

The Hubble Space Telescope Advanced Camera for Surveys Quicklook Application

Matthew Bourque¹, Sara Ogaz¹, Alex Viana², Meredith Durbin³, and Norman Grogan¹

[1] Space Telescope Science Institute, Baltimore, Maryland 21218. email: bourque@stsci.edu, ogaz@stsci.edu, grogin@stsci.edu

[2] Terbium Labs, Baltimore, Maryland 21201. email: alexcostaviana@gmail.com

[3] Department of Astronomy, The University of Washington, Box 351580, U.W. Seattle, Washington 98195. email: mdurbin@uw.edu

Abstract—The Hubble Space Telescope (HST) Advanced Camera for Surveys (ACS) has been acquiring thousands of astronomical images each year since its installation in 2002. The ACS Quicklook Application (`acsq1`) provides a means for users to discover and interact with these data through a database-driven web application. The system is comprised of several components: (1) A ~40 TB network file system, which stores all on-orbit ACS data files on disk, (2) a MySQL database, which stores observational metadata in a normalized relational form and allows users to build custom datasets based on observational parameters, (3) A Python/Flask-based web application, which allows users to view “Quicklook” JPEG images of any publicly-available ACS data along with their metadata, and (4) a Python code library, which provides a platform on which users can build automated instrument calibration and monitoring routines. The `acsq1` application may be extended to support the forthcoming James Webb Space Telescope (JWST) mission, which is scheduled to launch in October 2018.

1 INTRODUCTION

The Advanced Camera for Surveys (ACS) is an imaging instrument on board the Hubble Space Telescope (HST) that was installed in 2002 during Servicing Mission 3B (shuttle mission STS-109)[1]. It is comprised of three detectors: (1) the Wide Field Camera (WFC), which is designed for wide-field imaging and spectroscopy in visible to near-infrared wavelengths, (2) the High Resolution Channel, which is designed for high resolution near-ultraviolet to near-infrared wavelength images and coronography, and (3) the Solar Blind Channel (SBC), designed for far-ultraviolet imaging and spectroscopy. ACS experienced an electronics failure in 2007 that affected the WFC and HRC detectors until 2009 when astronauts successfully restored the WFC detector during Servicing Mission 4 (shuttle mission STS-125)[2]; the HRC still remains unoperational.

Despite this electronics failure, ACS has been steadily acquiring astronomical images over its 15 year on-orbit lifetime. Figure 1 shows an estimate of the number of observations over time for each of the three detectors. To date, there have been nearly 200,000 observations total. Further information about the ACS instrument including its history, configuration, performance, and scientific capability can be found in the ACS Instrument Handbook[3].

ACS data, along with the data from the other HST instruments past and present (e.g. The Wide Field Camera 3 (WFC3), The Cosmic Origins Spectrograph (COS), etc.), are publicly available and primarily stored in the Barbara A. Mikulski Archive for Space Telescopes (MAST)¹[4]. Through

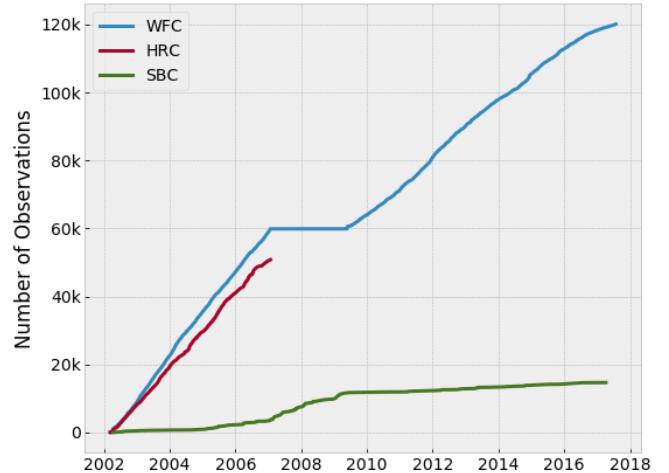


Fig. 1: The number of observations over time for each of the three detectors on ACS.

MAST, users can request and retrieve data for any publicly-available dataset via `ftp`, `sftp`, or `DVD` by mail². Like is standard for other astronomical data, ACS data are stored using the Flexible Image Transport System (FITS) data format[5]. This data format has several unique characteristics, as will be discussed in section 1.1.

The ACS Quicklook Application (hereinafter referred to as “`acsq1`”) is a Python-based application for discovering,

Manuscript received Month DD, YYYY

1. Named after the U.S. Senator from Maryland who has been a pivotal political driving force behind the manned servicing missions, the Hubble Space Telescope, and the forthcoming James Webb Space Telescope.

2. Not all HST data are publicly available; most HST data of scientific targets are considered proprietary for up to one calendar year, after which they are publicly released.

viewing, and querying all publicly-available ACS data. It consists of: (1) A filesystem that stores ACS instrument data files and “Quicklook” JPEGs in an organized Network File System (NFS) (hereinafter referred to as the “`acsq1` filesystem”), (2) A MySQL database that stores observational metadata of each observation (hereinafter referred to as the “`acsq1` database”), (3) A Python/Flask-based web application for interacting with the filesystem and database (hereinafter referred to as the “`acsq1` web application”), and (4) A Python code library that contains software for connecting to the database, ingesting new data, logging production code execution, and building/maintaining the web application (hereinafter referred to as the “`acsq1` library” or “`acsq1` package”). Each component is explained in further detail in Chapter 3.

This paper outlines the `acsq1` application as part of the Towson University Computer Science Masters Program Graduate Project. The remaining sections in this chapter discuss the motivation and use cases for this application, as well as details on the underlying data structure on which this project was built. Chapter 2 discusses how the `acsq1` application compares to related work. Chapter 3 details the implementation of each `acsq1` component. Chapter 4 outlines the results of the project and the project deliverables. Lastly, chapters 5 and 6 conclude the paper with a discussion of possible extensions and modifications to the application.

We note that the work that went into this project by the authors was accomplished on behalf of the Space Telescope Science Institute (STScI) located in Baltimore, Maryland. STScI is the home institution for instrument, data, and user support of HST, the forthcoming James Webb Space Telescope (JWST), and MAST. STScI is part of the Association of Universities for Research in Astronomy (AURA)[6].

1.1 Data Structure

The design of `acsq1`, particularly the `acsq1` database, is heavily dependent on the underlying data structure of ACS FITS files. As such, it is important for the reader to have a conceptual understanding of this data structure. The following sections are dedicated to providing an overview on the FITS data format and ACS-specific intricacies.

1.1.1 Filenames

The data from each ACS observation is contained within a FITS data file and named in a consistent fashion:

```
<rootname>_<filetype>.fits
```

Each `<rootname>` consists of nine unique alpha-numeric characters, and `<filetype>` is one of several possible three-character filetype options (discussed in section 1.1.4). For example, one ACS observation has the filename `j6mf161hq_raw.fits` (Principle Investigator Gary Bernstein, observation date 2016-09-22). Each character in the 9-character `rootname` has a specific meaning, and is discussed in section 5.2 of the Introduction to the HST Data Handbooks[7]. The `.fits` extension at the end of the filename signifies that the file is of the FITS data format.

The unique 9-character `rootname` is associated with an individual, distinct ACS observation. However, it should be

noted that only the first eight characters of the 9-character `rootname` are truly unique; the last character, which identifies the source of data transmission from the telescope³, has several possible values. However, varying values of the last character do not associate with different observations. For this reason, as will be discussed in section 3.5, we adopt an 8-character `rootname` as a primary key for the `acsq1` database tables to denote unique ACS observations.

1.1.2 FITS file structure

Each ACS FITS file consists of several “extensions”, each describing a particular aspect of the observation. Each extension consists of two parts: (1) an extension “header”, which contain key/value pairs describing image metadata (for example, `DATE-OBS = '2016-09-22'` indicates that the date the particular observation was made was 2016-09-22), and (2) the extension data, which may be a binary table or, more commonly, a multi-dimensional array of detector pixel values.

The type of extension data can also vary. The most common extension data types are (1) ‘science’ (`SCI`), in which the data values are the measure of the brightness of the astronomical scene, (2) ‘error’ (`ERR`), in which the data values are the measure of the uncertainty in the pixel values of the `SCI` extension, and (3) ‘data quality’ (`DQ`), in which the data values are 16-bit flags that describe the quality of the pixel values for the detector (for example, they may indicate that certain pixels were affected by cosmic rays during the observation). Typically, for a given file, the 1st extension is the `SCI` extension, the 2nd extension is the `ERR` extension, and the 3rd extension is the `DQ` extension. Furthermore, the 0th extension typically has no extension data and only an extension header, containing metadata that is common to all extensions. This is referred to as the “primary header”.

Tables 1 and 2 describe the different extensions of ACS FITS files for each of the three ACS detectors. Note that there are two sets of `SCI/ERR/DQ` extensions for WFC, since WFC is comprised of two separate CCD chips.

TABLE 1: ACS/WFC FITS file extensions

Extension	Purpose	Image Dimensions (pixels)	Data Type
0	Primary header	–	String
1	<code>SCI</code> , Chip 2	(4096, 2048)	Float
2	<code>ERR</code> , Chip 2	(4096, 2048)	Float
3	<code>DQ</code> , Chip 2	(4096, 2048)	Integer
4	<code>SCI</code> , Chip 1	(4096, 2048)	Float
5	<code>ERR</code> , Chip 1	(4096, 2048)	Float
6	<code>DQ</code> , Chip 1	(4096, 2048)	Integer

Over the years there have been several tools written in various programming languages to read FITS files into memory as well as automatically convert their extension data into multi-dimensional array data types and their extension headers to dictionary or string data types. For this project, the `astropy.io.fits` Python module is used

3. For example, `q` = solid-state recorder, `s` = retransmitted solid-state recorder

TABLE 2: ACS/HRC and ACS/SBC FITS file extensions

Extension	Purpose	Image Dimensions (pixels)	Data Type
0	Primary header	–	String
1	SCI	(1024, 1024)	Float
2	ERR	(1024, 1024)	Float
3	DQ	(1024, 1024)	Integer

extensively to read and interact with the data of ACS FITS files[8].

1.1.3 FITS file extension headers

As mentioned in section 1.1.2, each FITS extension contains a header, which contains key/value pairs of metadata associated with the extension data. Such metadata may describe the astronomical observation (e.g. target name, exposure time, principle investigator name, etc.), telemetry of ACS instrument or HST in general at the time of observation (e.g. temperature of the ACS instrument, orientation of the telescope pointing, position of the telescope relative to Earth, etc.) or the FITS file itself (e.g. the number of extensions, file creation date, etc.). A subsection of an example header is shown in Figure 2. Note that extension headers may contain a large number of keyword/value pairs; some extension headers contain upwards of 300 keywords, while others may contain only ~40.

```

SIMPLE = T / data conform to FITS standard
BITPIX = 16 / bits per data value
NAXIS = 0 / number of data axes
EXTEND = T / File may contain standard extensions
NEXTEND = 6 / Number of standard extensions
GROUPS = F / image is in group format
DATE = '2016-09-22' / date this file was written (yyyy-mm-dd)
FILENAME= 'j6mf16lhq_raw.fits' / name of file
FILETYPE= 'SCI' / type of data found in data file

TELESCOP= 'HST' / telescope used to acquire data
INSTRUME= 'ACS' / identifier for instrument used to acquire data
EQUINOX = 2000.0 / equinox of celestial coord. system

/ DATA DESCRIPTION KEYWORDS

ROOTNAME= 'j6mf16lhq' / rootname of the observation set
IMAGETYP= 'DARK' / type of exposure identifier
PRIMESI = 'ACS' / instrument designated as prime

/ TARGET INFORMATION

TARGNAME= 'DARK' / proposer's target name
RA_TARG = 0.0000000000E+00 / right ascension of the target (deg) (J2000)
DEC_TARG= 0.0000000000E+00 / declination of the target (deg) (J2000)

/ PROPOSAL INFORMATION

PROPOSID= 9433 / PEP proposal identifier
LINENUM = '16.055' / proposal logsheet line number
PR_INV_L= 'Bernstein' / last name of principal investigator
PR_INV_F= 'Gary' / first name of principal investigator
PR_INV_M= '' / middle name / initial of principal investigator

/ EXPOSURE INFORMATION

SUNANGLE= 93.563698 / angle between sun and V1 axis
MOONANGL= 33.222004 / angle between moon and V1 axis
SUN_ALT = 68.062172 / altitude of the sun above Earth's limb
FGSLLOCK = 'FINE' / commanded FGS lock (FINE,COARSE,GYROS,UNKNOWN)
GYROMODE= '3' / number of gyros scheduled, T=3+OBAD
REFFRAME= 'GSC1' / guide star catalog version
MTFLAG = '' / moving target flag; T if it is a moving target

DATE-OBS= '2003-01-27' / UT date of start of observation (yyyy-mm-dd)
TIME-OBS= '15:20:01' / UT time of start of observation (hh:mm:ss)
EXPSTART= 5.266663890058E+04 / exposure start time (Modified Julian Date)
EXPEND = 5.266665048715E+04 / exposure end time (Modified Julian Date)
EXPTIME = 1000.000000 / exposure duration (seconds)--calculated

```

Fig. 2: A section of an example header, taken from the 0th extension of the file *j6mf16lhq_raw.fits*.

1.1.4 FITS filetypes for ACS

As discussed in section 1.1.1, each ACS observation may result in several FITS filetypes. Each filetype has a specific scientific application and the set of available filetypes for a given observation is dependent on the characteristics of the observation, the details of which are beyond the scope of this paper. However, these details can be ascertained from the ACS Data Handbook[9]. To provide some context, below we give a brief description of each possible filetype that a given observation may contain:

- **raw** - the raw, uncalibrated data that comes directly from HST
- **flt** - nominally calibrated data
- **flc** - nominally calibrated data corrected for Charge Transfer Efficiency (CTE) deficits.
- **drz** - geometric distortion-corrected data
- **drc** - geometric distortion-corrected and CTE corrected data
- **spt** - telescope telemetry data
- **jit** - telescope pointing data
- **jif** - telescope drifting data
- **crj** - cosmic ray rejected data
- **crc** - cosmic ray rejected plus CTE corrected data
- **asn** - observation association table.

As noted earlier, a given observation may not result in the set of all possible filetypes. For example, the observation *j6mf16lhq* only results in the filetypes **raw**, **flt**, **jit**, **jif**, and **spt**.

1.2 Key Metadata

There are several metadata key/value pairs that are particularly important for the `acsq1` application, namely the web application which often presents these metadata to the user. To provide some context for the remainder of this paper, these metadata are briefly described below. Note that the `rootname` and `proposal_type` are not metadata from extension headers, but rather are metadata explicitly added to the `acsq1` database schema.

APERTURE - The portion of the WFC, HRC, or SBC detector that was used during an observation. This can either be the entire detector (e.g. WFC, referred to as a “full-frame image”), or a subsection of the detector (e.g. WFC1-1K, referred to as a “subarray”).

DATE-OBS - The date of the observation in the format `YYYY-MM-DD`, measured in Universal Time (e.g. ‘2017-08-05’).

DEC_TARG - The declination of the target (i.e. the angular distance the target north or south of the celestial equator) (e.g. 41.2842).

DETECTOR - The detector used for the observation. Can either be WFC, HRC, or SBC.

EXPFLAG - Indicates if an observation was interrupted (e.g. INTERRUPTED) or not (e.g. NORMAL).

EXPSTART - The exposure start time of the observation, in units of Modified Julian Date (e.g. 52473.84839).

EXPTIME - The exposure duration of the observation, in units of seconds (e.g. 1000.0).

FILTER1 - The selected element from the ACS filter wheel # 1 (e.g. F606W).

FILTER2 - The selected element from the ACS filter wheel # 2 (e.g. F814W).

IMAGETYP - The type of exposure for the observation (e.g. BIAS, EXT, etc.).

OBSTYPE - The type of observation, either IMAGING, SPECTROSCOPIC, CORONOGRAPHIC, or INTERNAL.

proposal_type - The type of proposal that the observation belongs to, such as Calibration (i.e. CAL) or General Observer (i.e. GO).

PROPOSID - The 4 or 5-digit proposal number that the observation belongs to (e.g. 10695).

RA_TARG - The right ascension of the target (i.e. the angular distance of the target east and west on the celestial sphere) (e.g. 49.5375).

rootname - The 8-character unique rootname of the observation (e.g. j5915401).

SUBARRAY - A boolean flag that indicates if the observation is a full-frame APERTURE (i.e. 0) or a subarray APERTURE (i.e. 1).

TARGNAME - The name of the target (e.g. M87, NGC-4536, ANDROMEDA-I, etc.).

TIME-OBS - The time of the start of the observation in the format HH:MM:SS, measured in Universal Time (e.g. 14:21:56).

1.3 Motivation

The motivation for the acsql application is driven by several limitations of the FITS file structure as well as limitations in the current capabilities of MAST from specific user perspectives. Some of these limitations are described below, along with how the acsql application aims to address them. We also discuss the intended users of the application and their expected use cases.

1.3.1 Data retrieval latency

Currently, users who wish to retrieve data from MAST must submit a retrieval request via the MAST online interface or an internal XML request. Once the retrieval request is processed (usually automatically unless it is a request of a large number of datasets), the data are either transferred to the user directly via Secure File Transfer Protocol (sftp),

transferred to a “staging area” in which the user can log into and copy the data via File Transfer Protocol (ftp) at their leisure, or sent by mail via DVD, depending on which option the user selects. In any case, the duration between a download request and the time at which the user has fully retrieved the data may be significant. In the fastest scenario of an automatically accepted request and the sftp option, a typical request can take minutes to hours to be completed. Furthermore, there are limited options available for programmatically obtaining new data; users who wish to retrieve the latest available data must (1) discover which datasets are new via a query to the MAST database, (2) construct a download request, (3) submit the download request, and (4) await or retrieve the data. This workflow makes it non-trivial to discover and perform analysis on new data within the same software program.

The acsql application attempts to circumnavigate this retrieval process by making the full data products instantly available via read-only access of the acsql filesystem, as well as a view of the image data (and corresponding metadata) instantly available through the acsql web application. By having all of the available data centrally located, users need not to go through this request process; data can be directly read from the storage areas on disk.

1.3.2 File I/O

A significant limitation of the FITS file format is the amount of time required for file input/output. With users often finding themselves analyzing hundreds to thousands of images, this file I/O can be quite a burden on the total processing time of such analysis. The acsql application attempts to mitigate this processing time by storing the valuable image metadata that users often seek in a relational database.

Queries to the acsql database prove to be much faster than opening and closing individual FITS files, and the difference in data retrieval times increases linearly with increasing amount of files. Consider an example situation in which a user wishes to retrieve the date of observation (i.e. the DATE-OBS header keyword in the 0th extension of a raw FITS file) for one hundred files. Figure 3 shows that the total time required to gather this information is \sim 10-fold less when querying the acsql database than when retrieving the data directly from the FITS files, with this gap increasing linearly with increasing number of files at a rate of \sim 2.5 seconds per 100 files.

In order to build the acsql database, each ACS FITS file must ultimately be opened in order to retrieve the header information. However, this operation is only performed once and needs not to be performed by the user; once the database is built, users can retrieve data by fast database queries.

1.3.3 Data redundancy

As will be discussed in section 1.3.5, the primary user base for the acsql application is a group of ACS instrument analysts who use ACS data on a daily basis. As such, each user may require access to a varying amount of ACS data over time; some data may be useful for their work on a particular day, but not on another, while some data are consistently used over time. Furthermore, instrument analysts may share the need for specific data with one or

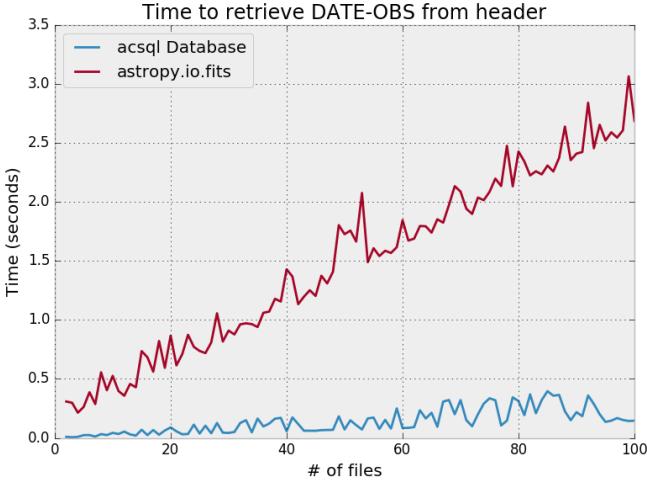


Fig. 3: The amount of time it takes to retrieve the `DATE-OBS` header keyword from a varying number of ACS data files via reading the FITS file using the `astropy.io.fits` module (red) and querying the `acsq1` database (blue).

more of their colleagues. Consider a situation in which two instrument analyst require data from the proposal 11655. Traditionally, for the two instrument analysts to obtain the data they need, they would have to submit a download request to MAST for data from 11655 and store the data in personal directories. Unless this effort is explicitly coordinated amongst the two individuals, they will each likely download and store a separate copy of this data, when in reality, only one copy is needed. This workflow leads to possible data redundancy. With the ACS instrument team employing roughly a dozen instrument analysts, the scale of this data redundancy can grow quite large.

Recently, MAST has mitigated this issue through what is known as the MAST public cache, which is a centrally-located, organized network file system that stores all publicly-available HST data and is internal to STScI. The `acsq1` filesystem is built on top of the MAST public cache to provide accessibility to all ACS data in one central location. Furthermore, the `acsq1` application provides the `acsq1` database (for observational metadata needs) and the `acsq1` web application (for quick data viewing). With the combination of these three components, instrument analysts have instant access to all ACS data at any time in one centralized location; there is no need for coordination amongst other instrument analysts or storage of separate copies.

1.3.4 Data discovery

As will be discussed in section 1.3.5, ACS instrument analysts often wish to know which datasets (i.e. rootnames) exist for various observational parameters. For example, one may need to analyze data that were observed between a specific time period, or observed with a specific pointing on the sky. Conversely, an instrument analyst may have knowledge of specific rootnames, but wish to learn of the observational parameters associated with them.

MAST has tools which allow users to discover rootnames and/or parameters for observations. The MAST portal is a

public-facing online data discovery tool for various data collections such as HST data, high level science products, and catalogs of astronomical source positions[10]. The MAST portal offers features that may be of particular interest to research astronomers at academic institutions, for example; an image of the location of the scientific target on the sky ('AstroView'), an interactive GUI for displaying results, and an option to export results to various file types. An example query using this tool is shown in Figure 4.

Though the MAST portal has many useful features, there are some limitations to the tool from the perspective of an instrument analyst at STScI. One such limitation is that the available observational metadata do not include all of the data available in the FITS headers; it is limited to a small subset of header keywords, mainly from the RAW and FLT filetypes, as well as other MAST-generated metadata not found in the FITS headers (e.g. 'Target Classification'). Another limitation is that the queries are keyed off of an 'Observation ID' as opposed to a traditional 9-character rootname. The Observation ID may map to one or more individual observations (and thus one or more individual rootnames), making it difficult for instrument analysts to discover which rootnames (and thus data files) that are associated with the observational metadata they seek.

The `acsq1` application provides an intuitive, one-to-one correspondance between the FITS header values and the `acsq1` database schema. The column names in the tables of the `acsq1` database exactly match the names found in the FITS headers, and each extension of each ACS filetype is supported. Furthermore, the primary key of each table in the database is the 9-character rootname that instrument analysts use in their day-to-day work, as opposed to an Observation ID. With this configuration, it is straightforward to discover data via a database query using header keywords.

