current mission.

- **requests**: Necessary for establishing connections to websites from which data is collected.

2.1.3. **07MAR2024**:

- **Padas:** Integrated for querying image metadata stored in a SQLite database, identified as a vital component for the project's functionality.
- **NumPy**: Employed for image scaling, refining, and final conversion steps, enhancing the visual quality and usability of images.
- **PyTest:** Utilized for automated testing, filling the need for a robust testing framework to ensure the reliability and integrity of the NNIF application.

3. NNIF Source Files

3.1. Space User Interface (SUI) Module/Component

3.1.1.1. Web Stack Code Structure:

3.1.1.2. Public/

- Contains images, music files, and JSON files. This is the folder that contains all the files for the display objects on the page.

3.1.1.3. src/

- **components/:** Contains React components used throughout the website.
- **scripts/:** these are the scripts needed to populate data from files to the webpage.
- **App.css**: Contains CSS and SCSS files for styling the website.
- **App.js**: The main component that renders the entire application.
- **index.js**: Entry point of the application, renders the App component.

3.1.1.3.1. Components/

3.1.1.3.1.1. Pages/ Contains top-level pages of the website.

- AboutDes.css: Styles for the AboutDes component.
- AboutDes.js: Component for the about description page.
- Arrow.js: Component for navigation arrows.
- Button.css: Styles for buttons.
- Button.js: Component for buttons.
- Dots.js: Component for carousel pagination dots.
- Footer.css: Styles for the footer component.
- Landing.css: Styles for the landing page.
- Landing.js: Component for the landing page.
- NavBar.css: Styles for the navigation bar.
- NavBar.js: Component for the navigation bar.
- Slider.js: Component for the slider.
- SliderContent.js: Component for slider content.
- SourcesJS.css: Styles for the sources component.

  - SoursesJS.js: Component for the sources page.
  - carousel.js: Script for the carousel functionality.
  - Pic1.jpg: Image file for the landing page.
  - Pics.json: JSON file containing image data.
  - Slider.css: Styles for the slider component.
  - Test.jsx: Main carousel component for images and metadata
  - Test.css: styling for Test,jsx regarding metadata benign displayed

3.1.1.4. Functionality:

- **Navigation**: Users can navigate between different pages using the header navigation.

3.1.1.5. Technologies Used:

- React.js: Frontend JavaScript library for building user interfaces.
- CSS / SCSS: Styling the components and layout of the website.

3.1.1.6. Programers Notes:

- Utilize reusable components wherever possible to maintain code
- efficiency and scalability.

3.1.2. moveJSON.py

3.1.2.1. Introduction:

- This provides comprehensive documentation for the insert_file_at_line function implemented in the moveJSON.py script. The function facilitates the insertion of the contents of one file into another file at a specified line number..

3.1.2.2. Authorship and Credits:

- **Author**: Freddy Lopez

3.1.2.3. Functionality:

- The write_file_from_another function reads the contents of two files: the original file and the file to be inserted. It then inserts the contents of the insert file into the original file overwriting any contents. The function utilizes Python's file handling capabilities to read and write file contents efficiently.

3.1.2.4. Dependencies:

- Sys

3.1.2.5. Usage:

- To utilize the write_file_from_another function, the script moveJSON.py should be invoked from the command line with two arguments: the path to the original file, the path to the file to be inserted.
- Example:
  - python3 moveJSON.py <original_file> <insert_file>

3.1.2.6. Code Documentation:

3.1.2.6.1. Function: insert_file_at_line

3.1.2.6.1.1. Parameters:

- **original_file:** A string representing the path of the original file.
- **insert_file**: A string representing the path of the file to be inserted.

3.1.2.6.1.2. Description:

- The insert_file_at_line function reads the contents of the original file and the insert file into memory. It then inserts the contents of the insert file into the original file. Finally, it writes the modified contents back to the original file.

3.1.2.6.1.3. **Returns**:

- NONE

3.1.2.6.1.4. Docstrings and Programers notes:

- The script lacks explicit docstrings, which would provide detailed documentation directly within the code. Including docstrings would enhance the script's readability and maintainability by providing clear descriptions of the function's purpose, parameters, and expected behavior. Additionally, error handling could be improved to handle edge cases such as invalid file paths.

3.2. Photo and Metadata Coalescence - Fits→Notation+PNG (PMC-FNG):

3.2.1. Convert.py

3.2.1.1. Introduction:

- convert.py is a Python script designed to automate the processes involved in handling Flexible Image Transport System (FITS) files obtained from the Mikulski Archive for Space Telescopes (MAST) database. The script facilitates various operations such as reading FITS files, applying different scaling methods to enhance image contrast, converting processed image data into PNG format, and visualizing the comparison of scaling methods.

3.2.1.2. Authorship and Credits:

- **Author**: Simon Zhao

3.2.1.3. Functionality:

3.2.1.3.1. The script performs the following tasks:

- Reading FITS files to extract astronomical image data.
- Applying different scaling methods (linear, logarithmic, square root, histogram equalization, and asinh scaling) to enhance image contrast and detail visibility.

- Converting processed image data into PNG format.
- Visual comparison of scaling methods to help with selecting the most appropriate technique for specific data sets.

3.2.1.4. Dependencies

3.2.1.4.1. The script relies on the following Python libraries:

- astropy.io: For reading FITS files.
- numpy: For numerical operations.
- matplotlib.pyplot: For plotting and visualization.
- astropy.visualization: For image visualization and normalization.
- os: For operating system-related functions.
- logging: For logging messages.
- json: For JSON file handling.

3.2.1.5. Class Structure

- The script defines a class named Processing, which encapsulates methods for processing FITS files, applying scaling methods, visualizing data, comparing scaling methods, and managing metadata.

3.2.1.6. Usage

- The script can be utilized by creating an instance of the Processing class and calling its methods with appropriate parameters. Specific functionalities include processing FITS files (process_fits method), visualizing processed data (visualize_fits method), comparing scaling methods (compare_scaling_methods method), and converting processed data to PNG format (convert_to_png method).