1.3.5 Use Cases

The intended primary users of the `acsq1` application are ACS instrument analysts that work for the ACS instrument team at STScI. ACS instrument analysts use ACS data frequently in their day-to-day work, using it to help calibrate, monitor, and characterize various aspects of the ACS instrument. On any given day, an instrument analyst may need to analyze data from a number of observing modes, time periods, or filetypes. In this regard, the `acsq1` application aims to limit the amount of time and effort required by the analysts to discover and access the data that they need to perform such work. It should be noted, though, that the `acsq1` application is not necessarily limited to this user base; since the application contains only public data, there is potential for users external to STScI, such as scientific researchers at other academic institutions.

With the nature of ACS instrument work in mind, the `acsq1` application was designed to support four main use cases, each briefly described below.

Use case 1: Visually inspect an image from the ACS archive: The application shall allow users to view a 'Quicklook' JPEG of any publicly-available image in the ACS archive, along with useful corresponding observational metadata.

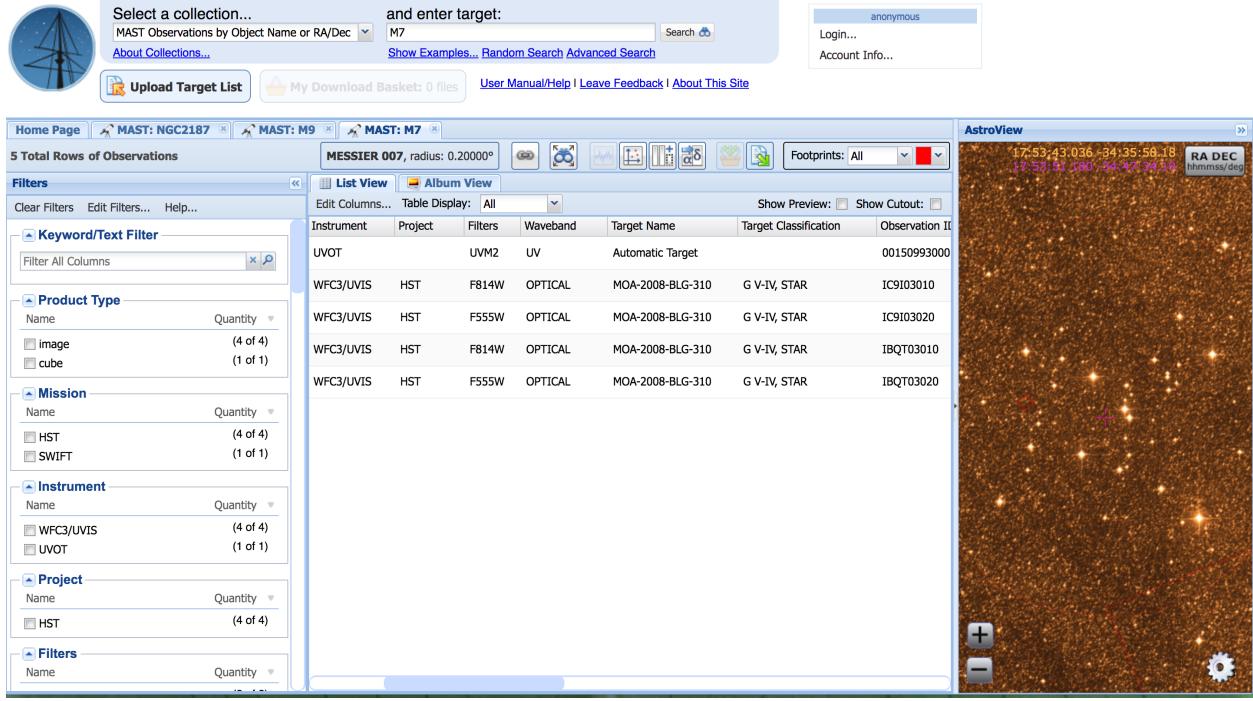


Fig. 4: An example of a query to the MAST data discovery portal.

Use case 2: Determine which datasets exist in the ACS archive for a given set of observational parameters:

The application shall allow users to determine which 9-character rootnames exist for a defined set of observational parameters via ACS header keywords.

Use case 3: Determine the observational parameters for a given dataset: The application shall allow users to determine observational parameters (i.e. FITS header keyword values) for a given 9-character rootname.

Use case 4: Programmatically analyze images across custom datasets: The application shall allow users to directly read ACS image data into memory and perform analysis for data obtained in Use Case (2).

2 RELATED WORK

The main inspiration for the construction of the `acsq1` application was derived from the success of a similar application for the Wide Field Camera 3 (WFC3) instrument (also on board HST), known as the WFC3/Quicklook (`wfc3ql`) application. The `wfc3ql` application shares many of the core components that `acsq1` does, including a filesystem that stores all WFC3 data on disk, a relational database that stores WFC3 FITS header information, and a web application for user interactivity with the system[11][12]. The `wfc3ql` application has been developed by the WFC3 team at STScI since the instrument was installed on HST in 2009 during Servicing Mission 4[2]. The genesis of the application was a single script that performed a download request to MAST for new data, created Quicklook JPEGs from the resulting images, and saved them to a centrally-located directory. Over time, the project has evolved into the

filesystem-database-web stack that more closely resembles that of the `acsq1` application.

Despite the similarities, it should be noted that the implementation of the `wfc3ql` infrastructure differs significantly from that of `acsq1` (implementation details of the `acsq1` components are outlined in Chapter 3). For example, unlike the `acsq1` filesystem, which contains only publicly-available data, the `wfc3ql` filesystem is essentially a duplicate of the MAST archive for WFC3, contains all data, including proprietary data which can only be used internally by the WFC3 team at STScI. Since the `wfc3ql` filesystem contains proprietary data, much support is needed in the software infrastructure to restrict data to certain file directories, user groups, and file permissions. Additionally, the `wfc3ql` database only supports a small subset of FITS filetypes and extensions, while `acsq1` supports all ACS filetypes and extensions.

With `wfc3ql` being continuously developed over the past eight years by a team of several developers, many features have been implemented for the application that are not currently implemented on `acsq1`. For example, the `wfc3ql` application supports the daily visual inspection of new WFC3 data, some data of which are proprietary. Such a feature is not possible for `acsq1`, as the `acsq1` filesystem is built using the MAST cache of only publicly-available data (and thus the `acsq1` application can be considered open-source and non-proprietary). Secondly, `wfc3ql` currently contains several instrument calibration and monitoring routines built upon the `wfc3ql` automation platform and code library (known as `pyql`), while `acsq1` has yet to implement any such routines. A third example is that the `wfc3ql` web application supports the tracking of WFC3 instrument image ‘anomalies’ (i.e. anomalous features in images that are tracked by instrument team members[13]). These fea-

tures, amongst others, clearly serve as possible extensions to the `acsq1` system; such extensions are discussed further in Chapter 5.

3 METHODOLOGY AND IMPLEMENTATION

In this chapter, we discuss the methods by which we implemented the various components of the `acsq1` application. Additionally, we discuss the programming standards and workflows (sections 3.1 and 3.2) that were employed to foster code qualities such as readability, maintainability, extensibility, etc.; we believe that this aspect of the project is equally important to the application and its future maintenance as its individual components.

3.1 Version control

All software associated with this project (including this paper itself) is version-controlled using the `git` Version Control System (VCS)[14]. The `git` repository for the project is named “`acsq1`” and is hosted on `GitHub`, a repository hosting service[15]. The repository is publicly available at <http://github.com/spacetelescope/acsq1>.

Several feature branches of the software were created throughout the building of the `acsq1` application such that the `master` branch (which is considered the ‘production’ branch) always contained operational software while the feature branches may have contained unfinished implementations. Such feature branches include `create-database` (for implementation of the `acsq1` database schema), `add-logging` (for implementation of system logging), `build-ingest` (for implementation of the data ingestion software), and `web-application` (for implementation of the `acsq1` web application). For each merge of a feature branch, a tag and release was created for the `master` branch, which allowed a specific version of the `master` branch to be saved in the repository. These releases are available at <https://github.com/spacetelescope/acsq1/releases>.

Additionally, `GitHub` allowed for issue tracking for bugs, questions, and potential enhancements to the code repository. Current open issues of the repository can be found at <https://github.com/spacetelescope/acsq1/issues>.

3.2 Programming and Documentation Standards

All code contained within this project was written to adhere to specific standards and conventions, namely (1) the PEP8 Style Guide for Python code[16], (2) The PEP257 Python guide for module and function docstrings [17], and (3) the `numpydocs` documentation standard [18]. More details on each of these standards and conventions are given below.

The PEP8 Style Guide for Python code (abbreviated for ‘Python Enhancement Proposal #8’) documents Python coding conventions including variable naming, spacing, line length, module layout, function layout, comments, and design patterns. Only in specific cases did we diverge from these conventions, such as exceeding the recommended 80 characters line length to allow for greater readability. By following these conventions, the style of the `acsq1` software is consistent across each module and reflects the style of industry-grade Python code.

The PEP257 guide for docstring conventions describes standards used for function and module docstrings (i.e. the API documentation found in block comments at the beginning of modules or immediately after function declarations). Like PEP8, following these conventions ensure consistency amongst the `acsq1` code documentation. Furthermore, the `numpydocs` documentation convention provides some details in addition to the PEP257 conventions and is used in many Python packages including the `numpy` (numerical Python) and `scipy` (scientific Python) packages[19]. Figure 5 shows an example of these conventions, taken from the `acsq1.ingest.ingest.get_proposal_type` function.

Another benefit of using PEP257 and `numpydoc` docstring conventions is that API documentation creation tools such as `sphinx`[20] or `epydoc`[21] can automatically convert the docstrings into other output formats such as `HTML` and `PDF`. For this project, we use `sphinx` to convert API documentation to `HTML` and host the webpages online using `readthedocs`, an open-source, community supported tool for hosting and browsing documentation[22]. The output documentation as seen on `readthedocs` for the example function in Figure 5 is provided in Figure 6. The documentation for `acsq1` is available at <http://acsq1.readthedocs.io/>.

3.3 Filesystem: Archive of ACS data

The `acsq1` filesystem is a Network File System (NFS) that stores all publically available on-orbit ACS data on disk in an organized set of directories and subdirectories hosted at STScI. Figure 7 shows an example of this directory structure. The parent directory is the first four characters of the 9-character `rootname`, which has a one-to-one correspondence with an individual PROPOSID. The subdirectories of the parent directories are named after the full 9-character `rootname` such that each parent directory contains a `rootname` subdirectory for each `rootname` that were observed for the particular PROPOSID. Furthermore, each `rootname` subdirectory contains every available file-type (as described in Section 1.1.4) for the particular observation.

```
def get_proposal_type(proposid):
    """Return the ``proposal_type`` for the given ``proposid``.

    The ``proposal_type`` is the type of proposal (e.g. ``CAL``, ``GO``,
    etc.). The ``proposal_type`` is scraped from the MAST
    proposal status webpage for the given ``proposid``. If the
    ``proposal_type`` cannot be determined, a ``None`` value is returned.

    Parameters
    -----
    proposid : str
        The proposal ID (e.g. ``12345``).

    Returns
    -----
    proposal_type : int or None
        The proposal type (e.g. ``CAL``).
    """

```

Fig. 5: An example of the PEP257 and `numpydoc` docstring conventions, using the `get_proposal_type` function from `acsq1.ingest.ingest`.

```
ingest.ingest.get_proposal_type(proposid)

Return the proposal_type for the given proposid.

The proposal_type is the type of proposal (e.g. CAL, GO, etc.). The
proposal_type is scraped from the MAST proposal status webpage for the
given proposid. If the proposal_type cannot be determined, a None value is
returned.

Parameters: proposid : str
    The proposal ID (e.g. 12345).

Returns: proposal_type : int or None
    The proposal type (e.g. CAL).
```

Fig. 6: The readthedocs documentation for the `acsq1` example function seen in Figure 5.

```
filesystem/
jcp0/
    jcp001kwq/
        jcp001kwq_flc.fits
        jcp001kwq_flt.fits
        jcp001kwq_raw.fits
        jcp001kwq_spt.fits
        jcp001kwq_jif.fits
        jcp001kwq_jit.fits
    jcp001010/
        jcp001010_asn.fits
        jcp001010_drc.fits
        jcp001010_drz.fits
        jcp001010_jif.fits
        jcp001010_jit.fits
    jcp014tyq/
    ...
    jcp001kt1/
    ...
    ...
jcp3/
...
jcp7/
...
```

Fig. 7: A representation of the directory structure within the `acsq1` filesystem, using a few observations as an example.

Figure 8 shows how the total size of the filesystem has evolved over the lifetime of the ACS mission; currently, the filesystem occupies ~ 40 TB of storage space. Note that the file sizes across the detectors and across the various filetypes may vary depending on the nature of the particular observation (for example, full-frame observations result in larger file sizes than subarray observations, calibrated filetypes have larger file sizes than un-calibrated filetypes, etc.).

Currently, the `acsq1` filesystem utilizes the MAST public cache (described in section 1.3.3) to store all of the ACS data. We purposely omit details of the location and exact structure of the MAST cache as to not expose sensitive data that is internal to STScI. We note, however, that it is possible for users external to STScI to reconstruct the `acsq1` filesystem (or a chosen subset of the filesystem) by requesting the publicly-available data from MAST and organizing the files in a similar manner to that shown in Figure 7.

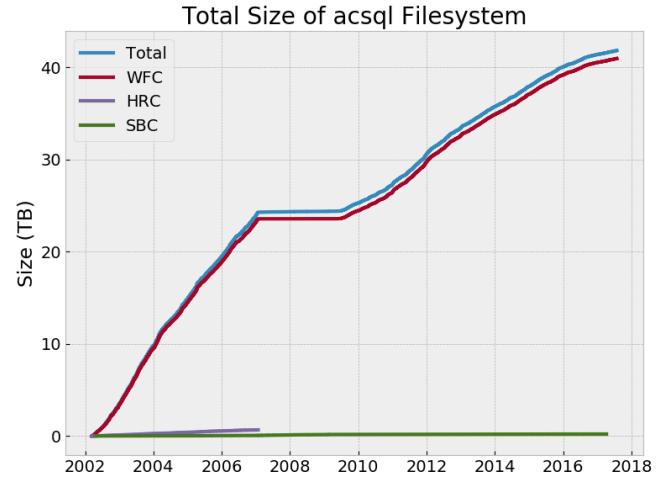


Fig. 8: The size of the `acsq1` filesystem as a function of observation date.

3.4 Filesystem: Archive of JPEGs and Thumbnails

In addition to the ACS data products described in section 3.3, the `acsq1` filesystem also stores ‘Quicklook’ JPEG and thumbnail images of each RAW, FLT, and FLC filetype (when applicable) in an organized directory structure. These images are utilized by the `acsq1` web application to allow users to quickly and easily view ACS data without having to physically open the corresponding FITS files.

The JPEG images are generated by (1) taking the two-dimensional data from the SCI extension(s), (2) sigma-clipping the top and bottom 1% of the values (as to avoid large outlier values and enhance the scaling of the image), and (3) saving the data as a JPEG format. The thumbnail images are created by simply resizing the corresponding JPEG into a 128x128 pixel image and saving the resulting image to a `.thumb` extension; the purpose of these thumbnail images are to be able to view many of them on a single webpage in the `acsq1` web application. An example of a JPEG image and its corresponding thumbnail is shown in Figure 9.

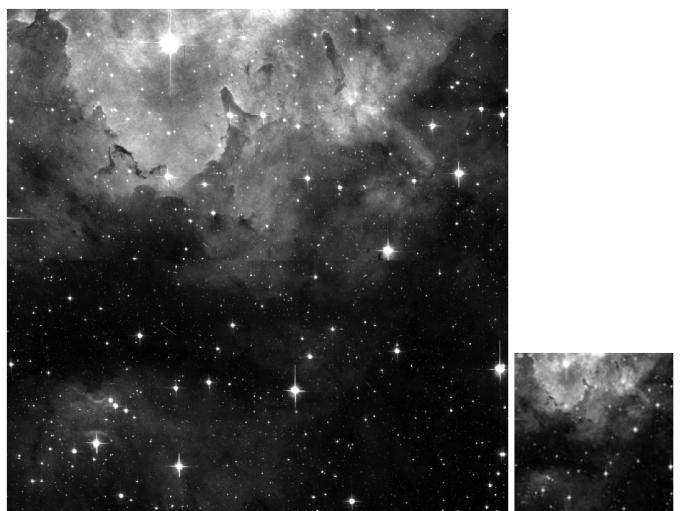


Fig. 9: An example of a JPEG image (left) and its corresponding thumbnail image using example dataset `jcs718koq`.

```

jpeg/
12780/
    jbw901jtq_flc.jpg
    jbw901jtq_flt.jpg
    jbw901jtq_raw.jpg
    jbw901jxq_flc.jpg
    jbw901jxq_flt.jpg
    jbw901jxq_raw.jpg
...
12781/
    jbx101f5q_flc.jpg
    jbx101f5q_flt.jpg
    jbx101f5q_raw.jpg
    jbx101f7q_flc.jpg
    jbx101f7q_flt.jpg
    jbx101f7q_raw.jpg
...
12783/
...
12787/
...
...

thumbnails/
12780/
    jbw901jtq_flt.thumb
    jbw901jxq_flt.thumb
...
12781/
    jbx101f5q_flt.thumb
    jbx101f7q_flt.thumb
...
12783/
...
12787/
...
...

```

Fig. 10: A representation of the directory structure for the JPEG (top) and thumbnail (bottom) portion of the acsql filesystem, using a few observations as an example.

Unlike the ACS data products portion of the filesystem (section 3.3), the JPEG and thumbnail portions of the filesystem are organized based on the four-or-five digit PROPOSID of the corresponding observation instead of the first four characters of the rootname. This design was chosen as a means to simplify the design of the web application; users often wish to view data based on the 4/5-digit PROPOSID and less often on the details of the rootname. An example of this structure is shown in Figure 10. Note that the thumbnail/ filesystem only contains thumbnails created from FLT filetypes, since the thumbnails are only intended for navigation and quick-viewing.

3.5 Database: Relational Schema

Another major component of the acsql application is a relational database that stores all FITS header key/value pairs for each ACS filetype and each FITS file extension for all on-orbit ACS observations. Such a database allows users to perform relational queries for any observational metadata based on the header keywords.

To accomplish this, we implemented the relational schema shown in Figure 11. The acsql database contains 111 tables in total: one master table, which contains basic information about each rootname that is important for maintaining the database, one datasets table which indicate which filetypes are available for a particular rootname,

and 109 ‘header’ tables which stores the header key/value pairs, one for each detector/filetype/extension combination (e.g. wfc_raw_0). Each of these tables are described in detail below.

The master table contains information that is particularly useful for maintaining and using the acsql database. Its primary key is the first 8 characters of the 9-character rootname for the particular observation (recall from section 1.1.1 that only the first 8 characters of a rootname are actually unique). The path column contains the location of the observation in the acsql filesystem. The first_ingest_date and last_ingest_date contains the date in which the observation was first inserted into the database and the date in which the observation was most recently updated in the database, respectively. The last_ingest_date allows the database maintainer to determine when data in the database may become outdated and require re-ingestion.

The datasets table lists which filetypes are available for each observation. If a particular filetype is available for the given rootname, the value for the appropriate column in the table is the full <rootname>_<filetype>.fits filename (for example, the raw column contains the value jcs718koq_raw.fits for rootname jcs718ko). If a particular filetype is not available, the value of the column is NULL. This table allows a user to determine which header tables are queryable for a given rootname. The rootname in the datasets table acts as both a primary key for the table as well as a foreign key that maps to the rootname of the master table.

The remaining 109 tables were designed to be in direct correspondence with the header metadata key/value pairs found in the ACS FITS files; each column is named in the same manner as the header keys, with the value of that column reflecting the header value. There is one table for each detector, filetype, and extension combination; collectively, these are referred to as the ‘header’ tables. Like the datasets table, the rootname column serves as a primary key for the header tables as well as a foreign key that maps to the rootname of the master table.

3.6 Database: MySQL + SQLAlchemy

The acsql database is stored on a MySQL server (Version 5.6)[23] that is hosted at STScI. The database schema was implemented using SQLAlchemy, which is an open-source SQL toolkit and Object Relational Mapper (ORM) for Python[24]. As an ORM, SQLAlchemy enables Python classes to be easily translated to SQL-based database tables, and vice versa. Additionally, SQLAlchemy provides methods for connecting to a SQL-based database and performing typical SQL tasks such as inserts, updates, and queries.

There are several key functions and classes that were used to construct the acsql database (all of which can be found in the acsql.database.database_interface.py module, further discussed in section 3.10). One such function is load_connection, shown in Figure 12. This function creates three SQLAlchemy objects that are used to establish a connection with the acsql database: engine, base, and session, each described below.

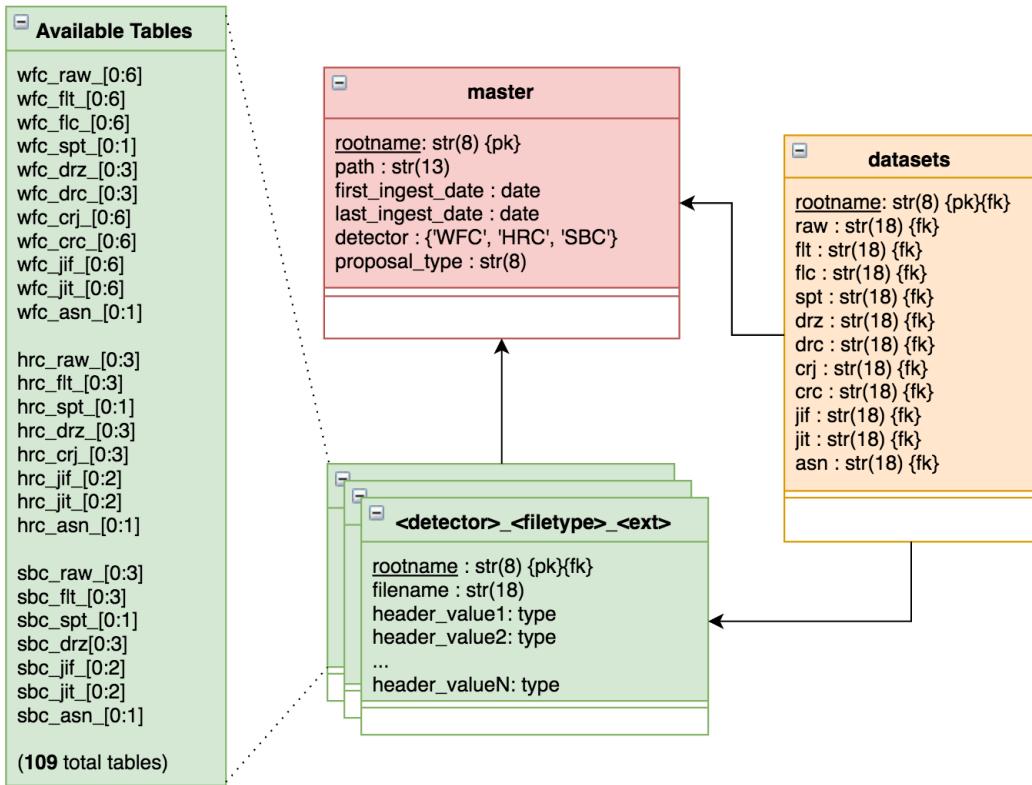


Fig. 11: The relational database schema for the `acsq1` database.

```
def load_connection(connection_string):
    """Return ``session``, ``base``, and ``engine`` objects for
    connecting to the ``acsq1`` database.

    Create an ``engine`` using a given ``connection_string``. Create a
    ``base`` class and ``session`` class from the ``engine``. Create an
    instance of the ``session`` class. Return the ``session``,
    ``base`` and ``engine`` instances.

    Parameters
    -----
    connection_string : str
        The connection string to connect to the ``acsq1`` database. The
        connection string should take the form:
        ``dialect+driver://username:password@host:port/database``

    Returns
    -----
    session : session object
        Provides a holding zone for all objects loaded or associated
        with the database.
    base : base object
        Provides a base class for declarative class definitions.
    engine : engine object
        Provides a source of database connectivity and behavior.
    """

    engine = create_engine(connection_string, echo=False, pool_timeout=10000)
    base = declarative_base(engine)
    Session = sessionmaker(bind=engine)
    session = Session()

    return session, base, engine
```

Fig. 12: The `load_connection` function, used to build a connection to the `acsq1` database.

The `engine` object contains the Python Database API Specification (also known as DBAPI), which provides a low-level API for Python-specific, commonly-used database tasks[25]. Created from the `sqlalchemy.create_engine` method, this requires a user-supplied `connection_string`. The `connection_string` contains information about the database type (e.g. MySQL, the specific database dialect being used, and the user credentials (e.g. username, password, port number, and host server name). In the case of the `acsq1` database, this connection string takes the form of '`mysql+pymysql://username:password@host:port/acsq1`'. The `connection_string` is imported from a user supplied config file within the `acsq1` library (as will be discussed in section 3.10).

The `base` object serves as a base class for declarative class definitions (i.e. the classes that are used to define the database tables). It is created from the `sqlalchemy.ext.declarative.declarative_base` method. Perhaps most importantly, the `base` object contains methods for creating and dropping the tables defined in the class definitions (e.g. `base.metadata.create_all()` and `base.metadata.drop_all()`, respectively).

The `session` object provides a primary usage interface for database operations, and is created via the `sqlalchemy.sessionmaker` method, which takes as a parameter the `engine` object. The methods of the `session` object are primarily used to query the database (i.e. `session.query()`) as well as committing inserts or updates (i.e. `session.commit()`).

The `master` and `datasets` tables were implemented

```

class Master(base):
    """ORM for the master table."""
    def __init__(self, data_dict):
        self._dict_.update(data_dict)

    __tablename__ = 'master'
    rootname = Column(String(8), primary_key=True, index=True, nullable=False)
    path = Column(String(15), unique=True, nullable=False)
    first_ingest_date = Column(Date, nullable=False)
    last_ingest_date = Column(Date, nullable=False)
    detector = Column(Enum('WFC', 'HRC', 'SBC'), nullable=False)
    proposal_type = Column(Enum('CAL/ACS', 'CAL/OTA', 'CAL/STIS', 'CAL/WFC3',
                                'ENG/ACS', 'GO', 'GO/DD', 'GO/PAR', 'GTO/ACS',
                                'GTO/COS', 'NASA', 'SM3/ACS', 'SM3/ERO',
                                'SM4/ACS', 'SM4/COS', 'SM4/ERO', 'SNAP'),
                           nullable=True)

```

Fig. 13: The class definition for constructing the master table via SQLAlchemy.

```

class Datasets(base):
    """ORM for the datasets table."""
    def __init__(self, data_dict):
        self._dict_.update(data_dict)

    __tablename__ = 'datasets'
    rootname = Column(String(8), ForeignKey('master.rootname'),
                      primary_key=True, index=True, nullable=False)
    raw = Column(String(18), nullable=True)
    flt = Column(String(18), nullable=True)
    flc = Column(String(18), nullable=True)
    spt = Column(String(18), nullable=True)
    drz = Column(String(18), nullable=True)
    drc = Column(String(18), nullable=True)
    crj = Column(String(18), nullable=True)
    crc = Column(String(18), nullable=True)
    jif = Column(String(18), nullable=True)
    jit = Column(String(18), nullable=True)
    asn = Column(String(18), nullable=True)

```

Fig. 14: The class definition for constructing the datasets table via SQLAlchemy.

via explicit class definitions in `database_interface`, and are shown in Figures 13 and 14, respectively. Each table column is defined using the `sqlalchemy.Column` object, which is a class that can be initialized with the column datatype (e.g. `String`, `Float`, `Integer`, etc.) as well as parameters that set SQL-like constraints on the column values. These include, but are not limited to, primary keys (e.g. the `primary_key=True` parameter in the `master.rootname` column), foreign key constraints (e.g. the `ForeignKey` constraint in the `datasets.rootname` column), uniqueness constraints (e.g. the `unique=True` parameters in the `master.path` column), and NULL constraints (e.g. the `nullable=False` parameter in the `master.first_ingest_date` column). SQLAlchemy determines the name of the table via the `__tablename__` attribute, and determines the name of the columns by the name of the variable used to initialize each `Column` object.

Since there are 109 header tables, some of which have hundreds of columns, it is not practical to construct a class definition for each table in a similar man-

```

def orm_factory(class_name):
    """Create a SQLAlchemy ORM Classes with the given ``class_name``.

    Parameters
    -----
    class_name : str
        The name of the class to be created

    Returns
    -----
    class : obj
        The SQLAlchemy ORM
    """

    data_dict = {}
    data_dict['__tablename__'] = class_name.lower()
    data_dict['rootname'] = Column(String(8), ForeignKey('master.rootname'),
                                  primary_key=True, index=True,
                                  nullable=False)
    data_dict['filename'] = Column(String(18), nullable=False, unique=True)
    data_dict = define_columns(data_dict, class_name)
    data_dict['__table_args__'] = {'mysql_row_format': 'DYNAMIC'}

    return type(class_name.upper(), (base,), data_dict)

```

Fig. 15: The `orm_factory` function, used to define class definitions for header tables.

ner to that of the `master` and `datasets` table. Instead, these class definitions were implemented via the `database_interface.orm_factory` function, which is a factory function that creates and returns a class definition for each header table, based on the given `class_name` that reflects the `detector` / `filetype` / `extension` combination (e.g. `wfc_raw_0`). The `orm_factory` function is shown in Figure 15. Similar to the `Master` and `Datasets` classes, some of the columns in the `orm_factory` function are explicitly defined via the SQLAlchemy `Column` class. However, the columns that correspond to header key/value pairs are defined in a separate function named `define_columns`, shown in Figure 16.

The purpose of the `define_columns` function is to define SQLAlchemy `Column` objects for each header keyword in the headers of the particular `detector` / `filetype` / `extension` combination (provided in the given `class_name` parameter). This is accomplished by reading in a text file (named `<class_name>.txt` that contains the header keywords and their datatype (one per line) for the given `class_name`. A portion of an example text file is shown in Figure 17.

Furthermore, the 109 text files used to define the header table columns are also generated in an automated fashion via the `acsq1.database.make_tabledefs.py` module. This module uses a small set of example ACS FITS files to scrape their header contents, determine all of the header keywords and their datatypes, and write the results to a text file. Similarly, the `acsq1.database.update_tabledefs.py` module adds any new header keywords to the text files by comparing the header contents of a given FITS file to the existing column definition text files⁴.

4. New header keywords are occasionally introduced to ACS data proceeding updates to its calibration software.

```

def define_columns(data_dict, class_name):
    """Dynamically define the class attributes for the ORM

Parameters
-----
data_dict : dict
    A dictionary containing the ORM definitions
class_name : str
    The name of the class/ORM.

Returns
-----
data_dict : dict
    A dictionary containing the ORM definitions, now with header
    definitions added.

"""

special_keywords = ['RULEFILE', 'FWERROR', 'FW2ERROR', 'PROPTTL1',
                    'TARDESCR', 'QUALCOM2']

with open(os.path.join(os.path.split(__file__)[0], 'table_definitions',
                      class_name.lower() + '.txt'), 'r') as f:
    data = f.readlines()
keywords = [item.strip().split(',') for item in data]
for keyword in keywords:
    if keyword[0] in special_keywords:
        data_dict[keyword[0].lower()] = get_special_column(keyword[0])
    elif keyword[1] == 'Integer':
        data_dict[keyword[0].lower()] = Column(Integer())
    elif keyword[1] == 'String':
        data_dict[keyword[0].lower()] = Column(String(50))
    elif keyword[1] == 'Float':
        data_dict[keyword[0].lower()] = Column(Float(precision=32))
    elif keyword[1] == 'Decimal':
        data_dict[keyword[0].lower()] = Column(Float(precision='13,8'))
    elif keyword[1] == 'Date':
        data_dict[keyword[0].lower()] = Column(Date())
    elif keyword[1] == 'Time':
        data_dict[keyword[0].lower()] = Column(Time())
    elif keyword[1] == 'DateTime':
        data_dict[keyword[0].lower()] = Column(DateTime())
    elif keyword[1] == 'Bool':
        data_dict[keyword[0].lower()] = Column(Boolean())
    else:
        raise ValueError('unrecognized header keyword type: {}:{}'.
                         format(keyword[0], keyword[1]))

    if 'aperture' in data_dict:
        data_dict['aperture'] = Column(String(50), index=True)

return data_dict

```

Fig. 16: The `define_columns` function, used to define columns used in the header tables.

With the implementation of the `orm_factory` and `define_columns` function, it is trivial to create class definitions for each of the 109 header tables. An example of this is shown in Figure 18, where several of the WFC header tables are defined.

With the `master`, `datasets`, and each of the 109 header tables defined in the `database_interface` module, creating the database tables on the MySQL server is accomplished by executing the `base.metadata.create_all()` method.

3.7 Data ingestion software: Algorithm

Another critical component of the `acsq1` application is the modules that are used to ingest data

```

DETECTOR, String
NEXTEND, Integer
EXTEND, Bool
SIMPLE, Bool
NAXIS, Integer
LINENUM, String
GROUPS, Bool
DATE, String
EQUINOX, Float
INSTRUIME, String
PROPOSID, Integer
ASN_ID, String
ASN_PROD, Bool
ASN_STAT, String
DEC_TARG, Float
FILETYPE, String
ASN_TAB, String
PRIMESI, String
RA_TARG, Float
TARGNAME, String
TELESCOP, String
PR_INV_F, String
BITPIX, Integer
PR_INV_M, String
PR_INV_L, String

```

Fig. 17: The contents of an example text file used to define the columns of a header table in the `define_columns` function. The example table used here is the `wfc_asn_0` table.

```

WFC_raw_0 = orm_factory('WFC_raw_0')
WFC_raw_1 = orm_factory('WFC_raw_1')
WFC_raw_2 = orm_factory('WFC_raw_2')
WFC_raw_3 = orm_factory('WFC_raw_3')
WFC_raw_4 = orm_factory('WFC_raw_4')
WFC_raw_5 = orm_factory('WFC_raw_5')
WFC_raw_6 = orm_factory('WFC_raw_6')

WFC_flt_0 = orm_factory('WFC_flt_0')
WFC_flt_1 = orm_factory('WFC_flt_1')
WFC_flt_2 = orm_factory('WFC_flt_2')
WFC_flt_3 = orm_factory('WFC_flt_3')
WFC_flt_4 = orm_factory('WFC_flt_4')
WFC_flt_5 = orm_factory('WFC_flt_5')
WFC_flt_6 = orm_factory('WFC_flt_6')

WFC_flc_0 = orm_factory('WFC_flc_0')
WFC_flc_1 = orm_factory('WFC_flc_1')
WFC_flc_2 = orm_factory('WFC_flc_2')
WFC_flc_3 = orm_factory('WFC_flc_3')
WFC_flc_4 = orm_factory('WFC_flc_4')
WFC_flc_5 = orm_factory('WFC_flc_5')
WFC_flc_6 = orm_factory('WFC_flc_6')

```

Fig. 18: An example of how the `orm_factory` function is called to create class definitions for the header tables.

into the `acsq1` database and to create the ‘Quick-look’ JPEGs and thumbnails. The ingestion algorithm is encapsulated within a collection of modules in the `acsq1` package, namely the `acsq1.ingest.ingest` and `acsq1.scripts.ingest_production.py` modules (further discussed in section 3.10). Collectively, we refer to this collection of modules as the “ingestion software”. We describe the overall algorithm of the ingestion software below.

1. Identify newly available public ACS data in the filesystem: This is accomplished by comparing the list of rootnames in the filesystem with the list of rootnames in the master table of the database. Any rootnames that

exist in the filesystem but not in the database are considered new rootnames to be ingested.

2. Loop over each rootname (in a parallelized manner):

The ingest.py module takes a single rootname as input, such that if there are multiple rootnames to be ingested, the calls to the ingest.py module can be parallelized over many CPUs. The ingestion of one rootname does not depend on the ingestion of another, and there is no importance to the order in which files are ingested. Note that steps 3 through 9 are written such that a single rootname is being ingested (i.e. inside of the loop).

3. Update the master table with information about the rootname:

At this point, the master table can be updated with data pertaining to the rootname. A generic insert_or_update function was written (available in the acsql.utils.utils module) to determine if an entry should be inserted (in the case of first-time ingestion) or updated (in the case of re-ingestion). This function uses various SQLAlchemy methods and the class definitions described in section 3.6 to perform the insert or update operation. The insert_or_update function is shown in Figure 19.

4. Loop over the available filetypes for the given rootname: The available filetypes are determined by traversing down a level in the tree structure of the filesystem and identifying which files are present. Once determined, the ingestion algorithm processes each <rootname>_<filetype>.fits file individually. Note that steps 5 through 9 are written such that a single filetype is being ingested (i.e. inside of the next nested loop).

5. Create a Python dictionary with metadata about the file: To reduce the amount of variables passed around to functions, a data container in the form of a Python dictionary data type is created to hold metadata needed by the remainder of the ingestion process. We refer to this data container as the file_dict. The file_dict contains metadata such as the absolute path of the file in the filesystem, the filetype, the available FITS file extensions of the file, and the absolute paths to where the Quicklook JPEGs and thumbnails will be written.

6. For each FITS file extension, extract the header information and update the appropriate header table in the acsql database: The header information is read into a Python dictionary via the astropy.io.fits module. Besides some minor fixes for a few corner cases (such as converting hyphens to underscores in header keys as to avoid Python errors), it is rather trivial to perform an insert or update operation via the insert_or_update function (see Figure 19).

7. Update the datasets table for the given filetype:

At this point, an entry in the datasets table is either inserted (if it is the first filetype for the rootname being ingested), or updated (if a filetype under the same rootname had already been ingested).

8. If the filetype is either raw, flt, or flc, then create a Quicklook JPEG image: JPEGs are produced only for raw, flt, and flc filetypes, since these are the filetypes that contain two-dimensional image data. The image data are read into multidimensional numpy array data types via the astropy.io.fits module. The data are then rescaled as to avoid an undesirable image stretch caused by extremely

```
def insert_or_update(table, data_dict):
    """Insert or update a record in the given ``table`` with the data
    in the ``data_dict``.

    A record is inserted if the primary key of the record does not
    already exist in the ``table``. A record is updated if it does
    already exist.

    Parameters
    -----
    table : str
        The name of the table to insert/update into.
    data_dict : dict
        A dictionary containing the data to insert/update.
    """

    table_obj = getattr(acsql.database.database_interface, table)
    session, base, engine = acsql.database.database_interface.\
        load_connection(SETTINGS['connection_string'])

    # Check to see if a record exists for the rootname
    query = session.query(table_obj)\.
        filter(getattr(table_obj, 'rootname') == data_dict['rootname'])
    query_count = query.count()

    # If there are no results, then perform an insert
    if not query_count:
        tab = Table(table.lower(), base.metadata, autoload=True)
        insert_obj = tab.insert()
        try:
            insert_obj.execute(data_dict)
        except (DataError, IntegrityError, InternalError) as e:
            logging.warning('\tUnable to insert {} into {}: {}'.format(
                data_dict['rootname'], table, e))

    else:
        query.update(data_dict)

    session.commit()
    session.close()
    engine.dispose()
```

Fig. 19: The insert_or_update function from the acsql.utils.utils module, used at various times during the data ingestion process to determine if an entry should be inserted or updated in the acsql database.

high or low-valued pixels, and saved to a .jpg format. The JPEGs are saved to the jpgs/ portion of the acsql filesystem (described in section 3.4).

9. If the filetype is flt, then create a Quicklook thumbnail image: Thumbnail images are only produced for flt filetypes since they are only meant to be viewed as a means to discover/identify the larger JPEG images via the acsql web application. Thumbnails are generated simply by reading in the corresponding flt JPEG and resizing it to 128x128 pixels. The thumbnails are saved to the thumbnails/ portion of the acsql filesystem (described in section 3.4).

The modules of the ingestion software are intended to be executed daily (as an automatically-spawned process) to keep the acsql application up-to-date on any public data as it becomes available.

```

08/15/2017 11:05:26 INFO: User: bourque
08/15/2017 11:05:26 INFO: System: plhstins1.stsci.edu
08/15/2017 11:05:26 INFO: Python Version: 3.5.2
08/15/2017 11:05:26 INFO: Python Path: /envs/anaconda3/envs/astroconda3/bin/python
08/15/2017 11:05:26 INFO: Numpy Version: 1.11.2
08/15/2017 11:05:26 INFO: Numpy Path: /site-packages/numpy
08/15/2017 11:05:26 INFO: Astropy Version: 1.2.1
08/15/2017 11:05:26 INFO: Astropy Path: /site-packages/astropy
08/15/2017 11:05:26 INFO: SQLAlchemy Version: 1.1.4
08/15/2017 11:05:26 INFO: SQLAlchemy Path: /site-packages/SQLAlchemy-1.1.4-py3.5-linux-x86_64.egg/sqlalchemy
08/15/2017 11:05:26 INFO: Gathering files to ingest
08/15/2017 11:06:00 INFO: j8zh21xv: Updated master table.
08/15/2017 11:06:01 INFO: j8zh21xv: Updated HRC_flt_0 table.
08/15/2017 11:06:01 INFO: j8zh21xv: Updated HRC_flt_1 table.
08/15/2017 11:06:01 INFO: j8zh21xv: Updated HRC_flt_2 table.
08/15/2017 11:06:01 INFO: j8zh21xv: Updated HRC_flt_3 table.
08/15/2017 11:06:01 INFO: j8zh21xv: Updated datasets table for flt.
08/15/2017 11:06:01 INFO: j8zh21xv: Creating JPEG
08/15/2017 11:06:02 INFO: j8zh21xv: Creating Thumbnail
08/15/2017 11:06:02 INFO: j8zh21xv: Updated HRC_raw_0 table.
08/15/2017 11:06:02 INFO: j8zh21xv: Updated HRC_raw_1 table.
08/15/2017 11:06:03 INFO: j8zh21xv: Updated HRC_raw_2 table.
08/15/2017 11:06:03 INFO: j8zh21xv: Updated HRC_raw_3 table.
08/15/2017 11:06:03 INFO: j8zh21xv: Updated datasets table for raw.
08/15/2017 11:06:03 INFO: j8zh21xv: Creating JPEG
08/15/2017 11:06:04 INFO: j8zh21xv: Updated HRC_spt_0 table.
08/15/2017 11:06:05 INFO: j8zh21xv: Updated HRC_spt_1 table.
08/15/2017 11:06:05 INFO: j8zh21xv: Updated datasets table for spt.
08/15/2017 11:06:05 INFO: j8zh21xv: End ingestion

```

Fig. 20: An example log file for the ingestion of a single file (j8zh21xv).

3.8 Data ingestion software: logging

Since the data ingestion software is intended to be executed by an automatic process and not by a human, we implemented a system to log the status of the ingestion to an output text file to be analyzed at a later time. Such log files can be used to assess if there were any issues with the ingestion process, such as if a new header keyword had appeared (requiring an update to the appropriate header table in the database). An example log file showing the ingestion of a single rootname (j8zh21xv) is provided in Figure 20.

When the ingestion software executes, it creates an empty log file is created with the filename <module_name>_<timestamp>.log, where <module_name> is the name of the ingestion module (in production, this is ingest_production.py, as will be discussed in section 3.10), and <timestamp> is the current time in the format YYYY-MM-DD-HH-MM. The naming convention of the log file allows system maintainers to determine which log file corresponds to a specific ingestion run.

Next, the Python logging module configures the format of the log statements by (1) setting the default logging level to INFO (meaning that, unless otherwise specified, each call to logging by the ingestion module will result in an INFO statement), (2) setting the timestamp format to YYYY-MM-DD HH:MM:SS, and (3) setting the logging message format to <timestamp> <level>: <message>.

With the logging settings configured, any call to the logging module within the ingestion software results in a log statement. For example, the command `logging.info('Gathering files to ingest')` results in a timestamped log message, e.g. 08/15/2017

11:05:26 INFO: Gathering files to ingest (as shown in line 10 of Figure 20.)

Calls to the logging module are strategically placed within the ingestion software to provide enough context of the status of the ingestion without cluttering the log file with too much detail. In most cases, logging statements only occur after a change of state to the filesystem or database (i.e. an updated database table, the creation of a JPEG or thumbnail, etc.).

3.9 Web Application

3.9.1 Overview

The front-end of the acsql application is the web application. The web application is built using Python and Flask, which is a Python based web framework[26]. Currently, the web application has two main features/modes of use: (1) viewing JPEGs and image metadata for any publicly-available ACS raw, flt, and flc image (when applicable), and (2) performing relational queries on the acsql database. To illustrate these features, we show examples of some of the different webpages that make up the web application in Figures 21 through 28, and further describe each below.

Figure 21 shows the acsql homepage. The homepage, as well as all other webpages in the web application, contains a menu bar at the top containing four links: (1) to the database query page, (2) to the archive links page, (3) to the acsql code repository on GitHub, and (4) to the readthedocs documentation page. Clicking the 'ACS Quicklook' button in the menu bar allows the user to return to the homepage, regardless of which webpage they are currently viewing. Additionally, the homepage also contains

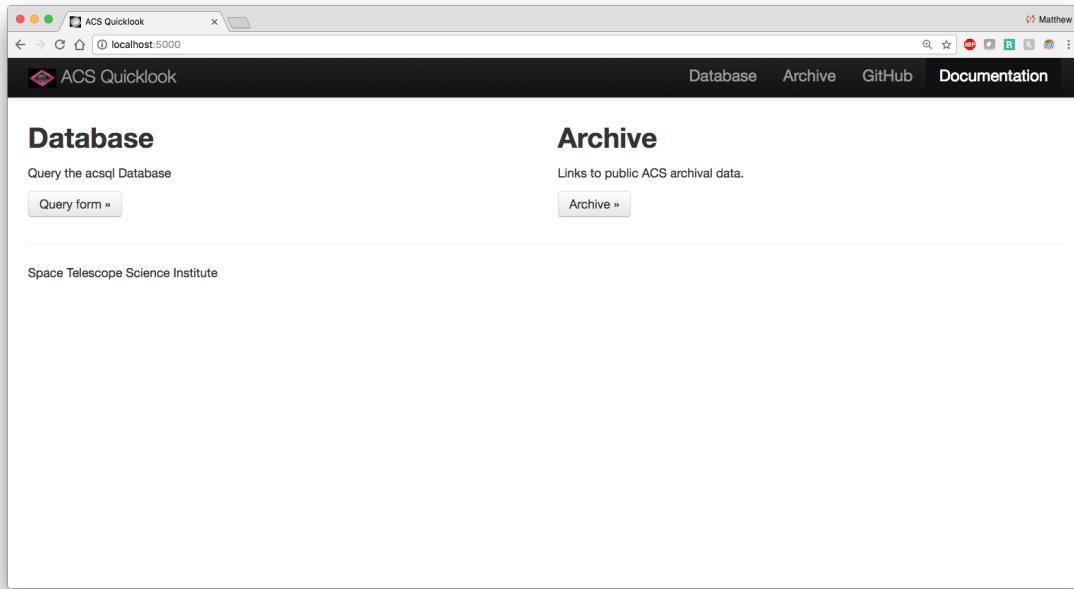


Fig. 21: The homepage of the *acsql* web application.