3.2.1.6.1. Example:

- processor = Processing()
- processor.process_fits("example.fits", scaling_method="linear")
- processor.visualize_fits(image_data)
- processor.compare_scaling_methods("example.fits", target_name="Example", instrument="Instrument", start_date="2024-03-10")
- processor.convert_to_png(processed_data, output_filename="output_image.png", switch=True)

3.2.1.7. Code Documentation

3.2.1.7.1. Function: linear_scaling

3.2.1.7.1.1. Parameters:

- data (np.ndarray): Input data array to be scaled.
- scale_min (float, optional): Minimum value for scaling. If not provided, the minimum value of the input data will be used.

- scale_max (float, optional): Maximum value for scaling. If not provided, the maximum value of the input data will be used.

3.2.1.7.1.2. Description:

- This function performs linear scaling on the input data array to the range [0, 1].

3.2.1.7.1.3. Returns:

- np.ndarray: Scaled data with values between 0 and 1.

3.2.1.7.1.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.2. Function: log_scaling

3.2.1.7.2.1. Parameters:

- data (np.ndarray): Input data array to be scaled.
- scale_min (float, optional): Minimum value for scaling. If not provided, the minimum value of the input data will be used.
- scale_max (float, optional): Maximum value for scaling. If not provided, the maximum value of the input data will be used.

3.2.1.7.2.2. Description:

- This function applies logarithmic scaling to the input data.

3.2.1.7.2.3. Returns:

- np.ndarray: Logarithmically scaled data.

3.2.1.7.2.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.3. Function: sqrt_scaling

3.2.1.7.3.1. Parameters:

- data (np.ndarray): Input data array to be scaled.
- scale_min (float, optional): Minimum value for scaling. If not provided, the minimum value of the input data will be used.
- scale_max (float, optional): Maximum value for scaling. If not provided, the maximum value of the input data will be used.
- percentiles (tuple): Percentile values to compute dynamic scale_min and scale_max if not provided.

3.2.1.7.3.2. Description:

- This function applies square root scaling to the input data with percentile-based dynamic range adjustment.

3.2.1.7.3.3. Returns:

- np.ndarray: Square root scaled data.

3.2.1.7.3.4. Progamer Notes & DocStings:
- Repeat of the data above

3.2.1.7.4. Function: hist_eq_scaling

3.2.1.7.4.1. Parameters:
- data (np.ndarray): Input data array for histogram equalization.

3.2.1.7.4.2. Description:
- Apply histogram equalization scaling to the data.

3.2.1.7.4.3. Returns:
- np.ndarray: Data scaled using histogram equalization.

3.2.1.7.4.4. Progamer Notes & DocStings:
- Repeat of the data above

3.2.1.7.5. Function: asinh_scaling

3.2.1.7.5.1. Parameters:
- data (np.ndarray): Input data array to be scaled.
- scale_min (float, optional): Minimum value for scaling.
- scale_max (float, optional): Maximum value for scaling.
- non_linear (float): Non-linearity factor for asinh scaling.

3.2.1.7.5.2. Description:
- Apply asinh scaling to the data.

3.2.1.7.5.3. Returns:
- np.ndarray: Data scaled using asinh function.

3.2.1.7.5.4. Progamer Notes & DocStings:
- Repeat of the data above

3.2.1.7.6. Function: process_fits

3.2.1.7.6.1. Parameters:
- fits_path (str): Path to the FITS file.
- scaling_method (str): The scaling method to use ("asinh", "linear", "log", "sqrt", "hist_eq").
- **kwargs: Additional arguments to pass to the scaling function.

3.2.1.7.6.2. Description:
- Processes a FITS file and return scaled image data using the specified scaling method.

3.2.1.7.6.3. Returns:
- np.ndarray: The scaled image data.

3.2.1.7.6.4. Progamer Notes & DocStings:
- Repeat of the data above

3.2.1.7.7. Function: visualize_fits

3.2.1.7.7.1. Parameters:

- image_data (np.ndarray): The image data to be visualized.
- frame (int): Index of the frame to visualize (default is 0).

3.2.1.7.7.2. Description:

- This function visualizes processed FITS data. It displays the image data using Matplotlib, allowing for visualization of astronomical images.

3.2.1.7.7.3. Returns:

- NONE

3.2.1.7.7.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.8. Function: compare_scaling_methods

3.2.1.7.8.1. Parameters:

- fits_path (str): Path to the FITS file.
- target_name (str): Name of the target.
- instrument (str): Name of the instrument.
- start_date (str): Date when the observation started.

3.2.1.7.8.2. Description:

- This function compares different scaling methods by visualizing them in a single plot. It provides a side-by-side comparison of the effects of different scaling techniques on image contrast and visibility.

3.2.1.7.8.3. Returns:

- NONE

3.2.1.7.8.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.9. Function: append_metadata_to_json

3.2.1.7.9.1. Parameters:

- metadata (dict): Metadata of a particular observation in dictionary format.
- json_filename (str): Filename of the JSON file.

3.2.1.7.9.2. Description:

- This function adds observation metadata into a JSON file. It appends the metadata to an existing JSON file or creates a new one if it doesn't exist.

3.2.1.7.9.3. Returns:

- NONE

3.2.1.7.9.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.10. Function: convert_to_png

3.2.1.7.10.1. Parameters:

- data (np.ndarray): Image data to be saved as a PNG.

- output_filename (str): Filename for the saved PNG file.
- switch (bool): Indicates whether to include a colorbar in the PNG image (default is False).

3.2.1.7.10.2. Description:

- This function converts processed FITS data to PNG format and saves them to a directory. It provides an option to include a colorbar in the PNG image.

3.2.1.7.10.3. Returns:

- NONE

3.2.1.7.10.4. Progamer Notes & DocStings:

- Repeat of the data above

3.2.1.7.11. Function: rename

3.2.1.7.11.1. Parameters:

- NONE

3.2.1.7.11.2. Description:

- This function is intended to rename files. However, it currently doesn't have an implementation.