The screenshot shows a detailed view of the 'Database' query form. It has several input fields and dropdown menus. At the top, there are fields for 'Rootname' (containing 'ex. OMEGACEN, NGC-3603, IRAS05129+5128; single or comma-separated') and 'Target Name' (containing 'ex. OMEGACEN, NGC-3603, IRAS05129+5128; single or comma-separated'). Below these are fields for 'Proposal ID' (with a dropdown for date format), 'Date Observed' (with a dropdown for date format), and 'Exposure Time' (with a dropdown for date format). There are also dropdowns for 'Observation Type' (IMAGING, SPECTROSCOPIC, CORONOGRAPHIC, INTERNAL) and 'Aperture' (WFC, WFC-FIX, WFC1, WFC1-1K). The 'Proposal Type' section lists various mission codes with checkboxes: GO, GTO/ACS, CAL/ACS, SM3/ACS, SM3/ERO, SNAP, GO/PAR, GO/DD, GTO/COS, CAL/OTA, ENG/ACS, NASA, SM4/ACS, SM4/ERO, SM4/COS, CALWFC3, and CAL/STIS. The 'Detector' section lists WFC, HRC, and SBC with checkboxes. The 'Filter1' and 'Filter2' sections list various filters with checkboxes: F115LP, F122M, F125LP, F140LP, F220W, F250W, F330W, and F344N. The 'Image Type' section lists BIAS, DARK, FLAT, and EXT with checkboxes. Below these are fields for 'PI Last Name' and 'PI First Name', both with dropdowns for 'Single or comma-separated'. The 'Output Columns' section contains a grid of checkboxes for selecting specific database fields. The 'Output Format' section includes radio buttons for 'HTML table', 'CSV', and 'Thumbnails', with 'Thumbnails' selected. At the bottom are 'Submit' and 'Reset' buttons.

Fig. 22: A portion of the database query page of the *acsql* web application.

button-type links to the database query page and the archive links page.

Figure 22 shows an example of the database query page. This page allows users to fill out a form that in turn creates a query of the `acsq1` database and executes on hitting the ‘Submit’ button at the bottom of the page. Note that the form contains only a subset of the many possible database parameters; we limited the database query page to only options that we deemed to be particularly useful by the user base. As such, we expect that this page be further expanded and/or modified in the future as more use cases become apparent.

In this example, we use the form to find all data taken with the WFC detector, are of Observation Type IMAGING, and were observed by the Principle Investigator Zolt Levay⁵. Note that there are three output options: (1) an HTML table, which returns a webpage showing the selected ‘Output Columns’ (in this case, the Rootname, PI last/first name, Proposal ID, Aperture, Date/Time of observation, and the Target Name; see Figure 23), (2) a CSV file downloaded to the user’s machine containing a comma-separated table of the selected output columns (see Figure 24), or (3) a webpage showing the thumbnail images of all of the resulting data (see Figure 25).

Figure 26 shows the archive links page. This page contains links to every proposal that contains publicly available ACS data via the 4-5 digit PROPOSID. Clicking one of the links opens a new window to a ‘view proposal’ webpage. Similar to the database query results shown in Figure 25, the ‘view proposal’ webpage shows thumbnail images of each ACS dataset available in the given proposal. The ‘view proposal’ pages contain several buttons and filters near the top of the page; using these will sort and/or filter out the thumbnail images based on the chosen parameters. Figure 27 shows an example of this webpage, using proposal 14039.

In the ‘view proposal’ webpages, users may click on any one of the thumbnail images to bring up the ‘view image’ webpage for that image. An example of this webpage is shown in Figure 28, using the dataset `jcs718kmq`. This webpage contains the Quicklook JPEG of the image along with useful metadata of the dataset near the top of the page. It also contains an additional link (`proposal`) to view the other images within the particular proposal via the appropriate ‘view proposal’ webpage, as well as links to view the corresponding RAW or FLC JPEG image (when applicable).

3.9.2 Implementation

The web application is implemented by several Python modules, namely:

- `acsq1_webapp.py`
- `data_containers.py`
- `form_options.py`
- `query_form.py`
- `query_lib.py`

as well as several subdirectories containing useful static files, style sheets, and HTML templates, namely:

5. Zolt Levay is a Science Visuals Developer at STScI, and has been the PI of many programs that have acquired images for HST public releases.

- `static/css/*.css`
- `static/js/*.js`
- `static/img/*`
- `templates/*.html`

Below we further describe these modules, directories, and files, and how they relate to each other.

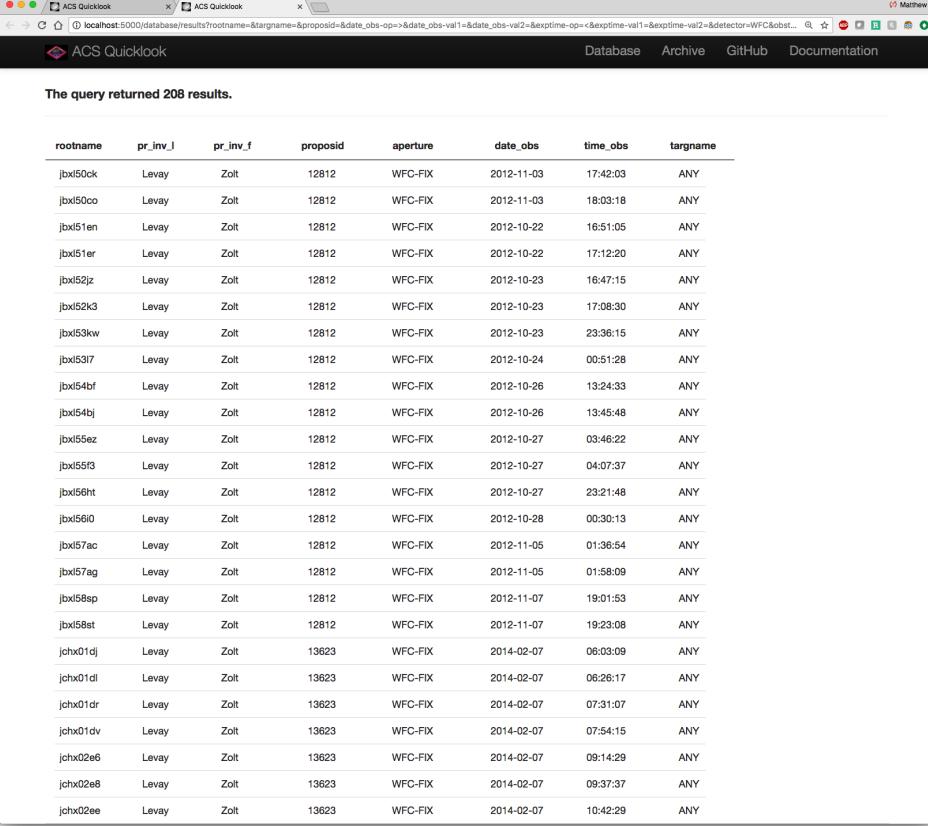
The `acsq1_webapp` module is the main module for running the `acsq1` web application; this module must be executed in order to serve the web application (either on a dedicated web server, or locally on a user’s machine and accessed through the user’s `localhost`). This module uses the Flask web framework to receive incoming HTML requests. It is structured such that each `acsq1` webpage relates to an individual function within `acsq1_webapp` (for example, accessing the `/archive/` webpage results in the execution of the `archive` function). Each of these functions contain the appropriate Python logic and rendering of HTML templates to return the desired webpage.

Much of the functionality required to gather data needed for the various `acsq1` webpages (e.g. lists of images, image metadata, paths to JPEG images on disk, etc.) is imported into `acsq1_webapp` from the `data_containers.py` module. That is to say, many of the functions within the `acsq1_webapp` module call functions from the `data_containers` module to gather the data that is eventually passed on to the HTML template. This design choice was made in order to separate the data from the functionality, often regarded as a best practice in software engineering.

Similarly, the `form_options.py` module is a container module for storing the possible form options (e.g. apertures, filters, output column, proposal types, etc.) for the database query form page. These data could have been defined within the `acsq1_webapp` module, but we chose to separate out these data.

The `query_form.py` and `query_lib.py` modules are used extensively to render the database query webpage and convert a completed form into an executable query on the `acsq1` database, respectively. The `query_form` module contains several class objects for supplying specific types of form fields (e.g. check box fields, text fields, multiple select fields) as well as functions to validate form entries. Several of the form objects are extensions of components (via subclassing) provided by the `wtforms` library, which is a form validation and rendering library for Python web development (`wtforms`)[27]. The `query_lib` module contains functions that parse, build, validate, and return `SQLAlchemy` query objects from the results of a submitted database query form. These queries are then executed on the `acsq1` database and results are used to render the database query result HTML, CSV, and/or thumbnail webpages.

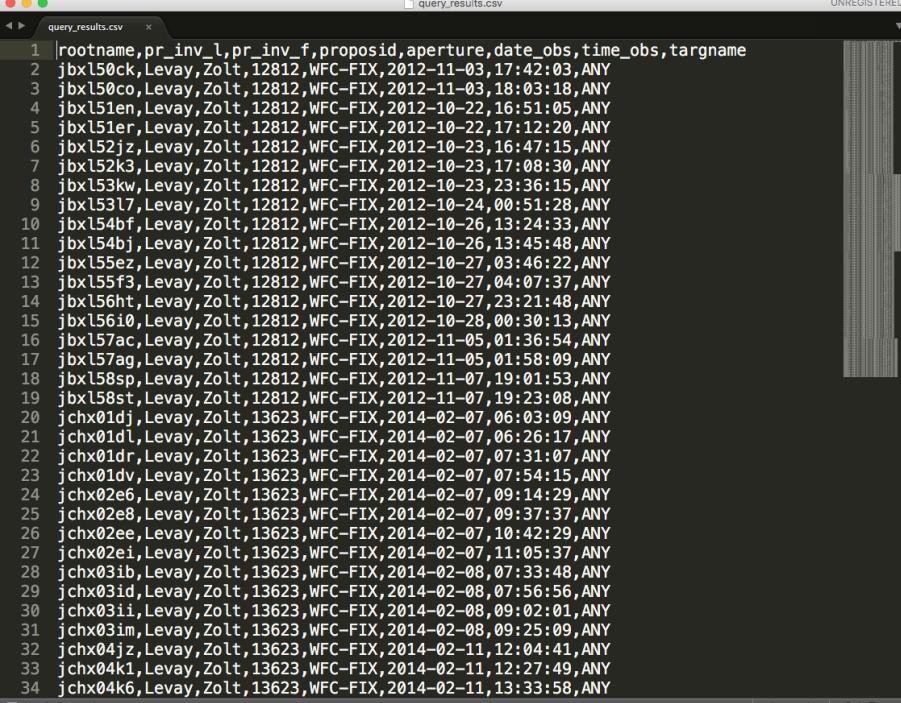
Like many web applications, we also store static files such as HTML templates, CSS templates, and static images in separate `static/` and `templates/` directories. The `static/css/` directory contains the CSS templates used in the HTML templates. Similarly, the `static/js/` directory contains several JavaScript libraries, also used in the HTML templates. Lastly, the `templates/` directory contains the HTML templates used to render the various webpages of the web application. There is one template for each type



The screenshot shows a web browser window titled "ACS Quicklook" with the URL "localhost:5000/database/results?rootname=&targname=&proposalid=&date_obs-op>&date_obs-val1=&date_obs-val2=&exptime-op<&exptime-val1=&exptime-val2=&detector=WFC&obst...". The page displays a table of query results with the following columns: rootname, pr_inv_l, pr_inv_f, proposid, aperture, date_obs, time_obs, and targname. The results list various observations made by different instruments (Levay, Zolt) over several months in 2012 and 2014.

rootname	pr_inv_l	pr_inv_f	proposid	aperture	date_obs	time_obs	targname
jbxl50ck	Levay	Zolt	12812	WFC-FIX	2012-11-03	17:42:03	ANY
jbxl50co	Levay	Zolt	12812	WFC-FIX	2012-11-03	18:03:18	ANY
jbxl51en	Levay	Zolt	12812	WFC-FIX	2012-10-22	16:51:05	ANY
jbxl51er	Levay	Zolt	12812	WFC-FIX	2012-10-22	17:12:20	ANY
jbxl52jz	Levay	Zolt	12812	WFC-FIX	2012-10-23	16:47:15	ANY
jbxl52k3	Levay	Zolt	12812	WFC-FIX	2012-10-23	17:08:30	ANY
jbxl53kw	Levay	Zolt	12812	WFC-FIX	2012-10-23	23:36:15	ANY
jbxl53l7	Levay	Zolt	12812	WFC-FIX	2012-10-24	00:51:28	ANY
jbxl54bf	Levay	Zolt	12812	WFC-FIX	2012-10-26	13:24:33	ANY
jbxl54bj	Levay	Zolt	12812	WFC-FIX	2012-10-26	13:45:48	ANY
jbxl55ez	Levay	Zolt	12812	WFC-FIX	2012-10-27	03:46:22	ANY
jbxl55f3	Levay	Zolt	12812	WFC-FIX	2012-10-27	04:07:37	ANY
jbxl56ht	Levay	Zolt	12812	WFC-FIX	2012-10-27	23:21:48	ANY
jbxl56i0	Levay	Zolt	12812	WFC-FIX	2012-10-28	00:30:13	ANY
jbxl57ac	Levay	Zolt	12812	WFC-FIX	2012-11-05	01:36:54	ANY
jbxl57ag	Levay	Zolt	12812	WFC-FIX	2012-11-05	01:58:09	ANY
jbxl58sp	Levay	Zolt	12812	WFC-FIX	2012-11-07	19:01:53	ANY
jbxl58st	Levay	Zolt	12812	WFC-FIX	2012-11-07	19:23:08	ANY
jchx01dj	Levay	Zolt	13623	WFC-FIX	2014-02-07	06:03:09	ANY
jchx01dl	Levay	Zolt	13623	WFC-FIX	2014-02-07	06:26:17	ANY
jchx01dr	Levay	Zolt	13623	WFC-FIX	2014-02-07	07:31:07	ANY
jchx01dv	Levay	Zolt	13623	WFC-FIX	2014-02-07	07:54:15	ANY
jchx02e6	Levay	Zolt	13623	WFC-FIX	2014-02-07	09:14:29	ANY
jchx02e8	Levay	Zolt	13623	WFC-FIX	2014-02-07	09:37:37	ANY
jchx02ee	Levay	Zolt	13623	WFC-FIX	2014-02-07	10:42:29	ANY

Fig. 23: The results of the database query shown in Figure 22 in the form of an HTML table.



The screenshot shows a terminal window titled "query_results.csv" with the file content displayed. The file contains 34 lines of comma-separated values (CSV) representing the same dataset as Figure 23. Each line corresponds to a specific observation entry, including the rootname, proposid, aperture, date_obs, and time_obs fields.

```

1 |rootname,pr_inv_l,pr_inv_f,proposid,aperture,date_obs,time_obs,targname
2 jbxl50ck,Levay,Zolt,12812,WFC-FIX,2012-11-03,17:42:03,ANY
3 jbxl50co,Levay,Zolt,12812,WFC-FIX,2012-11-03,18:03:18,ANY
4 jbxl51en,Levay,Zolt,12812,WFC-FIX,2012-10-22,16:51:05,ANY
5 jbxl51er,Levay,Zolt,12812,WFC-FIX,2012-10-22,17:12:20,ANY
6 jbxl52jz,Levay,Zolt,12812,WFC-FIX,2012-10-23,16:47:15,ANY
7 jbxl52k3,Levay,Zolt,12812,WFC-FIX,2012-10-23,17:08:30,ANY
8 jbxl53kw,Levay,Zolt,12812,WFC-FIX,2012-10-23,23:36:15,ANY
9 jbxl53l7,Levay,Zolt,12812,WFC-FIX,2012-10-24,00:51:28,ANY
10 jbxl54bf,Levay,Zolt,12812,WFC-FIX,2012-10-26,13:24:33,ANY
11 jbxl54bj,Levay,Zolt,12812,WFC-FIX,2012-10-26,13:45:48,ANY
12 jbxl55ez,Levay,Zolt,12812,WFC-FIX,2012-10-27,03:46:22,ANY
13 jbxl55f3,Levay,Zolt,12812,WFC-FIX,2012-10-27,04:07:37,ANY
14 jbxl56ht,Levay,Zolt,12812,WFC-FIX,2012-10-27,23:21:48,ANY
15 jbxl56i0,Levay,Zolt,12812,WFC-FIX,2012-10-28,00:30:13,ANY
16 jbxl57ac,Levay,Zolt,12812,WFC-FIX,2012-11-05,01:36:54,ANY
17 jbxl57ag,Levay,Zolt,12812,WFC-FIX,2012-11-05,01:58:09,ANY
18 jbxl58sp,Levay,Zolt,12812,WFC-FIX,2012-11-07,19:01:53,ANY
19 jbxl58st,Levay,Zolt,12812,WFC-FIX,2012-11-07,19:23:08,ANY
20 jchx01dj,Levay,Zolt,13623,WFC-FIX,2014-02-07,06:03:09,ANY
21 jchx01dl,Levay,Zolt,13623,WFC-FIX,2014-02-07,06:26:17,ANY
22 jchx01dr,Levay,Zolt,13623,WFC-FIX,2014-02-07,07:31:07,ANY
23 jchx01dv,Levay,Zolt,13623,WFC-FIX,2014-02-07,07:54:15,ANY
24 jchx02e6,Levay,Zolt,13623,WFC-FIX,2014-02-07,09:14:29,ANY
25 jchx02e8,Levay,Zolt,13623,WFC-FIX,2014-02-07,09:37:37,ANY
26 jchx02ee,Levay,Zolt,13623,WFC-FIX,2014-02-07,10:42:29,ANY
27 jchx02ei,Levay,Zolt,13623,WFC-FIX,2014-02-07,11:05:37,ANY
28 jchx03ib,Levay,Zolt,13623,WFC-FIX,2014-02-08,07:33:48,ANY
29 jchx03id,Levay,Zolt,13623,WFC-FIX,2014-02-08,07:56:56,ANY
30 jchx03ii,Levay,Zolt,13623,WFC-FIX,2014-02-08,09:02:01,ANY
31 jchx03im,Levay,Zolt,13623,WFC-FIX,2014-02-08,09:25:09,ANY
32 jchx04jz,Levay,Zolt,13623,WFC-FIX,2014-02-11,12:04:41,ANY
33 jchx04k1,Levay,Zolt,13623,WFC-FIX,2014-02-11,12:27:49,ANY
34 jchx04k6,Levay,Zolt,13623,WFC-FIX,2014-02-11,13:33:58,ANY

```

Fig. 24: The results of the database query shown in Figure 22 in the form of an CSV file.

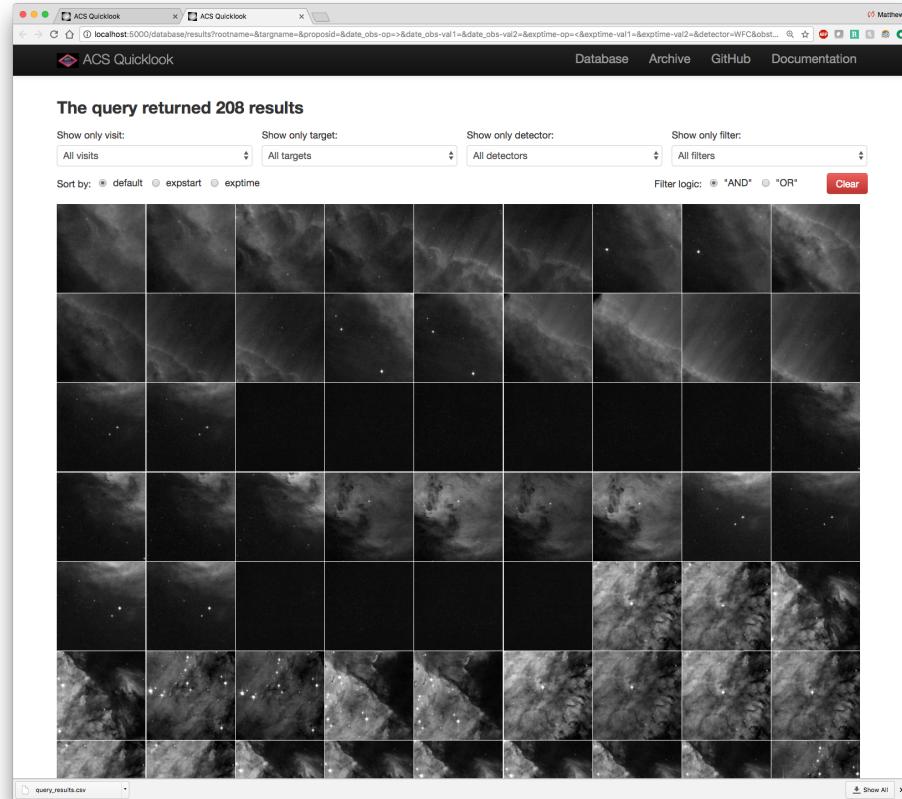


Fig. 25: *The results of the database query shown in Figure 22 in the form of thumbnail images.*

The screenshot shows a web browser window titled "ACS Quicklook" displaying the "Archive" links. The main content area is titled "ACS Archive" and contains a table with 208 rows of data. Each row represents an observation, with columns for ID (e.g., 8183, 9503, 9830), Target (e.g., 10188, 10459, 10626, 10907, 11655, 12250, 12759, 13407, 13942), and various dates and times. The table has a light gray background with alternating row colors. The top right corner of the browser window shows the user's name "Matthew".

8183	9503	9830	10188	10459	10626	10907	11655	12250	12759	13407	13942
8947	9558	9831	10189	10460	10627	10909	11658	12253	12760	13410	13945
8948	9560	9835	10190	10461	10628	10910	11663	12254	12780	13412	13952
8992	9562	9836	10192	10466	10629	10911	11669	12256	12781	13419	13953
8993	9563	9837	10195	10471	10630	10913	11670	12257	12782	13420	13954
9005	9564	9838	10196	10472	10631	10915	11675	12262	12783	13422	13955
9006	9565	9842	10198	10473	10632	10917	11676	12270	12787	13425	13956
9008	9566	9847	10199	10474	10633	10918	11677	12271	12788	13433	13957
9009	9567	9851	10200	10475	10634	10919	11679	12273	12789	13435	13958
9010	9568	9853	10201	10476	10635	10920	11681	12282	12790	13437	13959
9011	9574	9854	10204	10483	10695	10921	11683	12286	12791	13439	13960
9012	9575	9856	10205	10486	10696	10922	11684	12292	12811	13441	13961
9013	9578	9857	10206	10487	10697	10923	11688	12297	12812	13442	13962
9014	9583	9860	10207	10488	10698	10991	11689	12309	12817	13443	13964
9015	9584	9861	10210	10489	10701	10992	11691	12310	12858	13449	13965
9016	9586	9862	10213	10490	10703	10994	11695	12311	12860	13459	13966
9017	9587	9863	10214	10491	10705	10996	11704	12313	12866	13461	14034
9018	9588	9864	10216	10492	10707	10997	11710	12317	12871	13463	14035
9019	9647	9869	10217	10493	10710	11003	11711	12323	12872	13470	14037
9020	9648	9872	10227	10494	10711	11004	11713	12326	12875	13474	14038
9022	9649	9873	10228	10495	10713	11005	11715	12327	12877	13476	14039
9023	9650	9877	10229	10496	10715	11008	11718	12328	12878	13479	14056
9024	9651	9884	10231	10497	10718	11011	11722	12331	12880	13480	14057
9025	9652	9889	10233	10498	10720	11013	11724	12362	12884	13483	14061
9026	9654	9890	10235	10499	10722	11015	11731	12367	12908	13485	14063
9027	9655	9891	10237	10500	10729	11020	11734	12368	12911	13493	14072
...

Fig. 26: *The archive links webpage of the acsql web application.*

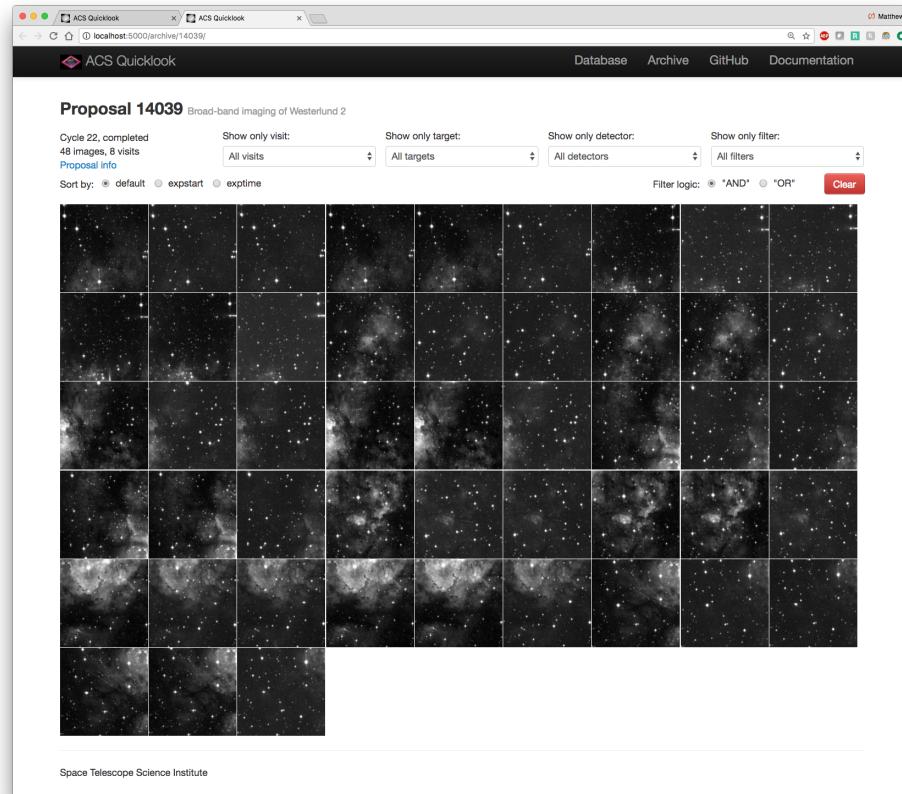


Fig. 27: An example ‘view proposal’ webpage, using proposal 14039.

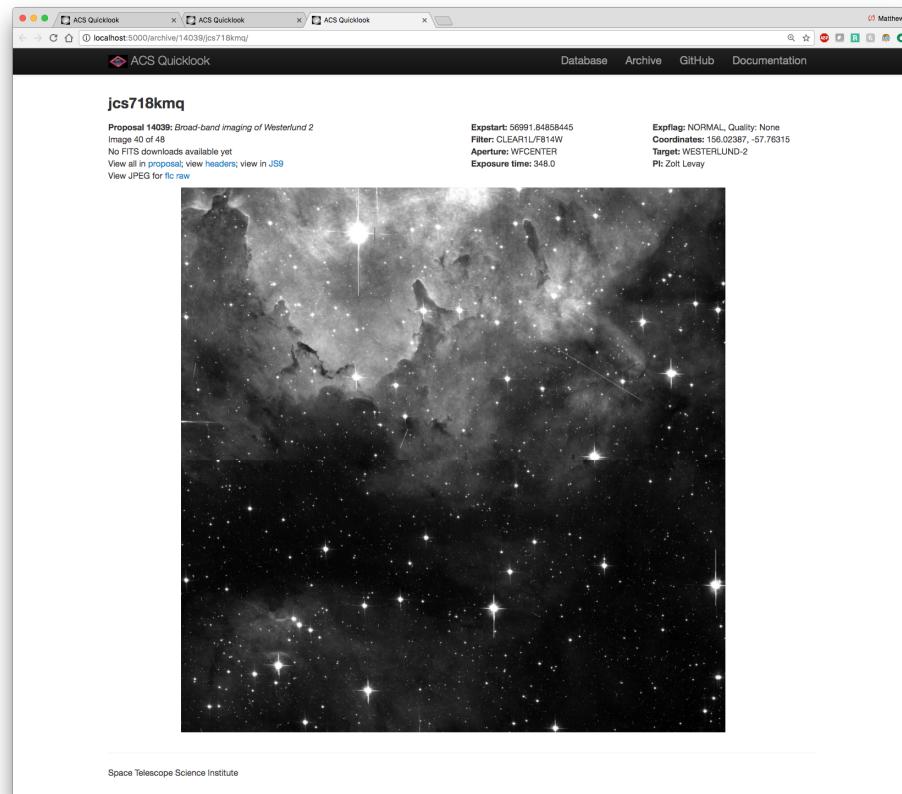


Fig. 28: An example ‘view image’ webpage, using dataset *jcs718kmq*.

of webpage, as well as a template that houses HTML that is common to each webpage (e.g. the menu bar), named `base.html`. Aside from some minor tweaks, we use templates from Bootstrap, which is a front-end component library and open source toolkit for developing HTML, CSS, and JS[28].

3.10 acsql Package

All software associated with the `acsql` project is contained within a single git repository (also named “`acsql`”), which we refer to as the “`acsql` library”, or “`acsql` package”. The package layout is shown below:

```
acsql/
    LICENSE
    README.md
    MANIFEST.in
    setup.py
    paper/
        ...
        presentation/
        ...
    docs/
        Makefile
        requirements.rst
        source/
            conf.py
            database.rst
            index.rst
            ingest.rst
            scripts.rst
            utils.rst
            website.rst
    acsql/
        __init__.py
        database/
            __init__.py
            database_interface.py
            make_tabledefs.py
            queries.py
            reset_database.py
            table_definitions/
                *.txt
            update_tabledefs.py
        ingest/
            __init__.py
            ingest.py
            make_file_dict.py
            make_jpeg.py
            make_thumbnail.py
        scripts/
            __init__.py
            ingest_production.py
        utils/
            __init__.py
            config.yaml
            utils.py
        website/
            __init__.py
            acsql_webapp.py
            data_containers.py
            form_options.py
            query_form.py
            query_lib.py
            static/
                css/
                    *.css
                img/
                    jpgs
                    thumbnails
                js/
                    *.js
            templates/
                *.html
```

We now provide a brief description of each package component. For further details on each Python module within the `acsql` package, readers are encouraged to view the official API documentation hosted at <http://acsql.readthedocs.io/>.

`LICENSE`: A BSD 3-Clause license, which states that the `acsql` package is an open source software package and may be used and redistributed.

`README.md`: A `README` file that describes how to install and use the `acsql` package.

`MANIFEST.in`: A list of static files to be included in the tarball file when the user installs the `acsql` package.

`setup.py`: The `acsql` package installation script. Executing this script with `python setup.py install` installs the package into the software environment.

`paper/`: A subdirectory which contains all materials used for the creation of this document.

`presentation/`: A subdirectory which contains all materials used for the creation of the Towson University COSC 880 presentation.

`docs/`: A subdirectory which contains all materials used for the creation of the `sphinx` API documentation hosted on `readthedocs` (see section 3.2).

`Makefile`: A `make` script that is used to build the `sphinx` API documentation from the source reStructured Text files (mentioned below) (see section 3.2).

`requirements.rst`: A list of `acsql` package dependencies, used by `readthedocs` to build a virtual machine that constructs the resulting HTML doc pages (see section 3.2).

`source/`: A subdirectory containing all of the reStructured Text files used for building the `sphinx` API documentation, one `.rst` file per subpackage, including a master `index.rst` file (see section 3.2).

`acsql/`: A subdirectory containing all Python code that is part of the official `acsql` library. This is the top level of the importable `acsql` package.

`__init__.py`: A Python file indicating that the subdirectory is part of the overall `acsql` package.

`database/`: The database subpackage, containing Python modules that pertain to the `acsql` database (see sections 3.5 and 3.6).

`database_interface.py`: The Python module for constructing and connecting to the `acsql` database (see section 3.6).

`make_tabledefs.py`: The Python module for creating the table definition text files (see section 3.6).

`queries.py`: A Python module containing several examples of queries that can be used with the `acsq1` database.

`reset_database.py`: A Python module that allows the user to ‘reset’ the `acsq1` database (i.e. delete all tables, then create all tables).

`table_definitions/`: A subdirectory containing all of the `<detector>_<filetype>_<extension>.txt` text files, each of which contain a list of header keys along with their datatypes (see section 3.6).

`update_tabledefs.py`: A Python module that allows the user to update the `table_definitions/` text files with new header keywords (see section 3.6).

`ingest/`: The `ingest` subpackage, containing Python modules for ingesting new data into the `acsq1` application, including database updates and the creation of JPEGs/thumbnails (see section 3.7).

`ingest.py`: A Python module for performing the ingestion of a single file (see section 3.7).

`make_file_dict.py`: A Python module for creating a `file_dict` for an individual file (see section 3.7).

`make_jpeg.py`: A Python module for creating a JPEG image from an individual file (see section 3.7).

`make_thumbnail.py`: A Python module for creating a thumbnail image from an individual JPEG (see section 3.7).

`scripts/`: The `scripts` subpackage, containing Python modules for ingesting multiple files from the `acsq1` filesystem. This also serves as storage place for possible future instrument calibration and monitoring routines (see section 3.7).

`ingest_production`: The Python module for ingesting new ACS data as it becomes publicly available, intended to be executed regularly (see section 3.7).

`utils/`: The `utils` subpackage, containing Python modules that are useful for general `acsq1` operations (e.g. configuring logging, supplying hard-coded instrument configurations, etc.) as well as a configuration file for storing sensitive credentials and directory locations.

`config.yaml`: A text file containing hard-coded user-specific directory locations and `acsq1` database credentials. Specifically, it contains values for the `acsq1` database `connection_string`, as well as locations for the `filesystem`, `log_dir`, `jpeg_dir`, and `thumbnail_dir`. The contents of the `config.yaml` file can be imported via the `utils.utils.SETTINGS` dictionary.

`utils.py`: A Python module containing various functions that are generally useful for `acsq1` operations, such as configuring logging, determining if a database entry requires an `insert` or an `update`, and hard-coded Python variables that reflect instrument/system configurations.

`website/`: The `website` subpackage, containing Python modules that are used in the construction and operations of the `acsq1` web application (see section 3.9).

`acsq1_webapp.py`: The main Python module for running the `acsq1` web application, using the Python `Flask` web framework (see section 3.9).

`data_containers.py`: The Python module that contains various functions for returning data to be used by the `acsq1` web application (see section 3.9).

`form_options.py`: A Python module that stores form data for the database query portion of the `acsq1` web application (see section 3.9).

`query_form.py`: A Python module that contains class objects for building the query form for the database query portion of the `acsq1` web application (see section 3.9).

`query_lib.py`: A Python module that contains various functions to support the querying of the `acsq1` database through the `acsq1` web application (see section 3.9).

`static/`: A subdirectory containing static materials used by the `acsq1` web application, such as CSS templates (i.e. `css/`), JavaScript functions (i.e. `js/`), and symbolic links to the JPEGs and thumbnails hosted on the web application (i.e. `img/`) (see section 3.9).

`templates/`: A subdirectory containing HTML templates used to render the various webpages of the `acsq1` web application, one for each page (see section 3.9).

4 RESULTS

The results of the system implementation (described in Chapter 3) were several project deliverables:

- 1) Filesystem
- 2) Database
- 3) Web application
- 4) Software package
- 5) Software documentation

As mentioned in section 1.3.5, the project deliverables will primarily be used by members of the ACS instrument team at STScI, but may also be used by users external to STScI. In the following subsections, we further describe each one of these deliverables.

4.1 Filesystem Deliverable

As described in sections 3.3 and 3.4, the `acsq1` filesystem is comprised of two parts: (1) A filesystem that stores the

archive of publicly-available ACS data (i.e. FITS files), and (2) a filesystem of Quicklook JPEGs and thumbnails.

For the former, we utilized an already-existing filesystem of ACS data internal to STScI known as the “MAST public cache”; this filesystem is organized in a similar way to that shown in Figure 7⁶. Though this service is internal to STScI, it is possible for an external user to reconstruct the filesystem, as all data within the MAST public cache is publicly available to download via the MAST archive (i.e. <https://archive.stsci.edu/>). The location of the filesystem is determined by the `filesystem` parameter in the user-supplied `config.yaml` file (see section 3.10).

Currently, the filesystem consists of ~1,030,000 total files. Figure 29 shows how this breaks down by individual filetype. We see that `spt`, `raw`, and `flt` make up the majority of the files, which is expected considering that these filetypes exist for nearly every ACS observing mode. On the other hand, we see a small amount of `crj` and `crc` files, which is also expected considering that these filetypes are only triggered for specific observing modes.

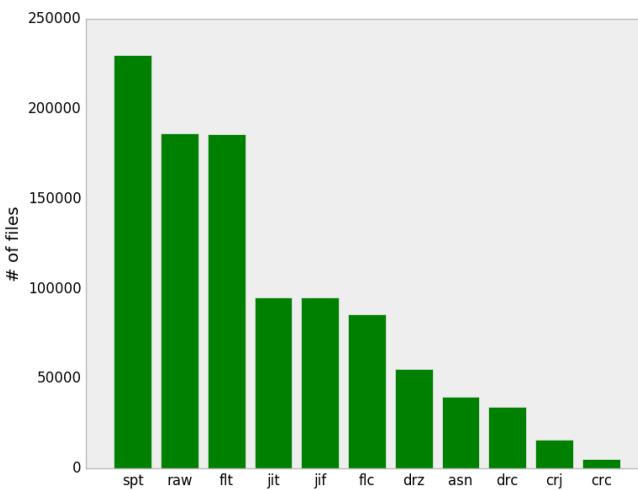


Fig. 29: The number of files in the `acsql` filesystem by filetype.

As for item (2), the filesystem of JPEGs and thumbnails was constructed during the ingestion of all publicly-available ACS data (via `ingest_production.py`, see section 3.9). The location of the JPEG/thumbnaill filesystem is determined by the `jpeg_dir` and `thumbnail_dir` parameters in the `config.yaml` file, respectively. Currently, there are ~457,000 JPEGs and ~185,000 corresponding thumbnail images in the filesystem⁷.

4.2 Database Deliverable

As described in sections 3.5 and 3.6, the `acsql` database is a MySQL database that stores the header information of every public ACS dataset. Currently, this database is hosted on a server that is internal to STScI. However, external users may build their own copy of the database via

6. We omit the mention of the parent directory structure and server names in this document as well as in the `acsql` code repository as to avoid exposing possible sensitive information.

7. recall that thumbnail images are only generated for `flt` filetypes, hence the discrepancy in number

the `ingest_production.py` and `database_interface` modules. All 111 tables of `acsql` database (master, datasets, and 109 header tables) are up to date as of the time of this writing (September 2017). Ideally, the database will be updated via regular executions of `ingest_production.py`.

Figure 30 shows the number of records in each table of the database. Currently, there are ~4,300,000 records in total amongst the 111 tables.

4.3 Web Application Deliverable

As described in section 3.9, the `acsql` web application consists of a series of Python modules and static files. Currently, the web application is not hosted on a dedicated web server (neither externally nor internally to STScI⁸), however, users can operate the application through the user’s `localhost`. Detailed installation and operation instructions for the web application are provided in the official documentation (see section 3.2). In short, users must (1) ensure that they have built (or have access to) the `acsql` filesystem and database, (2) have installed the Python package dependencies, and (3) have filled out the `config.yaml` file (see section 3.10).

4.4 Software Package Deliverable

As described in section 3.1 and 3.10, the `acsql` git repository serves as the software package that contains all of the software and supporting materials needed to operate the `acsql` application. The software package is open-source, and is available to download directly from the GitHub repository (<https://github.com/acsql/>). All future enhancements and bug fixes to the `acsql` application will occur through this version-controlled repository.

4.5 Software Documentation Deliverable

As described in section 3.2, another deliverable for the project is the API documentation that is automatically generated via `sphinx`. The resulting HTML pages are available through the `acsql` package, but are also hosted on `readthedocs`, available at <http://acsql.readthedocs.io/>. This is considered the official software documentation.

5 DISCUSSION

We consider the collection of the features implemented as described in this document to be the “official release” or “1.0 release” of the `acsql` application. This release is tagged as version `v1.0` in the `acsql` GitHub repository. It is with this release that we expect the user base to begin using the application.

Though this is the official release of the application, we expect future development by the author and/or members of the ACS instrument team at STScI. There are several possible features to be added to the application, namely those present in the `wfc3ql` application, as discussed in chapter 2. Furthermore, some or all of the `acsql` application components could be extended to support other instruments and missions. Some of these possible enhancements are discussed in the proceeding sections.

8. At the time of this writing, work is being done to build a web server that is internal to STScI to allow ACS instrument team members to easily access the `acsql` web application. We expect this web server to be operational within the next few months.

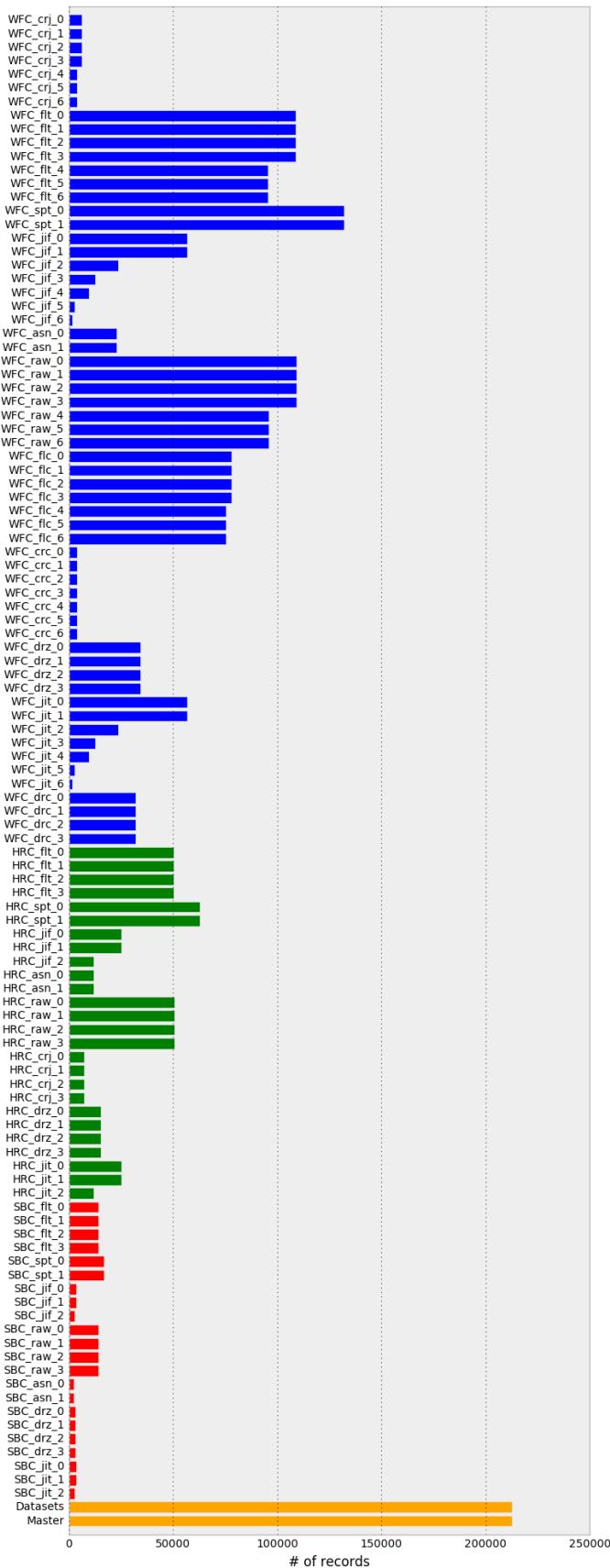


Fig. 30: The number of records in each of the acsql database tables.

5.1 Possible enhancements to the acsql application

As mentioned in chapter 2, there exist several features in the wfc3ql application that could serve as possible enhancements to the acsql application. We elaborate on five of these features below:

1. Including proprietary data in the acsql filesystem:

One of the main features of the wfc3ql application is that it allows users to view and diagnose new data just hours after it gets downlinked from HST. Since most HST data is proprietary up to one year after observation, this introduces a considerable security risk, and thus this feature is only made possible by taking special care of file permissions within the infrastructure of the wfc3ql filesystem and database. Implementing this feature in the acsql application would be advantageous from the perspective of an ACS instrument analyst who wishes to perform analysis on new data. However, allowing for the storage and handling of proprietary data would prohibit the application from being open-source and publicly-available to external users; the application would have to be restricted to ACS instrument team members at STScI.

2. Building automated instrument calibration and monitoring routines:

The wfc3ql application contains several WFC3 instrument calibration and monitoring routines. These routines are in the form of individual Python modules that are executed automatically by cron jobs on a daily, weekly, or monthly basis. Each one of these monitors serves a specific instrument-related purpose, such as monitoring detector hysteresis effects[29], monitoring the dark current of the detector over time[30], and characterizing the behaviour of the instrument Channel Select Mechanism (CSM)[31]. Like WFC3, the ACS instrument also has various detector behaviors and characteristics that could benefit from being automatically monitored or characterized over time. Currently, such efforts are made by individual ACS instrument team members through manual execution of software. However, by using the acsql filesystem, database, and code library, it is possible to automate these tasks, saving employee time resources.

3. Adding a script execution status dashboard:

If there were automated instrument calibration and monitoring routines implemented into the application, it would be useful to be able to visualize if the executions of these scripts were successful or not. The wfc3ql application accomplishes this with a script execution dashboard on the wfc3ql web application homepage. Each automated routine is labeled in the dashboard along with the date/time of last execution and a status column that indicates if the execution completed successfully or not. This helps to easily identify any disruptions in the routine cron jobs.