3.2.1.7.11.3. Returns:

- NONE

3.2.1.7.11.4. Progamer Notes & DocStings:

-

3.2.2. main.py

3.2.2.1. Introduction:

- This script serves as the main entry point for executing a series of tasks utilizing the MastQuery and Processing classes.

3.2.2.2. Authorship and Credits

- **Author:** Simon Zhao

3.2.2.3. Functionality

3.2.2.3.1. The script performs the following tasks:

- Imports the MastQuery class from mast_query module.
- Executes a series of operations using MastQuery and Processing classes.
- Reports the execution time of the script.

3.2.2.4. Dependencies

- mast_query module
- time module

3.2.2.5. Usage

- Execute this script to perform operations related to querying and processing astronomical data.

3.2.2.5.1. Example

- python main.py

3.2.2.6. Code Documentation

3.2.2.6.1. **Function: main**

3.2.2.6.1.1. Parameters:
- NONE

3.2.2.6.1.2. Description:
- The main function of the script. It orchestrates the execution of various tasks related to querying and processing astronomical data.

3.2.2.6.1.3. Returns:
- NONE

3.2.2.6.1.4. Progamer Notes & DocStings:
- This function initializes necessary components, performs authorization, fetches and segments data by week, processes weekly observations, and reports the execution time.

3.2.3. mast_query.py

3.2.3.1. Introduction:

- The mast_query.py file is a Python script designed to interact with the MAST (Mikulski Archive for Space Telescopes) database to query and retrieve observations related to specific astronomical targets, particularly those made by the James Webb Space Telescope (JWST). It provides functionalities for authentication, retrieval, filtering, and downloading of FITS files corresponding to the observations made by JWST instruments. Additionally, the script includes methods to segment observational data by week and process individual observations for extracting metadata.

3.2.3.2. Authorship and Credits:

- **Author:** Simon Zhao

3.2.3.3. Functionality:

3.2.3.3.1. The script offers the following functionalities:
- Authentication to access the MAST database.
- Establishment of a connection to a SQLite database.
- Conversion of MJD (Modified Julian Date) format to ISO format.
- Execution of queries on the SQLite database and retrieval of results as DataFrames.
- Fetching and segmentation of data by week from the SQLite database.
- Cleaning of instrument names.
- Processing of individual observations, including querying the MAST database for metadata and selecting the best FITS file.
- Streaming FITS data from the MAST database.
- Downloading specific FITS files.

- Combining FITS URIs with the MAST URL.

3.2.3.4. Dependencies:

3.2.3.4.1. The script relies on the following dependencies:

- astroquery.mast: For querying the MAST database.
- astropy: For handling astronomical data and time formats.
- pandas: For data manipulation and analysis.
- numpy: For numerical computing.
- sqlite3: For interacting with SQLite databases.
- requests: For making HTTP requests.
- os.path: For handling file paths.
- logging: For logging messages and errors.
- pprint: For pretty-printing data structures.

3.2.3.5. Usage:

- The script can be used to query and retrieve observational data from the MAST database, process it, and download relevant FITS files. It provides various methods for interacting with the database, segmenting data, processing observations, and handling FITS files.

3.2.3.5.1. Example:

- python main.py

3.2.3.6. Code Documentation

3.2.3.6.1. Function: __init

3.2.3.6.1.1. Parameters:

- download_dir (str): The directory where downloaded files will be saved. Default is "processed_png".

3.2.3.6.1.2. Description:

- Initializes the MastQuery object with a specified download directory.

3.2.3.6.1.3. Returns:

- NONE

3.2.3.6.1.4. Progamer Notes & DocStings:

- This function initializes the MastQuery object with a default download directory of "processed_png" if not specified.

3.2.3.6.2. Function: mast_auth

3.2.3.6.2.1. Parameters:

- token (str): The Mast API token for authentication.

3.2.3.6.2.2. Description:

- Authenticates access to the MAST database using the provided API token.

3.2.3.6.2.3. Returns:

- None

3.2.3.6.2.4. Progamer Notes & DocStings:

- This function performs authentication to access the MAST database using the provided API token.

3.2.3.6.3. Function: connect_sqlite3

3.2.3.6.3.1. Parameters:
- db_path (str): The path to the SQLite database.

3.2.3.6.3.2. Description:
- Establishes a connection to the SQLite database.

3.2.3.6.3.3. Returns:
- bool: True if connection is successful, False otherwise.

3.2.3.6.3.4. Progamer Notes & DocStings:
- This function establishes a connection to the SQLite database located at the specified path.

3.2.3.6.4. Function: disconnect_from_db

3.2.3.6.4.1. Parameters:
- NONE

3.2.3.6.4.2. Description:
- Closes connection to the SQLite database.

3.2.3.6.4.3. Returns:
- NONE

3.2.3.6.4.4. Progamer Notes & DocStings:
- This function closes the connection to the SQLite database if it is open.

3.2.3.6.5. Function: convert_mjd_to_datetime

3.2.3.6.5.1. Parameters:
- mjd (Time): Time in Modified Julian Date (MJD) format.

3.2.3.6.5.2. Description:
- Converts Modified Julian Date (MJD) format to ISO format.

3.2.3.6.5.3. Returns:
- Time: Time in ISO format.

3.2.3.6.5.4. Progamer Notes & DocStings:
- This function converts a given time in Modified Julian Date (MJD) format to ISO format.

3.2.3.6.6. Function: fetch_from_sql_db

3.2.3.6.6.1. Parameters:
- db_path (str): Path to the SQLite database.

3.2.3.6.6.2. Description:
- Executes a query on the SQLite database and returns results as a DataFrame.

3.2.3.6.6.3. Returns:
- pd.DataFrame: DataFrame containing the results of the query.

3.2.3.6.6.4. Progamer Notes & DocStings:

- This function executes a query on the SQLite database located at the specified path and returns the results as a DataFrame.

3.2.3.6.7. Function: fetch_and_segment_by_week

3.2.3.6.7.1. Parameters:

- db_path (str): Path to the SQLite database.

3.2.3.6.7.2. Description:

- Fetches all data from the SQLite database and segments it by week.

3.2.3.6.7.3. Returns:

- list: A list of DataFrames, each representing a week's worth of observation data.

3.2.3.6.7.4. Progamer Notes & DocStings:

- This function fetches all data from the SQLite database located at the specified path and segments it by week, returning a list of DataFrames.

3.2.3.6.8. Function: clean_instrument_name

3.2.3.6.8.1. Parameters:

- full_instrument_name (str): The full name of the instrument including mode.

3.2.3.6.8.2. Description:

- Removes any additional words from the instrument name.

3.2.3.6.8.3. Returns:

- str: The cleaned instrument name.

3.2.3.6.8.4. Progamer Notes & DocStings:

- This function takes the full name of the instrument (including mode) and removes any additional words, returning the cleaned instrument name.

3.2.3.6.9. Function: process_individual_observation

3.2.3.6.9.1. Parameters:

- observation_row: Dictionary containing observation data.
- week_count (int): Track weekly count.

3.2.3.6.9.2. Description:

- Queries MAST database for an individual observation and processes it.

3.2.3.6.9.3. Returns:

- dict or None: Metadata for the observation if found, None otherwise.

3.2.3.6.9.4. Progamer Notes & DocStings:

- This function queries the MAST database for an individual observation using the provided data, processes it, and returns the metadata if found.

3.2.3.6.10. Function: file_exist

3.2.3.6.10.1. Parameters:
- path (str): File path for PNG images.

3.2.3.6.10.2. Description:
- Checks if a file exists in the specified path.

3.2.3.6.10.3. Returns:
- bool: True if the file exists, False otherwise.

3.2.3.6.10.4. Progamer Notes & DocStings:
- This function checks if a file exists at the specified path and returns True if it does, False otherwise.

3.2.3.6.11. Function: convert_numpy

3.2.3.6.11.1. Parameters:
- obj: Object to convert.

3.2.3.6.11.2. Description:
- Recursively converts numpy data types to their native Python equivalents.

3.2.3.6.11.3. Returns:
- The converted object.

3.2.3.6.11.4. Progamer Notes & DocStings:
- This function recursively converts numpy data types to their native Python equivalents.

3.2.3.6.12. Function: process_weekly_observations

3.2.3.6.12.1. Parameters:
- weekly_dataframes (list): List of DataFrames, each representing a week's worth of observation data.
- start_week (int, optional): The starting week for processing. Defaults to 1.

3.2.3.6.12.2. Description:
- Processes observations segmented by week.

3.2.3.6.12.3. Returns:
- NONE

3.2.3.6.12.4. Progamer Notes & DocStings:
- This function processes observations segmented by week, starting from the specified week number (default is 1).

3.2.3.6.13. Function: query_mast

3.2.3.6.13.1. Parameters:
- target (str): The name of the target object to query.
- instrument (str): The name of the instrument used for the observation.

- category (str): The category or classification of the target.
- keywords (str): Keywords associated with the observation.

3.2.3.6.13.2. Description:

- Performs a query into MAST database for JWST observations based on target information and stores relevant metadata.

3.2.3.6.13.3. Returns:

- dict or None: Metadata for the observation if found, None otherwise.

3.2.3.6.13.4. Progamer Notes & DocStings:

- This function performs a query into the MAST database for JWST observations based on target information and stores relevant metadata.

3.2.3.6.14. Function: select_best_fits

3.2.3.6.14.1. Parameters:

- data_products (pd.DataFrame): DataFrame containing data product info.
- max_mb_size (float, optional): Maximum size of FITS file in MB. Defaults to 500.0.
- min_mb_size (float, optional): Minimum size of FITS file in MB. Defaults to 15.0.

3.2.3.6.14.2. Description:

- Selects the best FITS file based on calibration level, stage 3 product types, and file size.

3.2.3.6.14.3. Returns:

- str or None: URI of the best FITS file, or None if no suitable file is found.

3.2.3.6.14.4. Progamer Notes & DocStings:

- This function selects the best FITS file from the given DataFrame based on calibration level, stage 3 product types, and file size constraints.

3.2.3.6.15. Function: extract_and_store_fits_metadata

3.2.3.6.15.1. Parameters:

- selected_fits (dict): Selected FITS file info.
- obs_table (list): List of observations with each as a dictionary.

3.2.3.6.15.2. Description:

- Extracts metadata from a selected FITS file and its corresponding metadata and stores it.

3.2.3.6.15.3. Returns:

- NONE

3.2.3.6.15.4. Progamer Notes & DocStings:

- This function extracts metadata from a selected FITS file and its corresponding metadata and stores it for further processing.

3.2.3.6.16. Function: process_all_rows_from_db

3.2.3.6.16.1. Parameters:
- dataframe (pd.DataFrame): DataFrame containing the data.
- week_count (int): Count of the current week being processed.

3.2.3.6.16.2. Description:
- Processes each row from the DataFrame.

3.2.3.6.16.3. Returns:
- NONE

3.2.3.6.16.4. Progamer Notes & DocStings:
- This function iterates through each row of the DataFrame and processes individual observations for the given week.

3.2.3.6.17. Function: filter_files

3.2.3.6.17.1. Parameters:
- product_list (list): List of product dictionaries from data_products.

3.2.3.6.17.2. Description:
- Filters FITS files ending in "_i2d.fits" or "_s2d.fits" or "_cal.fits" or "_calints.fits".

3.2.3.6.17.3. Returns:
- list or None: List of filtered fits files if found, None otherwise.

3.2.3.6.17.4. Progamer Notes & DocStings:
- This function filters the given list of product dictionaries to select FITS files ending with specified extensions.