4. Expanding the database query form:

The database query form webpage on the acsql web application allows user to construct and execute a query to the acsql database using some of the most commonly used observational parameters (e.g. target name, filter, principle investigator name, etc.). However, these parameters are just a small subset of the total collection of parameters available in the acsql database. With the database containing all header keywords, it is possible to extend this query form use some more obscure (but useful) observational parameters. Exam-

ples of these may include telescope telemetry parameters (e.g. instrument temperature, telescope pointing, telescope latitude/longitude, etc.), engineering parameters (e.g. detector gain, detector readnoise, etc.), and photometric parameters (e.g. detector sensitivity, photometric zeropoints, etc.). We expect that the database query webpage in particular will endure the most feature development as the usefulness of certain parameters become evident from the user base.

5. Tracking ACS image anomalies: Another feature of the `wfc3ql` application is the capability to record the occurrences of image anomalies (i.e. features in anomalies that are unexpected or indicate a peculiar behavior of the instrument). The user accomplishes this in the ‘view image’ webpage by selecting a checkbox of the anomaly and hitting a ‘submit’ button. Then, a row is inserted into a separate ‘anomalies’ table of the `wfc3ql` database that indicates that the particular observation contains the particular anomaly. Such a feature could also be implemented in `acsq1` by adding an `anomalies` table to the `acsq1` database schema as well as the appropriate mechanisms to allow this functionality through the `acsq1` web application. Tracking such anomalies would allow ACS instrument analysts to identify images affected by certain anomalies, as well as aid the characterization of instrument anomalies over time.

5.2 Possible extensions to other instruments or missions

This type of database-driven, web-based application for interacting with instrument data is currently only implemented for the WFC3 (`wfc3ql`) and ACS (`acsq1`) instruments of HST. However, the continued success of `wfc3ql` and potential success of `acsq1` creates a strong case for implementing such a system for other instruments and missions. As this project has been fully developed at STScI, which is home of the operations for the Hubble Space Telescope and the forthcoming James Webb Space Telescope (JWST), it is reasonable to consider extending this application to other imaging instruments on HST (e.g. The Space Telescope Imaging Spectrograph; STIS) and the future imaging instruments on JWST.

6 CONCLUSION

The `acsq1` application, a web application built on top of a filesystem and database of ACS instrument data, has been released and is publicly available for use by ACS users. The application provides mechanisms for users to view images and their metadata for every publicly-available ACS dataset dating back to the installation of the instrument in 2002. While this document marks the version 1.0 release of this open-source application, we expect ongoing development to expand the capabilities of the `acsq1` web application. With the modularization and coding standards found in the `acsq1` software, we hope this application can be extended and/or used in reference to support a similar application for the forthcoming James Webb Space Telescope mission.

ACKNOWLEDGMENTS

The authors would like to thank various members of the STScI staff that were extremely valuable for helping make this project come together: members of the

WFC3/Quicklook team, for their years of ongoing development and support of the WFC3/Quicklook application, without which there would be no inspiration to create this similar application (Varun Bajaj, Ariel Bowers, Michael Dulude, Meredith Durbin, Jules Fowler, Catherine Gosmeyer, Heather Gunning, Harish Khandrika, Catherine Martlin, Abhi Rajan, Clare Shanahan, Ben Sunquist, Alex Viana); members of the Instruments Division, who supported the creation of this project (Francesca Boffi, Norman Grogan, Elena Sabbi); members of the Operations and Engineering Division as well as the Information Technology and Services Division who provided valuable resources such as the database server, GitHub repository, web server, and disk space used to store project materials (Alex Yermolaev, Fausat Ogunsanya, Vera Gibbs); Pey-Lian Lim and Roberto Avila, who provided valuable feedback upon beta testing the web application; Sara Ogaz for assistance in developing some components of the `acsq1` database as part of the Towson University COSC 657 Advance Database Management project; and finally, Jules Fowler and Sara Ogaz, who provided insightful comments and suggestions on this document.

REFERENCES

- [1] SM3B, National Aeronautics and Space Administration, Goddard Space Flight Center, [Online; accessed 2017-09-16], available at <https://asd.gsfc.nasa.gov/archive/hubble/missions/sm3b.html>.
- [2] SM4, National Aeronautics and Space Administration, Goddard Space Flight Center, [Online; accessed 2017-09-16], available at <https://asd.gsfc.nasa.gov/archive/hubble/missions/sm4.html>.
- [3] Avila, R., et al., 2017, *ACS Instrument Handbook*, Version 16.0 (Baltimore: STScI)
- [4] *The Barbara A. Mikulski Archive for Space Telescopes*, [Online; accessed 2017-07-30], available at <https://archive.stsci.edu/>.
- [5] *Definition of the Flexible Image Transport System (FITS): The FITS Standard*, 2008, International Astronomical Union FITS Working Group, available at https://fits.gsfc.nasa.gov/standard30/fits_standard30aa.pdf.
- [6] *STScI: Space Telescope Science Institute*, [Online; accessed 2017-09-16], available at www.stsci.edu.
- [7] Smith, E., et al., 2011, *Introduction to the HST Data Handbooks*, Version 8.0 (Baltimore: STScI)
- [8] Lucas, R. A., et al., 2016, *ACS Data Handbook*, Version 8.0 (Baltimore: STScI)
- [9] Robitaille, T.P., et al., 2013, *Astropy: A community Python package for astronomy*, *Astronomy & Astrophysics*, 558, A33.
- [10] *The MAST Portal*, [Online; accessed 2017-09-15], available at <https://mast.stsci.edu/>.
- [11] Bourque, M., et al., 2016, *The Hubble Space Telescope Wide Field Camera 3 Quicklook Project*, *Astronomical Data Analysis Software & Systems Conference Proceedings*, ADASS XXVI, ASP-CS, submitted
- [12] Bourque, M., et al., 2016, *The HST/WFC3 Quicklook Project: A User Interface to Hubble Space Telescope Wide Field Camera 3 Data*, *Proceedings IAU Symposium No. 325*, 2016, Astroinformatics.
- [13] Gosmeyer, C.M., and The Quicklook Team, 2017, *WFC3 Anomalies Flagged by the Quicklook Team*, WFC3 Instrument Science Report, 2017-22, available at <http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2017-22.pdf>
- [14] *git*, [Online; accessed 2017-08-05], available at <https://git-scm.com>.
- [15] *GitHub*, [Online; accessed 2017-08-05], available at <https://github.com>.
- [16] van Rossum, G., 2001, *PEP 8 – Style Guide for Python Code*, Python Developer’s Guide, available at <https://www.python.org/dev/peps/pep-0008/>.
- [17] Goodger, D., 2001, *PEP 257 – Docstring Conventions*, Python Developer’s Guide, available at <https://www.python.org/dev/peps/pep-0257/>.

- [18] *A Guide to NumPy/SciPy Documentation*, [Online; accessed 2017-08-05], available at https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt/
- [19] van der Walt, S., Colbert, C., and Varoquaux, G., 2011, *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering, 13, 22-30.
- [20] Brandi, G., et al., 2007, *Sphinx: Python Documentation Generator*, available at <http://www.sphinx-doc.org/en/stable/>.
- [21] Loper, E., 2004, *Epydoc: Generating API Documentation in Python*, Proceedings of the Second Annual Python Conference, available at <http://epydoc.sourceforge.net/>.
- [22] *Read the Docs*, [Online; accessed 2017-08-05], available at <https://readthedocs.org>.
- [23] *MySQL 5.6 Reference Manual*, Oracle, [Online; accessed 2017-08-13], available at <https://dev.mysql.com/doc/refman/5.6/en/>.
- [24] Bayer, M., 2006, *SQLAlchemy: The database toolkit for Python*, [Online; accessed 2017-02-21], available at <http://www.sqlalchemy.org/>.
- [25] Lemburg, M., 2017, *PEP 249 – Python Database API Specification v2.0*, Python Developer’s Guide, available at <https://www.python.org/dev/peps/pep-0249/>.
- [26] Ronacher, A., et al., 2010, [Online; accessed 2017-09-04], available at <http://flask.pocoo.org/>.
- [27] *wtforms*, [Online; accessed 2017-09-04], available at <https://github.com/wtforms/wtforms/>.
- [28] *Bootstrap*, [Online; accessed 2017-09-04], available at <http://getbootstrap.com/>.
- [29] Bourque, M. and Baggett, S., 2013, *WFC3/UVIS Bowtie Monitor*, WFC3 Instrument Science Report, 2013-09, available at <http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2013-09.pdf>
- [30] Bourque, M. and Baggett, S., 2016, *WFC3/UVIS Dark Calibration: Monitoring Results and Improvements to Dark Reference Files*, WFC3 Instrument Science Report, 2016-08, available at <http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2016-08.pdf>
- [31] Bushouse, H., 2005, *CSM Particulate Investigation*, WFC3 Instrument Science Report, 2005-26, available at <http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2005-26.pdf>

APPENDIX A

ACSQL CODE

Table of Contents:

LICENSE	27
README.md	27
MANIFEST.in	28
setup.py	30
docs/Makefile	30
docs/requirements.txt	30
docs/source/conf.py	31
docs/source/database.rst	33
docs/source/index.rst	33
docs/source/ingest.rst	34
docs/source/scripts.rst	34
docs/source/utils.rst	34
docs/source/website.rst	34
acsq1/database/database_interface.py	35
acsq1/database/make_tabledefs.py	40
acsq1/database/queries.py	42
acsq1/database/reset_database.py	46
acsq1/database/update_tabledefs.py	46
acsq1/ingest/ingest.py	49
acsq1/ingest/make_file_dict.py	53
acsq1/ingest/make_jpeg.py	55
acsq1/ingest/make_thumbnail.py	56
acsq1/scripts/ingest_production.py	57
acsq1/utils/utils.py	60
acsq1/website/acsq1_webapp.py	63
acsq1/website/data_containers.py	66
acsq1/website/form_options.py	71
acsq1/website/query_form.py	72
acsq1/website/query_lib.py	76
acsq1/website/templates/404.html	81
acsq1/website/templates/500.html	81
acsq1/website/templates/_formhelpers.html	81
acsq1/website/templates/archive.html	82
acsq1/website/templates/base.html	82
acsq1/website/templates/database.html	83
acsq1/website/templates/database_error.html	84
acsq1/website/templates/database_table.html	84
acsq1/website/templates/header.html	86
acsq1/website/templates/index.html	86
acsq1/website/templates/view_image.html	86
acsq1/website/templates/view_proposal.html	88
acsq1/website/templates/view_query_results.html	89

LICENSE

```

1 Copyright (c) 2017, AURA
2 All rights reserved.
3
4 Redistribution and use in source and binary forms, with or without
5 modification, are permitted provided that the following conditions are met:
6
7 * Redistributions of source code must retain the above copyright notice, this
8   list of conditions and the following disclaimer.
9
10 * Redistributions in binary form must reproduce the above copyright notice,
11   this list of conditions and the following disclaimer in the documentation
12   and/or other materials provided with the distribution.
13
14 * Neither the name of acsql nor the names of its
15   contributors may be used to endorse or promote products derived from
16   this software without specific prior written permission.
17
18 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20 IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
21 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
22 FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
23 DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
24 SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
25 CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
26 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
27 OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

README.md

```

1 # acsql
2 The Advanced Camera for Surveys Quicklook Project
3
4 Official Documentation: http://acsql.readthedocs.io/
5
6 ## Installation
7
8 #### Dependencies
9
10 The `acsql` package requires the following dependencies:
11
12 - `python 3.+`
13 - `astropy`
14 - `flask`
15 - `numpy`
16 - `numpydoc`
17 - `pymysql`
18 - `requests`
19 - `sqlalchemy`
20 - `wtforms`
21 - `wtforms_components`
22 - `stak` (https://github.com/spacetelescope/stak)
23
24 #### Installing the `acsql` Package
25
26 The `acsql` application can be installed by cloning this repository and running the `setup.py` script:
27
28 ```
29 git clone https://github.com/spacetelescope/acsql.git
30 cd acsql
31 python setup.py install
32 ```
33
34 Users must fill in the `config.yaml` configuration file, located at `acsql/utils/config.yaml`:
35
36 ```yaml
37 connection_string : 'mysql+pymysql://username:password@host:12345/database'
38 filesystem : ''
39 log_dir : ''
40 jpeg_dir : ''
41 thumbnail_dir : ''
42 ncores : 1
43 ```
44
45 The `connection_string` item should contain the users credentials to the `acsql` database. Please ask [@bourque] (https://github.com/bourque) to set up an account.
46
```

```

47 The 'filesystem' item should point to the directory that holds ACS FITS files in a directory structure of
48 '<proposal_id>/<rootname>/<rootname_<filetype>.fits', for example:
49
50   ```
51   filesystem : '/Users/bourque/filesystem/'
52   ```
53
54 where '/Users/bourque/filesystem/' contains subdirectories such as:
55
56   ```
57   /users/bourque/filesystem/jbm1/
58   /users/bourque/filesystem/jbm1/jbm110u2q/
59   /users/bourque/filesystem/jbm1/jbm110u0q/
60   /users/bourque/filesystem/jbm1/jbm110tyq/
61   /users/bourque/filesystem/jbm1/jbm110txq/
62   /users/bourque/filesystem/jbm1/jbm110010/
63   /users/bourque/filesystem/jbm1/jbm105y7q/
64   /users/bourque/filesystem/jbm1/jbm105y5q/
65   /users/bourque/filesystem/jbm1/jbm105y3q/
66   /users/bourque/filesystem/jbm1/jbm105y2q/
67   /users/bourque/filesystem/jbm1/jbm105010/
68   ```
69
70 and each one of these subdirectories contains the data files, for example:
71
72   ```
73 >>> ls /users/bourque/filesystem/jbm1/jbm110u2q/
74
75 jbm110u2q_raw.jpg
76 jbm110u2q_flt.jpg
77 jbm110u2q_trl.fits
78 jbm110u2q_flt_hlet.fits
79 jbm110u2q_spt.fits
80 jbm110u2q_flt.fits
81 jbm110u2q_flc.fits
82 jbm110u2q_raw.fits
83   ```
84
85 In production, this 'filesystem' item points to the directory of the MAST Online Cache.
86
87 The 'log_dir' item should point to a directory in which log files that describe code execution will be
     written.
88
89 The 'jpeg_dir' item should point to a directory in which JPEGs will be written.
90
91 The 'thumbnail_dir' item should point to a directory in which smaller Thumbnail images will be written.
92
93 The 'ncores' item is set to the number of processors that should be used when performing data ingestion.
94
95 ##### Running the 'acsq1' web application locally:
96
97 Once the 'acsq1' package is installed, the 'acsq1' web application can be run locally:
98
99   ```
100 python acsq1/website/acsq1_webapp.py
101   ```
102
103 The just go to 'localhost:5000' in a browser.

```

MANIFEST.in

```

1 include LICENSE
2 include acsq1/utils/config.yaml
3 include acsq1/database/table_definitions/hrc_asn_0.txt
4 include acsq1/database/table_definitions/hrc_asn_1.txt
5 include acsq1/database/table_definitions/hrc_crj_0.txt
6 include acsq1/database/table_definitions/hrc_crj_1.txt
7 include acsq1/database/table_definitions/hrc_crj_2.txt
8 include acsq1/database/table_definitions/hrc_crj_3.txt
9 include acsq1/database/table_definitions/hrc_drz_0.txt
10 include acsq1/database/table_definitions/hrc_drz_1.txt
11 include acsq1/database/table_definitions/hrc_drz_2.txt
12 include acsq1/database/table_definitions/hrc_drz_3.txt
13 include acsq1/database/table_definitions/hrc_flt_0.txt
14 include acsq1/database/table_definitions/hrc_flt_1.txt
15 include acsq1/database/table_definitions/hrc_flt_2.txt
16 include acsq1/database/table_definitions/hrc_flt_3.txt
17 include acsq1/database/table_definitions/hrc_jif_0.txt

```



```

95 include acsql/database/table_definitions/wfc_jif_6.txt
96 include acsql/database/table_definitions/wfc_jit_0.txt
97 include acsql/database/table_definitions/wfc_jit_1.txt
98 include acsql/database/table_definitions/wfc_jit_2.txt
99 include acsql/database/table_definitions/wfc_jit_3.txt
100 include acsql/database/table_definitions/wfc_jit_4.txt
101 include acsql/database/table_definitions/wfc_jit_5.txt
102 include acsql/database/table_definitions/wfc_jit_6.txt
103 include acsql/database/table_definitions/wfc_raw_0.txt
104 include acsql/database/table_definitions/wfc_raw_1.txt
105 include acsql/database/table_definitions/wfc_raw_2.txt
106 include acsql/database/table_definitions/wfc_raw_3.txt
107 include acsql/database/table_definitions/wfc_raw_4.txt
108 include acsql/database/table_definitions/wfc_raw_5.txt
109 include acsql/database/table_definitions/wfc_raw_6.txt
110 include acsql/database/table_definitions/wfc_spt_0.txt
111 include acsql/database/table_definitions/wfc_spt_1.txt

```

setup.py

```

1 from setuptools import setup
2 from setuptools import find_packages
3
4 setup(
5     name = 'acsql',
6     description = 'The Advanced Camera for Surveys Quicklook Project',
7     url = 'https://github.com/spacetelescope/acsql.git',
8     author = 'Matthew Bourque, Sara Ogaz, Meredith Durbin, Alex Viana',
9     author_email = 'bourque@stsci.edu, ogaz@stsci.edu, mdurbin@uw.edu, alexcostaviana@gmail.com',
10    keywords = ['astronomy'],
11    classifiers = ['Programming Language :: Python'],
12    packages = find_packages(),
13    install_requires = [],
14    version = 0.0,
15    include_package_data=True
16 )

```

docs/Makefile

```

1 # Minimal makefile for Sphinx documentation
2 #
3
4 # You can set these variables from the command line.
5 SPHINXOPTS =
6 SPHINXBUILD = sphinx-build
7 SPHINXPROJ = acsql
8 SOURCEDIR = source
9 BUILDDIR = build
10
11 # Put it first so that "make" without argument is like "make help".
12 help:
13     @$(SPHINXBUILD) -M help "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)
14
15 .PHONY: help Makefile
16
17 # Catch-all target: route all unknown targets to Sphinx using the new
18 # "make mode" option. $(O) is meant as a shortcut for $(SPHINXOPTS).
19 %: Makefile
20     @$(SPHINXBUILD) -M $@ "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)

```

docs/requirements.txt

```

1 astropy
2 flask
3 numpy
4 numpydoc
5 pymysql
6 pyyaml
7 sqlalchemy
8 wtforms
9 wtforms_components
10 -e git://github.com/spacetelescope/stak.git#egg=stak

```

docs/source/conf.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 #
4 # acsql documentation build configuration file, created by
5 # sphinx-quickstart on Wed Feb 22 16:11:29 2017.
6 #
7 # This file is execfile()d with the current directory set to its
8 # containing dir.
9 #
10 # Note that not all possible configuration values are present in this
11 # autogenerated file.
12 #
13 # All configuration values have a default; values that are commented out
14 # serve to show the default.
15 #
16 # If extensions (or modules to document with autodoc) are in another directory,
17 # add these directories to sys.path here. If the directory is relative to the
18 # documentation root, use os.path.abspath to make it absolute, like shown here.
19 #
20 import os
21 import sys
22 sys.path.insert(0, os.path.abspath('../..../acsql'))
23
24
25 # -- General configuration -----
26
27 # If your documentation needs a minimal Sphinx version, state it here.
28 #
29 # needs_sphinx = '1.0'
30
31 # Add any Sphinx extension module names here, as strings. They can be
32 # extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
33 # ones.
34 extensions = ['sphinx.ext.autodoc', 'numpydoc']
35
36 # Add any paths that contain templates here, relative to this directory.
37 # templates_path = ['_templates']
38
39 # The suffix(es) of source filenames.
40 # You can specify multiple suffix as a list of string:
41 #
42 # source_suffix = ['.rst', '.md']
43 source_suffix = '.rst'
44
45 # The master toctree document.
46 master_doc = 'index'
47
48 # General information about the project.
49 project = 'acsql'
50 copyright = '2017, Matthew Bourque'
51 author = 'Matthew Bourque'
52
53 # The version info for the project you're documenting, acts as replacement for
54 # |version| and |release|, also used in various other places throughout the
55 # built documents.
56 #
57 # The short X.Y version.
58 version = '0.0'
59 # The full version, including alpha/beta/rc tags.
60 release = '0.0'
61
62 # The language for content autogenerated by Sphinx. Refer to documentation
63 # for a list of supported languages.
64 #
65 # This is also used if you do content translation via gettext catalogs.
66 # Usually you set "language" from the command line for these cases.
67 language = None
68
69 # List of patterns, relative to source directory, that match files and
70 # directories to ignore when looking for source files.
71 # This patterns also effect to html_static_path and html_extra_path
72 exclude_patterns = []
73
74 # The name of the Pygments (syntax highlighting) style to use.
75 pygments_style = 'sphinx'
76

```

```

77 # If true, 'todo' and 'todoList' produce output, else they produce nothing.
78 todo_include.todos = False
79
80
81 # -- Options for HTML output -----
82
83 # The theme to use for HTML and HTML Help pages. See the documentation for
84 # a list of builtin themes.
85 #
86 html_theme = 'default'
87
88 # Theme options are theme-specific and customize the look and feel of a theme
89 # further. For a list of options available for each theme, see the
90 # documentation.
91 #
92 # html_theme_options = {}
93
94 # Add any paths that contain custom static files (such as style sheets) here,
95 # relative to this directory. They are copied after the builtin static files,
96 # so a file named "default.css" will overwrite the builtin "default.css".
97 # html_static_path = ['_static']
98
99
100 # -- Options for HTMLHelp output -----
101
102 # Output file base name for HTML help builder.
103 htmlhelp_basename = 'acsqldoc'
104
105
106 # -- Options for LaTeX output -----
107
108 latex_elements = {
109     # The paper size ('letterpaper' or 'a4paper').
110     #
111     # 'papersize': 'letterpaper',
112
113     # The font size ('10pt', '11pt' or '12pt').
114     #
115     # 'pointsize': '10pt',
116
117     # Additional stuff for the LaTeX preamble.
118     #
119     # 'preamble': '',
120
121     # Latex figure (float) alignment
122     #
123     # 'figure_align': 'htbp',
124 }
125
126 # Grouping the document tree into LaTeX files. List of tuples
127 # (source start file, target name, title,
128 # author, documentclass [howto, manual, or own class]).
129 latex_documents = [
130     (master_doc, 'acsqld.tex', 'acsqld Documentation',
131      'Matthew Bourque', 'manual'),
132 ]
133
134
135 # -- Options for manual page output -----
136
137 # One entry per manual page. List of tuples
138 # (source start file, name, description, authors, manual section).
139 man_pages = [
140     (master_doc, 'acsqld', 'acsqld Documentation',
141      [author], 1)
142 ]
143
144
145 # -- Options for Texinfo output -----
146
147 # Grouping the document tree into Texinfo files. List of tuples
148 # (source start file, target name, title, author,
149 # dir menu entry, description, category)
150 texinfo_documents = [
151     (master_doc, 'acsqld', 'acsqld Documentation',
152      author, 'acsqld', 'One line description of project.',
153      'Miscellaneous'),]