3.2.3.6.18. Function: stream_fits_data

3.2.3.6.18.1. Parameters:
- data_uri (str): The URI of data to stream.

3.2.3.6.18.2. Description:
- Streams FITS data from the MAST database.

3.2.3.6.18.3. Returns:
- list or None: The HDU list object loaded directly from the streamed FITS data if successful, None otherwise.

3.2.3.6.18.4. Progamer Notes & DocStings:
- This function streams FITS data from the specified data URI and returns the HDU list object if successful.

3.2.3.6.19. Function: get_fits_uris

3.2.3.6.19.1. Parameters:
- data_products (list): The list of data products from the MAST query.
- file_endings (tuple, optional): A tuple of strings with file endings to look for. Defaults to ("_i2d.fits", "_s2d.fits").

3.2.3.6.19.2. Description:
- Gets URIs for FITS files from data products.

3.2.3.6.19.3. Returns:
- list: A list of URIs for the FITS files.

3.2.3.6.19.4. Progamer Notes & DocStings:
- This function retrieves URIs for FITS files from the provided list of data products based on specified file endings.

3.2.3.6.20. Function: download_specific_fits

3.2.3.6.20.1. Parameters:
- fits_filename (str): The filename of the FITS file to download.

3.2.3.6.20.2. Description:
- Downloads a specific FITS file by filename and saves it to the specified download directory.

3.2.3.6.20.3. Returns:
- NONE

3.2.3.6.20.4. Progamer Notes & DocStings:
- This function is used for testing purposes to download a specific FITS file by filename and save it to the specified directory.

3.2.3.6.21. Function: combine

3.2.3.6.21.1. Parameters:
- fits_uri (str): The FITS file name.

3.2.3.6.21.2. Description:
- Combines FIT URI with the MAST URL and adds it to the dictionary.

3.2.3.6.21.3. Returns:
- str: The combined URL for the FITS file.

3.2.3.6.21.4. Progamer Notes & DocStings:
- This function takes a FITS URI, combines it with the MAST URL, and returns the full URL for the FITS file.
- The combined URL is added to the fits_URIs dictionary for reference.

3.2.3.6.22. Function: process_all_rows_from_db

3.2.3.6.22.1. Parameters:

- dataframe (pd.DataFrame): The DataFrame containing the data.
- week_count (int): The week count for processing.

3.2.3.6.22.2. Description:

- Processes each row from the DataFrame.

3.2.3.6.22.3. Returns:

- NONE

3.2.3.6.22.4. Progamer Notes & DocStings:

- This function iterates over each row in the provided DataFrame and processes each observation individually for the specified week.

3.2.3.6.23. Function: filter_files

3.2.3.6.23.1. Parameters:

- product_list (list): List of product dictionaries from data_products.

3.2.3.6.23.2. Description:

- Filters FITS files ending in "_i2d.fits", "_s2d.fits", "_cal.fits", or "_calints.fits".

3.2.3.6.23.3. Returns:

- list or None: List of filtered FITS files if found, None otherwise.

3.2.3.6.23.4. Progamer Notes & DocStings:

- This function filters the provided list of product dictionaries to include only FITS files with specific endings.
- Returns the filtered list of FITS files if found, None otherwise.

3.2.3.6.24. Function: stream_fits_data

3.2.3.6.24.1. Parameters:

- data_uri (str): The URI of data to stream.

3.2.3.6.24.2. Description:

- Streams FITS data from the MAST database.

3.2.3.6.24.3. Returns:

- list or None: The HDU list object loaded directly from the streamed FITS data, or None if an error occurs.

3.2.3.6.24.4. Progamer Notes & DocStings:

- This function streams FITS data from the MAST database using the provided data URI.
- It returns the HDU list object loaded directly from the streamed FITS data if successful, otherwise returns None.

- Handles exceptions such as RequestException and other general exceptions gracefully, logging any errors that occur.

3.2.3.6.25. Function: get_fits_uris

3.2.3.6.25.1. Parameters:
- data_products (list): The list of data products from the MAST query.
- file_endings (tuple): A tuple of strings with file endings to look for. Default value is ("_i2d.fits", "_s2d.fits").

3.2.3.6.25.2. Description:
- Gets URIs for FITS files from data products.

3.2.3.6.25.3. Returns:
- list: A list of URIs for the FITS files.

3.2.3.6.25.4. Progamer Notes & DocStings:
- This function retrieves URIs for FITS files from the provided list of data products, filtering by file endings specified in file_endings.
- Returns a list of URIs for the FITS files found in the data products.

3.2.3.6.26. Function: download_specific_fits

3.2.3.6.26.1. Parameters:
- fits_filename (str): The filename of the FITS file to download.

3.2.3.6.26.2. Description:
- Downloads a specific FITS file by filename and saves it to the specified download directory.

3.2.3.6.26.3. Returns:
- NONE

3.2.3.6.26.4. Progamer Notes & DocStings:
- This function is primarily used for testing purposes.
- It constructs the download URL for the FITS file using the provided filename and MAST URL.
- The FITS file is downloaded and saved to the specified download directory.
- Handles exceptions such as RequestException and other general exceptions gracefully, logging any errors that occur.

3.2.3.6.27. Function: combine

3.2.3.6.27.1. Parameters:
- fits_uri (str): The fits file name.

3.2.3.6.27.2. Description:
- Combines FIT URI with the MAST URL and adds it to the dictionary.

3.2.3.6.27.3. Returns:

- str: The combined URL for the FITS file.

3.2.3.6.27.4. Progamer Notes & DocStings:

- This function constructs the full URL for the FITS file by combining the provided FIT URI with the MAST URL.
- The combined URL is added to the fits_URIs dictionary attribute.
- Returns the combined URL for the FITS file.
- Handles cases where the FITS URI is None gracefully, logging an error message and returning an empty string.

3.3. MAST Validator (MASTv) Module/Component

3.3.1. mast_query_test.py

3.3.1.1. Introduction:

- This file provides tests for the methods within the MastQuery class.

3.3.1.2. Authorship and Credits

- **Author:** Simon Zhao

3.3.1.3. Functionality

- This file contains test cases to verify the functionality of methods within the MastQuery class, which is responsible for querying the MAST (Mikulski Archive for Space Telescopes) database.

3.3.1.4. Dependencies

- Sys
- pathlib.Path
- Os
- Pytest
- Unittest.mock.patch
- unittest.mock.MagicMock
- Pandas
- astropy.time.Time

3.3.1.5. Usage

- This file is used to run test cases for the MastQuery class. It should be executed using a testing framework like pytest.

3.3.1.5.1. Example

- An example usage of this file would be to run the test cases using a command like:
- pytest mast_query_test.py

3.3.1.6. Code Documentation

3.3.1.6.1. **Function**: mast_query

3.3.1.6.1.1. Parameters:

- NONE

3.3.1.6.1.2. Description:
- Pytest fixture to create a MastQuery instance for testing.

3.3.1.6.1.3. Returns:
- Instance of MastQuery class with a MagicMock connection.

3.3.1.6.2. **Function**: test_mast_auth_success

3.3.1.6.2.1. Parameters:
- mast_query (fixture)

3.3.1.6.2.2. Description:
- Test authentication to MAST database successful.

3.3.1.6.2.3. Returns:
- NONE

3.3.1.6.3. **Function**: test_connect_sqlite3_success

3.3.1.6.3.1. Parameters:
- mast_query (fixture)

3.3.1.6.3.2. Description:
- Test successful connection to SQLite database.

3.3.1.6.3.3. Returns:
- NONE

3.3.1.6.4. **Function**: test_disconnect_from_db

3.3.1.6.4.1. Parameters:
- mast_query (fixture)

3.3.1.6.4.2. Description:
- Test disconnection from SQLite database.

3.3.1.6.4.3. Returns:
- NONE

3.3.1.6.5. **Function**: test_fetch_from_sql_db_success

3.3.1.6.5.1. Parameters:
- mast_query (fixture)

3.3.1.6.5.2. Description:
- Test fetching data from SQL database successfully.

3.3.1.6.5.3. Returns:
- NONE

3.3.1.6.6. **Function**: test_convert_mjd_to_datetime

3.3.1.6.6.1. Parameters:
- mast_query (fixture)

3.3.1.6.6.2. Description:
- Test conversion of Modified Julian Date (MJD) to datetime.

3.3.1.6.6.3. Returns:
- NONE

3.3.1.6.7. Progamer Notes & DocStrings

- Ensure all test cases cover various scenarios and edge cases to verify the robustness of the MastQuery class.
- Document each test case clearly to provide insights into what functionality it is testing.

3.4. Mission Information Gatherer (MIG)

3.4.1. jwstDataFinder.py

3.4.1.1. Introduction:

- `jwstDataFinder.py` is a Python script developed as part of the CS 422 Project 2. It scrapes data related to the James Webb Space Telescope (JWST) from the official observing schedules website and writes the data to a text file.

3.4.1.2. Authorship and Credits:

- **Author:** Isabella Cortez
- **Credits:** The author acknowledges resources from YouTube and GeeksforGeeks for guidance and inspiration.

3.4.1.3. Functionality:

3.4.1.3.1. The script performs the following tasks:

- Scrapes data from the JWST observing schedules website.
- Writes the scraped data to a text file (jwst_data.txt).
- Utilizes session management for robust web scraping, handling connection errors and retries.

3.4.1.4. Dependencies:

3.4.1.4.1. The script relies on the following external libraries:

- requests
- bs4 (Beautiful Soup)
- json
- time

3.4.1.5. Usage:

3.4.1.5.1. To use `jwstDataFinder.py`, ensure that all dependencies are installed. Run the script in a Python environment.

3.4.1.5.2. *Example*:

- python jwstDataFinder.py

3.4.1.6. Code Documentation:

3.4.1.6.1. Function: create_session()

3.4.1.6.1.1. Parameters: NONE

3.4.1.6.1.2. Returns:

- session (requests.Session): A session object configured with retry settings.

3.4.1.6.1.3. Description:

- This function creates and configures a session object for making HTTP requests with retries.

3.4.1.6.1.4. Progamer Notes & DocStings:

- Pull form Willard’s project 1 webscarpper.

3.4.1.6.2. Function: scrape_jwst_data(session, u)

3.4.1.6.2.1. Parameters: u (str):
- session (requests.Session): A session object for making HTTP requests.
- u (str): URL of the JWST observing schedules website.

3.4.1.6.2.2. Returns:
- jwst_data_list (list): A list containing the scraped data.

3.4.1.6.2.3. Description:
- This function scrapes data from the JWST website by making a GET request to the provided URL. It extracts relevant data links and retrieves the text data from each link.

3.4.1.6.2.4. Progamer Notes & DocStings:
- scrape_jwst_data takes data from the jwst table and returns the list
- create an empty list titled jwst_data_list
- wait for 5 seconds before getting request to scrape
- request: get request to specific url link
- data = BeautSoup.find('a', {'class': 'link-icon-added', 'target': '_blank'})
- extract the data from url and convert it to text
- since the data on the url is .txt and a table, you can get the text via .text
- extract_data = check_url.text
- jwst_list_links.append(jwst_link)
- print(jwst_list_links)
- each string in the text becomes a list of items
- jwst_scraped_data = jwst_text.splitlines()
- append to list
- return list

3.4.1.6.3. Function: write_to_txt(jwst_data_list, output_file='jwst_data.txt')

3.4.1.6.3.1. Parameters:
- jwst_data_list (list): List of scraped data.
- output_file (str): Output file path for writing the data. Default is 'jwst_data.txt'.

3.4.1.6.3.2. Description:
- This function writes the scraped data to a text file.

3.4.1.6.3.3. Returns:
- NONE

3.4.1.6.3.4. Progamer Notes & DocStings:

- NONE

3.4.1.6.4. Function: return_jwst_data(session)

3.4.1.6.4.1. Parameters:

- session (requests.Session): A session object for making HTTP requests.