```

docs/source/database.rst

```

1 =====
2 Database
3 =====
4
5 The ``database`` subpackage contains various modules for constructing and interacting with the ``acsqll``
Database.
6
7
8 database_interface
9 -----
10 .. automodule:: database.database_interface
11     :members: define_columns, get_special_column, load_connection, orm_factory
12     :undoc-members:
13     :show-inheritance:
14
15 make_tabledefs
16 -----
17 .. automodule:: database.make_tabledefs
18     :members:
19     :undoc-members:
20     :show-inheritance:
21
22 queries
23 -----
24 .. automodule:: database.queries
25     :members:
26     :undoc-members:
27     :show-inheritance:
28
29 reset_database
30 -----
31 .. automodule:: database.reset_database
32     :members:
33     :undoc-members:
34     :show-inheritance:
35
36 update_tabledefs
37 -----
38 .. automodule:: database.update_tabledefs
39     :members:
40     :undoc-members:
41     :show-inheritance:

```

docs/source/index.rst

```

1 .. acsql documentation master file, created by
2 sphinx-quickstart on Wed Feb 22 16:11:29 2017.
3 You can adapt this file completely to your liking, but it should at least
4 contain the root `toctree` directive.
5
6 The Hubble Space Telescope (HST) Advanced Camera for Surveys (ACS) Quicklook Project (ascql)
7 =====
8
9 The ``acsqll`` project is a database and web application that serves Hubble Space Telescope (HST)
10 Advanced Camera for Surveys (ACS) data for easy access and quick-viewing.
11
12
13 .. toctree::
14     :maxdepth: 2
15
16     database.rst
17     ingest.rst
18     utils.rst
19     website.rst
20
21
22 Indices and tables
23 =====
24
25 * :ref:`genindex`
26 * :ref:`modindex`
27 * :ref:`search`

```

docs/source/ingest.rst

```

1 =====
2 Ingest
3 =====
4
5 The ``ingest`` subpackage provides various modules to support the ingestion of files into the ``acsqll``
       database
6 and filesystem, such as creation of JPEGs & thumbnails and inserting or updating records in the database.
7
8 ingest
9 -----
10 .. automodule:: ingest.ingest
11     :members:
12     :undoc-members:
13     :show-inheritance:
14
15 make_file_dict
16 -----
17 .. automodule:: ingest.make_file_dict
18     :members:
19     :undoc-members:
20     :show-inheritance:
21
22 make_jpeg
23 -----
24 .. automodule:: ingest.make_jpeg
25     :members:
26     :undoc-members:
27     :show-inheritance:
28
29 make_thumbnail
30 -----
31 .. automodule:: ingest.make_thumbnail
32     :members:
33     :undoc-members:
34     :show-inheritance:

```

docs/source/scripts.rst

```

1 =====
2 Scripts
3 =====
4
5 ingest_production
6 -----
7 .. automodule:: scripts.ingest_production.py
8     :members:
9     :undoc-members:
10    :show-inheritance:

```

docs/source/utils.rst

```

1 =====
2 Utils
3 =====
4
5 The ``utils`` subpackage provides various various utility functions and objects that aid the other
6 subpackages, such as configuration settings, logging functions, and database convenience functions.
7
8 utils
9 -----
10 .. automodule:: utils.utils
11     :members:
12     :undoc-members:
13     :show-inheritance:

```

docs/source/website.rst

```

1 =====
2 Website
3 =====
4
5 The ``website`` subpackage provides various modules to support the ``acsqll`` web application.
6
7 acsqll_webapp
8 -----
9 .. automodule:: website.acsqll_webapp
10    :members:

```

```

11 :undoc-members:
12 :show-inheritance:
13
14 data_containers
15 -----
16 .. automodule:: website.data_containers
17   :members:
18   :undoc-members:
19   :show-inheritance:
20
21 form_options
22 -----
23 .. automodule:: website.form_options
24   :members:
25   :undoc-members:
26   :show-inheritance:
27
28 query_form
29 -----
30 .. automodule:: website.query_form
31   :members:
32   :undoc-members:
33   :show-inheritance:
34
35 query_lib
36 -----
37 .. automodule:: website.query_lib
38   :members:
39   :undoc-members:
40   :show-inheritance:
```

acsq1/database/database_interface.py

```

1 """This module provides ORMs for the ``acsq1`` database, as well as
2 ``engine`` and ``session`` objects for connecting to the database.
3
4 The ``load_connection()`` function within this module allows the user
5 to connect to the ``acsq1`` database via the ``session``, ``base``,
6 and ``engine`` objects (described below). The classes within serve as
7 ORMs (Object-relational mappings) that define the individual tables of
8 the relational database.
9
10 The ``engine`` object serves as the low-level database API and perhaps
11 most importantly contains dialects which allows the ``sqlalchemy``
12 module to communicate with the database.
13
14 The ``base`` object serves as a base class for class definitions. It
15 produces ``Table`` objects and constructs ORMs.
16
17 The ``session`` object manages operations on ORM-mapped objects, as
18 construed by the base. These operations include querying, for
19 example.
20
21 Authors
22 -----
23   Matthew Bourque
24
25 Use
26 ---
27   This module is intended to be imported from various ``acsq1``
28   modules and scripts. The importable objects from this module are
29   as follows:
30   ::

31       from acsq1.database.database_interface import base
32       from acsq1.database.database_interface import engine
33       from acsq1.database.database_interface import session
34       from acsq1.database.database_interface import Master
35       from acsq1.database.database_interface import Datasets
36       from acsq1.database.database_interface import <header_table>
37
38 Dependencies
39 -----
40   External library dependencies include:
41
42   - ``acsq1``
43   - ``pymysql``
44   - ``sqlalchemy``
```

```

46 """
47
48 import os
49
50 from sqlalchemy import Boolean
51 from sqlalchemy import Column
52 from sqlalchemy import create_engine
53 from sqlalchemy import Date
54 from sqlalchemy import DateTime
55 from sqlalchemy import Index
56 from sqlalchemy import Enum
57 from sqlalchemy import ForeignKey
58 from sqlalchemy import ForeignKeyConstraint
59 from sqlalchemy import Integer
60 from sqlalchemy.orm import sessionmaker
61 from sqlalchemy import String
62 from sqlalchemy import Time
63 from sqlalchemy.ext.declarative import declarative_base
64 from sqlalchemy.types import Float
65
66 from acsql.utils.utils import SETTINGS
67
68
69 def define_columns(data_dict, class_name):
70     """Dynamically define the class attributes for the ORM
71
72     Parameters
73     -----
74     data_dict : dict
75         A dictionary containing the ORM definitions
76     class_name : str
77         The name of the class/ORM.
78
79     Returns
80     -----
81     data_dict : dict
82         A dictionary containing the ORM definitions, now with header
83         definitions added.
84     """
85
86     special_keywords = ['RULEFILE', 'FWERROR', 'FW2ERROR', 'PROPTTL1',
87                         'TARDESCR', 'QUALCOM2']
88
89     with open(os.path.join(os.path.split(__file__)[0], 'table_definitions',
90                           class_name.lower() + '.txt'), 'r') as f:
91         data = f.readlines()
92     keywords = [item.strip().split(',') for item in data]
93     for keyword in keywords:
94         if keyword[0] in special_keywords:
95             data_dict[keyword[0].lower()] = get_special_column(keyword[0])
96         elif keyword[1] == 'Integer':
97             data_dict[keyword[0].lower()] = Column(Integer())
98         elif keyword[1] == 'String':
99             data_dict[keyword[0].lower()] = Column(String(50))
100        elif keyword[1] == 'Float':
101            data_dict[keyword[0].lower()] = Column(Float(precision=32))
102        elif keyword[1] == 'Decimal':
103            data_dict[keyword[0].lower()] = Column(Float(precision='13,8'))
104        elif keyword[1] == 'Date':
105            data_dict[keyword[0].lower()] = Column(Date())
106        elif keyword[1] == 'Time':
107            data_dict[keyword[0].lower()] = Column(Time())
108        elif keyword[1] == 'DateTime':
109            data_dict[keyword[0].lower()] = Column(DateTime())
110        elif keyword[1] == 'Bool':
111            data_dict[keyword[0].lower()] = Column(Boolean())
112        else:
113            raise ValueError('unrecognized header keyword type: {}:{}' .format(
114                keyword[0], keyword[1]))
115
116        if 'aperture' in data_dict:
117            data_dict['aperture'] = Column(String(50), index=True)
118
119    return data_dict
120
121 def get_special_column(keyword):

```



```

200     nullable=False)
201 data_dict['filename'] = Column(String(18), nullable=False, unique=True)
202 data_dict = define_columns(data_dict, class_name)
203 data_dict['__table_args__'] = {'mysql_row_format': 'DYNAMIC'}
204
205     return type(class_name.upper(), (base,), data_dict)
206
207
208 class Master(base):
209     """ORM for the master table."""
210     def __init__(self, data_dict):
211         self.__dict__.update(data_dict)
212
213     __tablename__ = 'master'
214     rootname = Column(String(8), primary_key=True, index=True, nullable=False)
215     path = Column(String(15), unique=True, nullable=False)
216     first_ingest_date = Column(Date, nullable=False)
217     last_ingest_date = Column(Date, nullable=False)
218     detector = Column(Enum('WFC', 'HRC', 'SBC'), nullable=False)
219     proposal_type = Column(Enum('CAL/ACS', 'CAL/OTA', 'CAL/STIS', 'CAL/WFC3',
220                               'ENG/ACS', 'GO', 'GO/DD', 'GO/PAR', 'GTO/ACS',
221                               'GTO/COS', 'NASA', 'SM3/ACS', 'SM3/ERO',
222                               'SM4/ACS', 'SM4/COS', 'SM4/ERO', 'SNAP'),
223                               nullable=True)
224
225
226 class Datasets(base):
227     """ORM for the datasets table."""
228     def __init__(self, data_dict):
229         self.__dict__.update(data_dict)
230
231     __tablename__ = 'datasets'
232     rootname = Column(String(8), ForeignKey('master.rootname'),
233                       primary_key=True, index=True, nullable=False)
234     raw = Column(String(18), nullable=True)
235     flt = Column(String(18), nullable=True)
236     flc = Column(String(18), nullable=True)
237     spt = Column(String(18), nullable=True)
238     drz = Column(String(18), nullable=True)
239     drc = Column(String(18), nullable=True)
240     crj = Column(String(18), nullable=True)
241     crc = Column(String(18), nullable=True)
242     jif = Column(String(18), nullable=True)
243     jit = Column(String(18), nullable=True)
244     asn = Column(String(18), nullable=True)
245
246     # foreign_keys = []
247     # for filetype in FILE_EXTS:
248     #     for ext in [0, 1]:
249     #         foreign_keys.append(ForeignKeyConstraint([filetype],
250 #                                         ['wfc_{}_{}.filename'.format(filetype, ext)]))
251     # foreign_keys = tuple(foreign_keys)
252     # __table_args__ = foreign_keys
253
254
255 # WFC tables
256 WFC_raw_0 = orm_factory('WFC_raw_0')
257 WFC_raw_1 = orm_factory('WFC_raw_1')
258 WFC_raw_2 = orm_factory('WFC_raw_2')
259 WFC_raw_3 = orm_factory('WFC_raw_3')
260 WFC_raw_4 = orm_factory('WFC_raw_4')
261 WFC_raw_5 = orm_factory('WFC_raw_5')
262 WFC_raw_6 = orm_factory('WFC_raw_6')
263
264 WFC_flt_0 = orm_factory('WFC_flt_0')
265 WFC_flt_1 = orm_factory('WFC_flt_1')
266 WFC_flt_2 = orm_factory('WFC_flt_2')
267 WFC_flt_3 = orm_factory('WFC_flt_3')
268 WFC_flt_4 = orm_factory('WFC_flt_4')
269 WFC_flt_5 = orm_factory('WFC_flt_5')
270 WFC_flt_6 = orm_factory('WFC_flt_6')
271
272 WFC_flc_0 = orm_factory('WFC_flc_0')
273 WFC_flc_1 = orm_factory('WFC_flc_1')
274 WFC_flc_2 = orm_factory('WFC_flc_2')
275 WFC_flc_3 = orm_factory('WFC_flc_3')
276 WFC_flc_4 = orm_factory('WFC_flc_4')

```

```

277 WFC_flc_5 = orm_factory('WFC_flc_5')
278 WFC_flc_6 = orm_factory('WFC_flc_6')
279
280 WFC_spt_0 = orm_factory('WFC_spt_0')
281 WFC_spt_1 = orm_factory('WFC_spt_1')
282
283 WFC_drz_0 = orm_factory('WFC_drz_0')
284 WFC_drz_1 = orm_factory('WFC_drz_1')
285 WFC_drz_2 = orm_factory('WFC_drz_2')
286 WFC_drz_3 = orm_factory('WFC_drz_3')
287
288 WFC_drc_0 = orm_factory('WFC_drc_0')
289 WFC_drc_1 = orm_factory('WFC_drc_1')
290 WFC_drc_2 = orm_factory('WFC_drc_2')
291 WFC_drc_3 = orm_factory('WFC_drc_3')
292
293 WFC_crj_0 = orm_factory('WFC_crj_0')
294 WFC_crj_1 = orm_factory('WFC_crj_1')
295 WFC_crj_2 = orm_factory('WFC_crj_2')
296 WFC_crj_3 = orm_factory('WFC_crj_3')
297 WFC_crj_4 = orm_factory('WFC_crj_4')
298 WFC_crj_5 = orm_factory('WFC_crj_5')
299 WFC_crj_6 = orm_factory('WFC_crj_6')
300
301 WFC_crc_0 = orm_factory('WFC_crc_0')
302 WFC_crc_1 = orm_factory('WFC_crc_1')
303 WFC_crc_2 = orm_factory('WFC_crc_2')
304 WFC_crc_3 = orm_factory('WFC_crc_3')
305 WFC_crc_4 = orm_factory('WFC_crc_4')
306 WFC_crc_5 = orm_factory('WFC_crc_5')
307 WFC_crc_6 = orm_factory('WFC_crc_6')
308
309 WFC_jif_0 = orm_factory('WFC_jif_0')
310 WFC_jif_1 = orm_factory('WFC_jif_1')
311 WFC_jif_2 = orm_factory('WFC_jif_2')
312 WFC_jif_3 = orm_factory('WFC_jif_3')
313 WFC_jif_4 = orm_factory('WFC_jif_4')
314 WFC_jif_5 = orm_factory('WFC_jif_5')
315 WFC_jif_6 = orm_factory('WFC_jif_6')
316
317 WFC_jit_0 = orm_factory('WFC_jit_0')
318 WFC_jit_1 = orm_factory('WFC_jit_1')
319 WFC_jit_2 = orm_factory('WFC_jit_2')
320 WFC_jit_3 = orm_factory('WFC_jit_3')
321 WFC_jit_4 = orm_factory('WFC_jit_4')
322 WFC_jit_5 = orm_factory('WFC_jit_5')
323 WFC_jit_6 = orm_factory('WFC_jit_6')
324
325 WFC_asn_0 = orm_factory('WFC_asn_0')
326 WFC_asn_1 = orm_factory('WFC_asn_1')
327
328
329 # HRC tables
330 HRC_raw_0 = orm_factory('HRC_raw_0')
331 HRC_raw_1 = orm_factory('HRC_raw_1')
332 HRC_raw_2 = orm_factory('HRC_raw_2')
333 HRC_raw_3 = orm_factory('HRC_raw_3')
334
335 HRC_flt_0 = orm_factory('HRC_flt_0')
336 HRC_flt_1 = orm_factory('HRC_flt_1')
337 HRC_flt_2 = orm_factory('HRC_flt_2')
338 HRC_flt_3 = orm_factory('HRC_flt_3')
339
340 HRC_spt_0 = orm_factory('HRC_spt_0')
341 HRC_spt_1 = orm_factory('HRC_spt_1')
342
343 HRC_drz_0 = orm_factory('HRC_drz_0')
344 HRC_drz_1 = orm_factory('HRC_drz_1')
345 HRC_drz_2 = orm_factory('HRC_drz_2')
346 HRC_drz_3 = orm_factory('HRC_drz_3')
347
348 HRC_crj_0 = orm_factory('HRC_crj_0')
349 HRC_crj_1 = orm_factory('HRC_crj_1')
350 HRC_crj_2 = orm_factory('HRC_crj_2')
351 HRC_crj_3 = orm_factory('HRC_crj_3')
352
353 HRC_jif_0 = orm_factory('HRC_jif_0')

```

```

354 HRC_jif_1 = orm_factory('HRC_jif_1')
355 HRC_jif_2 = orm_factory('HRC_jif_2')
356
357 HRC_jit_0 = orm_factory('HRC_jit_0')
358 HRC_jit_1 = orm_factory('HRC_jit_1')
359 HRC_jit_2 = orm_factory('HRC_jit_2')
360
361 HRC_asn_0 = orm_factory('HRC_asn_0')
362 HRC_asn_1 = orm_factory('HRC_asn_1')
363
364
365 # SBC tables
366 SBC_raw_0 = orm_factory('SBC_raw_0')
367 SBC_raw_1 = orm_factory('SBC_raw_1')
368 SBC_raw_2 = orm_factory('SBC_raw_2')
369 SBC_raw_3 = orm_factory('SBC_raw_3')
370
371 SBC_flt_0 = orm_factory('SBC_flt_0')
372 SBC_flt_1 = orm_factory('SBC_flt_1')
373 SBC_flt_2 = orm_factory('SBC_flt_2')
374 SBC_flt_3 = orm_factory('SBC_flt_3')
375
376 SBC_spt_0 = orm_factory('SBC_spt_0')
377 SBC_spt_1 = orm_factory('SBC_spt_1')
378
379 SBC_drz_0 = orm_factory('SBC_drz_0')
380 SBC_drz_1 = orm_factory('SBC_drz_1')
381 SBC_drz_2 = orm_factory('SBC_drz_2')
382 SBC_drz_3 = orm_factory('SBC_drz_3')
383
384 SBC_jif_0 = orm_factory('SBC_jif_0')
385 SBC_jif_1 = orm_factory('SBC_jif_1')
386 SBC_jif_2 = orm_factory('SBC_jif_2')
387
388 SBC_jit_0 = orm_factory('SBC_jit_0')
389 SBC_jit_1 = orm_factory('SBC_jit_1')
390 SBC_jit_2 = orm_factory('SBC_jit_2')
391
392 SBC_asn_0 = orm_factory('SBC_asn_0')
393 SBC_asn_1 = orm_factory('SBC_asn_1')
394
395
396 if __name__ == '__main__':
397
398     # Give user a second chance
399     prompt = ('About to reset all table(s) for database instance {}. Do you '
400               'wish to proceed? (y/n)\n'.format(SETTINGS['connection_string']))
401
402     response = input(prompt)
403
404     if response.lower() == 'y':
405         print('Resetting table(s)')
406         base.metadata.drop_all()
407         base.metadata.create_all()

```

acsq1/database/make_tabledefs.py

```

1  #! /usr/bin/env/ python
2
3 """Creates static text files that hold header keywords and keyword
4 data types for each ACS filetype (and each extension). Each text file
5 corresponds to a header table in the ``acsq1`` database.
6
7 Authors
8 -----
9   - Sara Ogaz
10   - Matthew Bourque
11
12 Use
13 ---
14   This module is intended to be run via the command line as such:
15   :::
16
17   python make_tabledefs.py
18
19 Dependencies
20 -----
21   External library dependencies include:

```

```

22     - ``acsqll``
23     - ``numpy``
24     - ``stak`` (``https://github.com/spacetelescope/stak``)
25
26
27 Notes
28 -----
29     The ``stak.Hselect`` dependency still depends on Python 2 at the
30     time of this writing.
31 """
32
33 import glob
34 import os
35
36 import numpy as np
37 #from stak import Hselect
38
39 from acsql.utils import utils
40 from acsql.utils import SETTINGS
41
42
43 def make_tabledefs(detector):
44 """
45     Function to auto-produce the table_definition files.
46
47     Note that due to how ``hselect`` handles ``ASN`` files, they must
48     be handled separately.
49
50 Parameters
51 -----
52 detector : str
53     The detector (e.g. ``wfc``).
54 """
55
56 file_exts = getattr(utils, '{}_FILE_EXTS'.format(detector.upper()))
57
58 for ftype in file_exts:
59
60     all_files = glob.glob('table_definitions/test_files/test_{}_*.{fits}'.format(detector, ftype))
61
62     for ext in file_exts[ftype]:
63
64         filename = 'table_definitions/{}_{}_{}.txt'.format(detector,
65                                               ftype, ext)
66         hsel = Hselect(all_files, '*', extension=(ext,))
67
68         print('Making file {}'.format(filename))
69         with open(filename, 'w') as f:
70             for col in hsel.table.itercols():
71                 column_name = col.name
72                 if column_name in ['ROOTNAME', 'Filename', 'FILENAME', 'Ext']:
73                     continue
74                 elif col.dtype in [np.dtype('S68'), np.dtype('S80')]:
75                     ptype = 'String'
76                 elif col.dtype in [np.int64]:
77                     ptype = 'Integer'
78                 elif col.dtype in [bool]:
79                     ptype = 'Bool'
80                 elif col.dtype in [np.float64]:
81                     ptype = 'Float'
82                 else:
83                     print('Could not find type match: {}:{}'.
84                           format(column_name, col.dtype))
85
86                 # If the column has a hyphen, switch it to underscore
87                 if '-' in column_name:
88                     column_name = column_name.replace('-', '_')
89
90                 f.write('{}, {}\n'.format(column_name, ptype))
91
92
93 if __name__ == '__main__':
94     make_tabledefs('wfc')
95     make_tabledefs('sbc')
96     make_tabledefs('hrc')

```

acsq1/database/queries.py

```

1 #! /usr/bin/env python
2
3 """Contains various functions to perform useful queries of the
4 ``acsq1`` database.
5
6 The available queries are:
7
8 1. ``all_filenames``
9 2. ``filters_for_rootname(rootname)``
10 3. ``filter_distribution()``
11 4. ``rootnames_for_target(targname)``
12 5. ``filenames_for_calibration(calibration_keyword, value)``
13 6. ``goodmean_for_dataset(dataset)``
14 7. ``rootnames_with_postflash()``
15 8. ``non_asn_rootnames()``
16 9. ``filenames_in_date_range()``
17
18 See each function's docstrings for further details.
19
20 Each function returns the ``sqlalchemy.query`` object for the query
21 performed so that the user may perform the query themselves and
22 perform additional operations with the query and/or its results.
23
24 Authors
25 -----
26 - Sara Ogaz
27 - Matthew Bourque
28
29 Use
30 ---
31 This script is intended to be imported as such:
32 ::

33
34     from acsq1.database import queries
35
36 ``queries`` can then be used to perform individual queries, e.g.:
37 ::

38     query = queries.filter_distribution()
39
40 Each function will print the results to the screen, but the user
41 may also perform the query and handle the results themselves, e.g.:
42 ::

43     results = query.all()
44
45 Dependencies
46 -----
47 External library dependencies include:
48
49 - ``acsq1``
50 - ``sqlalchemy``
51
52 """
53
54
55 from sqlalchemy import and_
56 from sqlalchemy import exists
57 from sqlalchemy import func
58
59 from acsq1.database.database_interface import session
60 from acsq1.database.database_interface import Master
61 from acsq1.database.database_interface import Datasets
62 from acsq1.database.database_interface import WFC_asn_0
63 from acsq1.database.database_interface import WFC_raw_0
64 from acsq1.database.database_interface import WFC_flt_1
65 from acsq1.database.database_interface import WFC_flt_4
66
67
68 def all_filenames(dataset):
69     """Queries for all filenames that exist for the given dataset.
70
71 Parameters
72 -----
73 dataset : str
74     Any portion of (or entire) rootname (e.g. 'jd2615qi', or
75     'jd2615').
```

```

77     Returns
78     -----
79     query : obj
80         The query object that contains attributes and methods for
81         performing the query.
82     """
83
84     query = session.query(Datasets) \
85         .filter(Datasets.rootname.like('{}%'.format(dataset)))
86     query_results = query.all()
87
88     print('\nQuery performed:\n\n{}\n'.format(str(query)))
89
90     for result in query_results:
91         results_dict = result.__dict__
92         del results_dict['_sa_instance_state']
93         print(results_dict)
94
95     return query
96
97
98 def filters_for_rootname(rootname):
99     """Queries for the FILTER1/FILTER2 combination for the given
100    observation.
101
102    Parameters
103    -----
104    rootname : str
105        The rootname to query by.
106
107    Returns
108    -----
109    query : obj
110        The query object that contains attributes and methods for
111        performing the query.
112    """
113
114     query = session.query(WFC_raw_0.filter1, WFC_raw_0.filter2) \
115         .filter(WFC_raw_0.rootname == rootname)
116     query_results = query.one()
117
118     print('\nQuery performed:\n\n{}\n'.format(str(query)))
119     print('{}: {}'.format(rootname, query_results))
120
121     return query
122
123
124 def filter_distribution():
125     """Queries for the FILTER1/FILTER2 combination for the given
126    observation.
127
128    Parameters
129    -----
130    rootname : str
131        The rootname to query by.
132
133    Returns
134    -----
135    query : obj
136        The query object that contains attributes and methods for
137        performing the query.
138    """
139
140     query = session.query(WFC_raw_0.filter1, WFC_raw_0.filter2,
141         func.count(WFC_raw_0.filter1)) \
142             .group_by(WFC_raw_0.filter1, WFC_raw_0.filter2)
143     query_results = query.all()
144     db_count = session.query(WFC_raw_0).count()
145
146     print('\nQuery performed:\n\n{}\n'.format(str(query)))
147
148     for result in query_results:
149         perc_used = round((result[2] / db_count) * 100., 2)
150         print('\t{}: {}%'.format(result[0], result[1], perc_used))
151
152     return query
153