3.4.1.6.4.2. Description:

- This function orchestrates the scraping and saving of JWST data. It calls scrape_jwst_data and write_to_txt, and returns the scraped data.

3.4.1.6.4.3. Returns:

- jwst_info (list): A list containing the scraped data.

3.4.1.6.4.4. Progamer Notes & DocStings:

- this function calls the scraped_jwst_data and then calls save_to_json and everything is output
- jwst url used
- scraping jwst data
- saving jwst data to json
- save_to_json(jwst_info)
- returning the data

3.4.1.6.5. Function: get_jwst_as_py_list()

3.4.1.6.5.1. Parameters:

- NONE

3.4.1.6.5.2. Description:

- This function loads the scraped JWST data from a JSON file and returns it as a Python list. This documentation provides a comprehensive overview of the script's functionality, usage, and code structure, making it easier for developers to understand and utilize the

3.4.1.6.5.3. Returns:

- scraped_jwst_data (list): A list containing the scraped data loaded from JSON.

3.4.1.6.5.4. Progamer Notes & DocStings:

- opens the json file (jwst_data.json)
- sets the jwst data list to json.loads
- returns list as python list

3.4.2. jwstDatabase.py

3.4.2.1. Introduction:

- jwstDatabase.py is a Python script developed as part of the CS 422 Project 2. Its primary function is to convert data from a JSON file (jwst_data.json) into SQLite format, creating a SQLite database (jwstDatabaseFile.sqlite) for storage and retrieval.

3.4.2.2. Authorship and Credits:

- **Author**: Isabella Cortez

- **Credits**: The author acknowledges the easyA GitHub repository of Willard's database implementation for inspiration and guidance.

3.4.2.3. Functionality:

3.4.2.3.1. The script performs the following tasks:
- Reads data from the jwst_data.json file.
- Converts the JSON data into SQLite format.
- Creates a SQLite database (jwstDatabaseFile.sqlite) and a table (jwst_data) to store the converted data.
- Inserts the JSON data into the SQLite database.

3.4.2.4. Dependencies:

3.4.2.4.1. The script relies on the following Python standard library modules:
- sqlite3: For SQLite database operations.
- json: For JSON data processing.

3.4.2.5. Usage:
- To use jwstDatabase.py, ensure that the jwst_data.json file is present in the same directory as the script. Execute the script in a Python environment.

3.4.2.5.1. Example:
- python jwstDatabase.py

3.4.2.6. Code Documentation:

3.4.2.6.1. Function: with open('jwst_data.json') as file:

3.4.2.6.1.1. Parameters:
- NONE

3.4.2.6.1.2. Description:
- The JSON file containing JWST data to be converted into SQLite format.

3.4.2.6.1.3. Returns:
- NONE

3.4.2.6.1.4. Progamer Notes & DocStings
- NONE

3.4.2.6.2. Function: sqlite3.connect('jwstDatabaseFile.sqlite')

3.4.2.6.2.1. Parameters:
- NONE

3.4.2.6.2.2. Description:
- This statement establishes a connection to the SQLite database file (jwstDatabaseFile.sqlite).

3.4.2.6.2.3. Returns:
- connection (sqlite3.Connection): Connection object to the SQLite database.

3.4.2.6.2.4. Progamer Notes & DocStings
- NONE

3.4.2.6.3. Function: CREATE TABLE IF NOT EXISTS jwst_data (...)

3.4.2.6.3.1. Parameters:

- NONE

3.4.2.6.3.2. Description:

- SQL statement to create the jwst_data table in the SQLite database if it does not already exist. The table schema includes columns for various data fields such as visit_id, pcs_mode, visit_type, etc.

3.4.2.6.3.3. Returns:

- NONE

3.4.2.6.3.4. Progamer Notes & DocStings

- NONE

3.4.2.6.4. Function: INSERT INTO jwst_data VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

3.4.2.6.4.1. Parameters:

- data_point (dict): A dictionary containing JWST data.

3.4.2.6.4.2. Description:

- SQL statement to insert a single data point into the jwst_data table. Values are provided as a tuple.

3.4.2.6.4.3. Returns:

- NONE

3.4.2.6.4.4. Progamer Notes & DocStings:

- NONE

3.4.2.6.5. Function: conn.commit()

3.4.2.6.5.1. Parameters:

- NONE

3.4.2.6.5.2. Description:

- Commits the transaction to the SQLite database. All changes made since the last commit are saved permanently.

3.4.2.6.5.3. Returns:

- NONE

3.4.2.6.5.4. Progamer Notes & DocStings

3.4.2.6.6. Function: conn.close()

3.4.2.6.6.1. Parameters:

- NONE

3.4.2.6.6.2. Description:

- Closes the connection to the SQLite database, releasing any associated resources. This documentation provides a detailed explanation of the script's functionality, usage instructions, and code structure, making it easier for developers to understand and utilize the

3.4.2.6.6.3. Returns:

- NONE

3.4.2.6.6.4. Progamer Notes & DocStings

- NONE

3.4.3. jwstJason.py

3.4.3.1. Introduction:

- jwstJson.py is a Python script developed as part of the CS 422 Project 2. Its purpose is to convert data from a text file (jwst_data.txt) into JSON format, producing an output file named jwst_data.json.

3.4.3.2. Authorship and Credits:

- **Author:** Isabella Cortez
- **Credit:** No specific external credits mentioned.

3.4.3.3. Functionality:

3.4.3.3.1. The script performs the following tasks:

- Reads the content of the jwst_data.txt file.
- Extracts visit information from the text file based on predefined column headers.
- Converts the extracted information into a list of dictionaries, where each dictionary represents a visit.
- Writes the visit information to a JSON file (jwst_data.json) with proper formatting.

3.4.3.4. Dependencies:

3.4.3.4.1. The script relies on the following Python standard library modules:

- re: For regular expression operations.
- JSON: For JSON data processing.

3.4.3.5. Usage

- To use jwstJson.py, ensure that the jwst_data.txt file is present in the same directory as the script. Execute the script in a Python environment.

3.4.3.5.1. Example:

- python jwstJson.py

3.4.3.6. Code Documentation

3.4.3.6.1. Function: parse_txt_to_json(txt_file)

3.4.3.6.1.1. Parameters:

- txt_file (str): Path to the text file containing JWST data.

3.4.3.6.1.2. Description:

- This function parses the content of the text file and extracts visit information based on predefined column headers. It then converts the extracted information into a list of dictionaries, where each dictionary represents a visit.

3.4.3.6.1.3. Returns:

- visit_info_list (list): A list of dictionaries containing visit information.

3.4.3.6.1.4. Progamer Notes & DocStings:

- Read the content of the text file
- Define a pattern for extracting column headers
- Find the line number where the header is located
- Extract column headers
- Initialize a list to store the visit information
- Iterate over lines below the header to extract visit information
- visit_info = re.findall(r'"([^"]*)"|(\S+)', line)
- visit_info = [value[0] if value[0] else value[1] for value in visit_info]
- visit_info = [" ".join(value) if value[0] else value[1] for value in visit_info]
- visit_info = [value.strip('"') if '"' in value else value for value in re.findall(r'"([^"]*)"|(\S+)', line)]
- print(visit_info)

3.4.3.6.2. Function: json.dump(visit_info_list, json_file, indent=2)

3.4.3.6.2.1. Parameters:

- visit_info_list (list): A list of dictionaries containing visit information.
- json_file (file object): File object for writing JSON data.
- indent (int, optional): Indentation level for the JSON output. Default is 2.

3.4.3.6.2.2. Description:

- This function writes the visit information (stored in visit_info_list) to a JSON file (jwst_data.json) with proper formatting. The indent parameter specifies the indentation level for readability.

3.4.3.6.2.3. Returns:

- NONE

3.4.3.6.2.4. Progamer Notes & DocStings:

- Write the visit information to a JSON file

3.5. Web Hosting Component (University of Oregon Server Static Hosting)

3.5.1. Install_script_linux.py

3.5.1.1. Introduction:

- The script is designed to automate the installation process of required programs and Python libraries necessary for executing and launching the NebulaNet website.

3.5.1.2. Authorship and Credits:

- **Author:** Jacob Burke

3.5.1.3. Functionality:

- The install_script_linux.py script automates the installation process of essential programs and Python libraries needed to execute and launch the NebulaNet website on a Linux-based system. It uses bash commands to update system packages, install specific versions of Python and required packages, and verify successful installations.

3.5.1.4. Dependanieces:

- The script relies on the following dependencies:
    - Python 3.11.6
    - Python Package Installer (pip)
    - SQLite3
    - Various Python libraries:
        - Astropy
        - Astroquery
        - Matplotlib
        - Numpy
        - Pandas
        - Beautifulsoup4
        - Html5lib
        - Equests
        - pytest

3.5.1.5. Usage:

- To use the install_script_linux.py script, follow these steps:
    - Ensure that the script is executable by running chmod +x install_script_linux.py in the terminal.
    - Execute the script by running ./install_script_linux.py in the terminal.

3.5.1.6. Code Documentation:

3.5.1.6.1. Script Description:

- The install_script_linux.py script automates the installation process of required programs and Python libraries for the NebulaNet website. It includes bash commands to update system packages, install Python 3.11.6 and pip, install SQLite3, update pip to the latest version, and install specific Python libraries using pip. Additionally, it verifies the successful installation of Python libraries by importing them into a Python script.

3.5.1.6.2. Function: N/A

3.5.1.6.3. Parameters: N/A

3.5.1.6.4. Description: N/A

3.5.1.6.5. Returns: N/A

3.5.1.6.6. Docstrings and Programmer's Notes:

- The script lacks explicit docstrings within the code. Including docstrings would enhance the script's readability and maintainability by providing clear descriptions of each function's purpose and parameters. Additionally, the script could benefit from error handling mechanisms to handle potential installation failures or system compatibility issues.

3.5.2. Run_script_linux.py

3.5.2.1. Introduction:

- The script facilitates the execution of necessary programs and packages to create the required files and photos for the NebulaNet website.

3.5.2.2. Authorship and Credits:

- **Author:** Jacob Burke

3.5.2.3. Functionality:

- The run_script_linux.py script automates the execution of various processes required to generate the necessary files and photos for the NebulaNet website. It invokes Python scripts and commands to run web scrapers, set up SQL databases, and execute specific processes to fetch and process data for the website.

3.5.2.4. Dependencies:

- The script relies on the following dependencies:
  - Python 3
  - Pytest
  - Various Python scripts for web scraping and data processing
    - SQL database setup scripts

3.5.2.5. Usage:

- To use the run_script_linux.py script, follow these steps:
  - Ensure that the script is executable by running chmod +x run_script_linux.py in the terminal.
  - Execute the script by running ./run_script_linux.py in the terminal.

3.5.2.6. Code Documentation:

3.5.2.6.1. Script Description:

- The run_script_linux.py script automates the execution of necessary processes for generating files and photos required for the NebulaNet website. It includes bash commands to run Python scripts responsible for web scraping, setting up SQL databases, fetching and processing data, and other tasks related to data preparation.

3.5.2.6.2. Function: N/A

3.5.2.6.3. Parameters: N/A

3.5.2.6.4. Description: N/A

3.5.2.6.5. Returns: N/A

3.5.2.6.6. Docstrings and Programmer's Notes:

- Similar to the previous script, run_script_linux.py lacks explicit docstrings within the code. Including docstrings would enhance the script's readability and maintainability by providing clear descriptions of each function's purpose and parameters. Additionally, the script could benefit from error handling mechanisms to handle potential failures during the execution of subprocesses and commands.