```

```

154
155 def rootnames_for_target(targname):
156     """Queries for the rootname and filename for a given target.
157
158     Parameters
159     -----
160     targname : str
161         The target name (e.g. 'NGC104')
162
163     Returns
164     -----
165     query : obj
166         The query object that contains attributes and methods for
167         performing the query.
168
169
170     query = session.query(WFC_raw_0.rootname, WFC_raw_0.filename,
171         WFC_raw_0.targname)\ \
172             .filter(WFC_raw_0.targname == targname)
173     query_results = query.all()
174
175     print('\nQuery performed:\n\n{}\n'.format(str(query)))
176
177     for result in query_results:
178         print(result)
179
180     return query
181
182
183 def filenames_for_calibration(calibration_keyword, value):
184     """Queries for the filenames that used a given calibration mode.
185
186     The 'calibration' mode is defined by the type of calibration and
187     the calibration reference file used
188     (e.g. 'BIASFILE = jref$06u15056j_bia.fits')
189
190     Parameters
191     -----
192     calibration_keyword : str
193         The calibration file to query on (e.g. DARKFILE, BIASFILE)
194     value : str
195         The calibration file value (e.g. jref$06u15056j_bia.fits)
196
197     Returns
198     -----
199     query : obj
200         The query object that contains attributes and methods for
201         performing the query.
202
203
204     calibration_keyword_obj = getattr(WFC_raw_0, calibration_keyword)
205     query = session.query(WFC_raw_0.filename)\ \
206         .filter(calibration_keyword_obj == value)
207     query_results = query.all()
208
209     print('\nQuery performed:\n\n{}\n'.format(str(query)))
210
211     for result in query_results:
212         print(result[0])
213
214     return query
215
216
217 def goodmean_for_dataset(dataset):
218     """Queries for the GOODMEAN values for a given dataset
219
220     The GOODMEAN describes the mean of all 'good' (i.e. non-flagged)
221     pixels in the image.
222
223     Parameters
224     -----
225     dataset : str
226         Any portion of (or entire) rootname (e.g. 'jd2615qi', or
227         'jd2615').
228
229     Returns
230     -----

```

```

231     query : obj
232         The query object that contains attributes and methods for
233             performing the query.
234     """
235
236     query = session.query(Master.rootname, WFC_flt_1.goodmean,
237         WFC_flt_4.goodmean) \
238             .join(WFC_flt_1) \
239             .join(WFC_flt_4) \
240             .filter(Master.rootname.like('{}%'.format(dataset)))
241     query_results = query.all()
242
243     print('\nQuery performed:\n\n{}\n'.format(str(query)))
244
245     for result in query_results:
246         print(result)
247
248     return query
249
250
251 def rootnames_with_postflash():
252     """Queries for rootnames and FLASHDURs for non-DARK observations
253     that have a FLASHDUR > 0.
254
255     Returns
256     -----
257     query : obj
258         The query object that contains attributes and methods for
259             performing the query.
260     """
261
262     query = session.query(Master.rootname, WFC_raw_0.flashdur) \
263         .join(WFC_raw_0) \
264         .filter(WFC_raw_0.flashdur > 0) \
265         .filter(WFC_raw_0.targname != 'DARK')
266
267     query_results = query.all()
268
269     print('\nQuery performed:\n\n{}\n'.format(str(query)))
270
271     for result in query_results:
272         print(result)
273
274     return query
275
276
277 def non_asn_rootnames():
278     """Queries for rootnames that are not part of an association.
279
280     Returns
281     -----
282     query : obj
283         The query object that contains attributes and methods for
284             performing the query.
285     """
286
287     query = session.query(Master.rootname) \
288         .filter(exists().where(and_(Master.rootname == WFC_asn_0.rootname)))
289     query_results = query.all()
290
291     print('\nQuery performed:\n\n{}\n'.format(str(query)))
292
293     for result in query_results:
294         print(result[0])
295
296     return query
297
298
299 def filenames_in_date_range(begin_date, end_date):
300     """Queries for filenames for observations that occur between the
301     ``begin_date`` and ``end_date``.
302
303     Parameters
304     -----
305     begin_date : str
306         The start of the date range (in the format YYYY-MM-DD).
307     end_date : str

```

```

308     The end of the date range (in the format YYYY-MM-DD).
309
310     Returns
311     -----
312     query : obj
313         The query object that contains attributes and methods for
314         performing the query.
315     """
316
317     query = session.query(WFC_raw_0.filename) \
318         .filter(WFC_raw_0.date_obs >= begin_date) \
319         .filter(WFC_raw_0.date_obs <= end_date)
320     query_results = query.all()
321
322     print('\nQuery performed:\n\n{}\n'.format(str(query)))
323
324     for result in query_results:
325         print(result[0])
326
327     return query

```

acsq1/database/reset_database.py

```

1 #! /usr/bin/env python
2
3 """Reset all tables in the ``acsq1`` database.
4
5 Authors
6 -----
7     Matthew Bourque, 2017
8
9 Use
10 ---
11     This script is intended to be used in the command line:
12     :::
13
14     python reset_database.py
15
16 Dependencies
17 -----
18     External library dependencies include:
19
20     - ``acsq1``
21 """
22
23 from acsq1.database.database_interface import base
24 from acsq1.utils.utils import SETTINGS
25
26
27 if __name__ == '__main__':
28
29     prompt = ('About to reset all tables for database instance {}. Do you '
30             'wish to proceed? (y/n)\n'.format(SETTINGS['connection_string']))
31     response = input(prompt)
32
33     if response.lower() == 'y':
34         print('Resetting database.')
35         base.metadata.drop_all()
36         base.metadata.create_all()

```

acsq1/database/update_tabledefs.py

```

1 #! /usr/bin/env python
2
3 """Updates the ``table_definitions`` text files that store header
4 keyword/data type pairs (see ``make_tabledefs.py`` module documentation
5 for further details).
6
7 A given ``logfile`` is used to scrape for logging ``WARNINGS`` that
8 warn of missing ``acsq1`` database columns (i.e. header keywords that
9 exist in the file header but does not exist as a database column in the
10 appropriate header table). The new header keywords are appended to the
11 appropriate ``table_definitions`` text file based on the ``detector``,
12 ``filetype`` and ``extension`` (e.g. ``WFC_raw_0.txt``).
13
14 Additionally, the ``ALTER TABLE`` commands needed to add the
15 corresponding columns are printed to the standard output. ``acsq1``
16 users can then use these commands within ``MySQL`` to add the
17 appropriate columns.

```

```

18
19 Authors
20 -----
21     Matthew Bourque
22
23 Use
24 ---
25     This module is intended to be used when an ``acsqll`` user sees fit
26     to add any new header keywords to the database. The module can
27     be used from the command line as such:
28     ::

29
30     python update_tabledefs.py <logfile>
31
32 Required arguments:
33 ::

34
35     logfile: The path to an ``acsqll.ingest.ingest.py`` log to be
36     used to determine new header keywords.
37
38 Dependencies
39 -----
40     External library dependencies include:
41
42     - ``acsqll``
43     - ``numpy``
44     - ``stak`` ('https://github.com/spacetelescope/stak'')
45
46 Notes
47 -----
48     The ``stak.Hselect`` dependency still depends on Python 2 at the
49     time of this writing.
50 """
51
52 import argparse
53 import os
54
55 import numpy as np
56 #from stak import Hselect
57
58 from acsqll.utils.utils import SETTINGS
59
60
61 def parse_args():
62     """Parse command line arguments. Returns ``args`` object
63
64     Returns
65     -----
66     args : obj
67         An argparse object containing all of the arguments
68     """
69
70     # Create help strings
71     logfile_help = 'The path to the logfile to use to determine new header '
72     logfile_help += 'keywords.'
73
74     # Add arguments
75     parser = argparse.ArgumentParser()
76     parser.add_argument('logfile',
77                         action='store',
78                         help=logfile_help)
79
80     # Parse args
81     args = parser.parse_args()
82
83     # Ensure that the logfile exists
84     assert os.path.exists(args.logfile), '{} does not exist.' \
85     .format(args.logfile)
86
87     return args
88
89
90 def update_tabledefs(logfile):
91     """The main function of the ``update_tabledefs`` module. See
92     module documentation for further details.
93
94     Parameters

```

```

95 -----
96     logfile : str
97         The path to an ``acsq1.ingest.ingest.py`` log to be used to
98         determine new header keywords.
99 """
100
101 # Read in the logfile
102 with open(logfile) as f:
103     data = f.readlines()
104
105 # Grab out the appropriate WARNING lines
106 warnings = [item.strip() for item in data if 'not in' in item]
107 warnings = [item.split(':') for item in warnings]
108
109 # Determine metadata for each warning
110 rootnames = [item[-2].strip() for item in warnings]
111 keywords = [item[-1].strip().split(' not in')[0] for item in warnings]
112 tables = [item[-1].strip().split('not in ')[-1] for item in warnings]
113 filetypes = [item.split('_')[1] for item in tables]
114 paths = [os.path.join(SETTINGS['filesystem'], rootname[0:4], rootname,
115                     '{}_{}.fits'.format(rootname, filetype)) for rootname, filetype
116                     in zip(rootnames, filetypes)]
117
118 # Take a set of the warnings to avoid duplications
119 files_to_test = ['{}, {}, {}'.format(path, keyword, table) for path,
120                  keyword, table in zip(paths, keywords, tables)]
121 keywords_to_add = set([(keyword, table) for keyword, table in zip(keywords,
122                           tables)])
123
124 # For each keyword to add, use test file to determine data type
125 command_list = []
126 for keyword, table in keywords_to_add:
127     test_file = [f for f in files_to_test if '{}, {}, {}'.format(keyword, table) in f][0]
128     test_file = test_file.split(',')[0]
129
130     # Use hselect to determine data type
131     try:
132         hsel = Hselect(test_file, keyword, extension=(int(table[-1]),))
133         dtype = hsel.table[keyword].dtype
134     except:
135         print('Cannot determine datatype for {}. Defaulting to String'
136               .format(keyword))
137         dtype = np.dtype('S80')
138
139     if dtype in [np.dtype('S68'), np.dtype('S80')]:
140         col_type = 'String'
141         db_type = 'VARCHAR(50)'
142     elif dtype in [np.int64]:
143         col_type = 'Integer'
144         db_type = 'INTEGER'
145     elif dtype in [bool]:
146         col_type = 'Bool'
147         db_type = 'BOOLEAN'
148     elif dtype in [np.float64]:
149         col_type = 'Float'
150         db_type = 'FLOAT'
151     else:
152         print('Could not find type match: {}:{}' .format(keyword, dtype))
153
154     # Update the appropriate table definitions file
155     tabledefs_file = 'table_definitions/{}.txt'.format(table.lower())
156     with open(tabledefs_file, 'r') as f:
157         data = f.readlines()
158     existing_keywords = [item.strip().split(',')[0] for item in data]
159     with open(tabledefs_file, 'a') as f:
160         if keyword not in existing_keywords:
161             f.write('{} , {} \n'.format(keyword, col_type))
162     print('Updated {} with {}' .format(tabledefs_file, keyword))
163
164     # Add ALTER TABLE command to list of commands
165     command = 'ALTER TABLE {} ADD {} {};' .format(table.lower(),
166                                                    keyword.lower(), db_type)
167     command_list.append(command)
168
169     # Print out the ALTER TABLE commands
170     print('\n\nALTER TABLE commands to execute for database:\n')
171

```

```

172     for command in command_list:
173         print(command)
174
175 if __name__ == '__main__':
176
177     args = parse_args()
178     update_tabledefs(args.logfile)

```

acsq1/ingest/ingest.py

```

1 """Ingests a given rootname (and its associated files) into the
2 ``ascq1`` database. The tables that are updated are the ``master``
3 table, the ``datasets`` table, and any appropriate header tables
4 (e.g. ``wfc_raw_0``) based on the available filetypes and header
5 extensions.
6
7 Authors
8 -----
9 - Matthew Bourque
10 - Sara Ogaz
11
12 Use
13 ---
14 This module is intended to be imported from and used by the
15 ``ingest_production`` script as such:
16 ::

17
18     from acsq1.ingest.ingest import ingest
19     ingest(rootname)
20
21 Dependencies
22 -----
23 External library dependencies include:
24
25 - ``acsq1``
26 - ``astropy``
27 - ``sqlalchemy``
28 ``````

29
30 from datetime import date
31 import glob
32 import logging
33 import os
34 import urllib.request
35
36 from astropy.io import fits
37 from astropy.io.fits.verify import VerifyError
38 from sqlalchemy import Table
39 from sqlalchemy.exc import IntegrityError
40
41 from acsq1.database.database_interface import Datasets
42 from acsq1.database.database_interface import load_connection
43 from acsq1.ingest.make_file_dict import get_metadata_from_test_files
44 from acsq1.ingest.make_file_dict import make_file_dict
45 from acsq1.ingest.make_jpeg import make_jpeg
46 from acsq1.ingest.make_thumbnail import make_thumbnail
47 from acsq1.utils.utils import insert_or_update
48 from acsq1.utils.utils import SETTINGS
49 from acsq1.utils.utils import TABLE_DEFS
50 from acsq1.utils.utils import VALID_FILETYPES
51 from acsq1.utils.utils import VALID_PROPOSAL_TYPES
52
53
54 def get_proposal_type(proposid):
55     """Return the ``proposal_type`` for the given ``proposid``.
56
57     The ``proposal_type`` is the type of proposal (e.g. ``CAL``,
58     ``GO``, etc.). The ``proposal_type`` is scraped from the MAST
59     proposal status webpage for the given ``proposid``. If the
60     ``proposal_type`` cannot be determined, a ``None`` value is returned.
61
62 Parameters
63 -----
64 proposid : str
65     The proposal ID (e.g. ``12345``).
66
67 Returns
68 -----

```

```

69     proposal_type : int or None
70         The proposal type (e.g. ``CAL``).
71     """
72
73     if not proposid:
74         proposal_type = None
75     else:
76         try:
77             url = 'http://www.stsci.edu/cgi-bin/get-proposal-info?id='
78             url += '{}&submit=Go&observatory=HST'.format(proposid)
79             webpage = urllib.request.urlopen(url)
80             proposal_type = webpage.readlines()[11].split(b'prop_type"')[-1]
81             proposal_type = proposal_type.split(b'</a>')[0].decode()
82         except:
83             logging.warning('Cannot determine proposal type for {}'\
84                             .format(proposid))
85             proposal_type = None
86
87     # Check for bad proposal types
88     if proposal_type not in VALID_PROPOSAL_TYPES:
89         logging.warning('Cannot determine proposal type for {}'\
90                         .format(proposid))
91         proposal_type = None
92
93     return proposal_type
94
95
96 def update_datasets_table(file_dict):
97     """Insert/update an entry for the file in the ``datasets`` table.
98
99     Parameters
100    -----
101    file_dict : dict
102        A dictionary containing various data useful for the ingestion
103        process.
104    """
105
106     session, base, engine = load_connection(SETTINGS['connection_string'])
107
108     # Check to see if a record exists for the rootname
109     query = session.query(Datasets) \
110         .filter(Datasets.rootname == file_dict['rootname'])
111     query_count = query.count()
112
113     # If there are no results, then perform an insert
114     if not query_count:
115
116         data_dict = {}
117         data_dict['rootname'] = file_dict['rootname']
118         data_dict[file_dict['filetype']] = file_dict['basename']
119
120         tab = Table('datasets', base.metadata, autoload=True)
121         insert_obj = tab.insert()
122
123         try:
124             insert_obj.execute(data_dict)
125         except IntegrityError as e:
126             logging.warning('{}: Unable to insert {} into datasets table: {}'\
127                             .format(file_dict['full_rootname'], file_dict['basename'], e))
128
129     # If there are results, add the filename to the existing entry
130     else:
131         data_dict = query.one().__dict__
132         del data_dict['_sa_instance_state']
133         data_dict[file_dict['filetype']] = file_dict['basename']
134
135         try:
136             query.update(data_dict)
137         except IntegrityError as e:
138             logging.warning('{}: Unable to update {} in datasets table: {}'\
139                             .format(file_dict['full_rootname'], file_dict['basename'], e))
140
141     session.commit()
142     session.close()
143     engine.dispose()
144
145     logging.info('{}: Updated datasets table for {}.'\
146                 .format(file_dict['rootname'], file_dict['filetype']))

```

```

146 def update_header_table(file_dict, ext):
147     """Insert/update an entry for the file in the appropriate header
148     table (e.g. ``wfc_raw_0``).
149
150     The header table that get updated depend on the detector, filetype,
151     and extension.
152
153     Parameters
154     -----
155     file_dict : dict
156         A dictionary containing various data useful for the ingestion
157         process.
158     ext : int
159         The header extension.
160
161     """
162
163     # Check if header is an ingestable header before proceeding
164     valid_extnames = ['PRIMARY', 'SCI', 'ERR', 'DQ', 'UDL', 'jit', 'jif',
165                       'ASN', 'WHT', 'CTX']
166     ext_exists = True
167     try:
168         header = fits.getheader(file_dict['filename'], ext)
169         if ext == 0:
170             extname = 'PRIMARY'
171         else:
172             extname = header['EXTNAME']
173     except IndexError:
174         ext_exists = False
175         extname = None
176
177     # Ingest the header if it is ingestable
178     if ext_exists and extname in valid_extnames:
179
180         table = "{}_{}_{!s}".format(file_dict['detector'].upper(),
181                                     file_dict['filetype'].lower(),
182                                     str(ext))
183
184         exclude_list = ['HISTORY', 'COMMENT', 'ROOTNAME', 'FILENAME', '']
185         input_dict = {'rootname': file_dict['rootname'],
186                      'filename': file_dict['basename']}
187
188         try:
189             for key, value in header.items():
190                 key = key.strip()
191
192                 # Switch hypens to underscores
193                 if '-' in key:
194                     key = key.replace('-', '_')
195
196                 if key in exclude_list or value == "":
197                     continue
198                 elif key not in TABLE_DEFS[table.lower()]:
199                     logging.warning('{}: {} not in {}\\'
200                                     .format(file_dict['full_rootname'], key, table))
201                     continue
202
203                 input_dict[key.lower()] = value
204
205             insert_or_update(table, input_dict)
206             logging.info('{}: Updated {} table.' .format(file_dict['rootname'],
207                                               table))
208
209         except VerifyError as e:
210             logging.warning('\\tUnable to insert {} into {}: {}' .format(
211                             file_dict['rootname'], table, e))
212
213 def update_master_table(rootname_path):
214     """Insert/update an entry in the ``master`` table for the given
215     file.
216
217     Parameters
218     -----
219     rootname_path
220         The path to the rootname directory in the MAST cache.
221
222     """

```

```

223
224     rootname = os.path.basename(rootname_path)[-1]
225     path = rootname_path[-15:]
226     proposid = get_metadata_from_test_files(rootname_path, 'proposid')
227     proposal_type = get_proposal_type(proposid)
228
229     # Insert a record in the master table
230     data_dict = {'rootname': rootname,
231                  'path': path,
232                  'first_ingest_date': date.today().isoformat(),
233                  'last_ingest_date': date.today().isoformat(),
234                  'detector': get_metadata_from_test_files(rootname_path,
235                                              'detector'),
236                  'proposal_type': proposal_type}
237     insert_or_update('Master', data_dict)
238     logging.info('{}: Updated master table.'.format(rootname))
239
240
241 def ingest(rootname_path, filetype='all'):
242     """The main function of the ingest module. Ingest a given rootname
243     (and its associated files) into the various tables of the ``acsqll``
244     database.
245
246     If for some reason the file is unable to be ingested, a warning is
247     logged.
248
249     Parameters
250     -----
251     rootname_path : str
252         The path to the rootname directory in the MAST cache.
253     """
254
255     rootname = os.path.basename(rootname_path)[-1]
256     logging.info('{}: Begin ingestion'.format(rootname))
257
258     # Update the master table for the rootname
259     update_master_table(rootname_path)
260
261     if filetype == 'all':
262         search = '*.fits'
263     else:
264         search = '{}.fits'.format(filetype)
265     file_paths = glob.glob(os.path.join(rootname_path, search))
266
267     for filename in file_paths:
268         filetype = os.path.basename(filename).split('.')[0][10:]
269         if filetype in VALID_FILETYPES:
270
271             # Make dictionary that holds all the information you would ever
272             # want about the file
273             file_dict = make_file_dict(filename)
274
275             # Update header tables
276             if 'file_exts' in file_dict:
277                 for ext in file_dict['file_exts']:
278                     update_header_table(file_dict, ext)
279
280             # Update datasets table
281             update_datasets_table(file_dict)
282
283             # Make JPEGs and Thumbnails
284             if file_dict['filetype'] in ['raw', 'flt', 'flc']:
285                 make_jpeg(file_dict)
286             if file_dict['filetype'] == 'flt':
287                 make_thumbnail(file_dict)
288
289     logging.info('{}: End ingestion'.format(rootname))

```

```

acsql/ingest/make_file_dict.py

1 """Create a dictionary containing useful information for the ingestion
2 process.
3
4 The ``file_dict`` contains various information that can be used by
5 ``ingest.py`` (e.g. filesystem paths, observational metadata) and can
6 be used as a data container that can be easily passed around to various
7 functions.
8
9 Authors
10 -----
11     Matthew Bourque
12
13 Use
14 ---
15     This module and its functionars are intended to be imported and
16 used by ``acsql.ingest.ingest.py`` as such:
17     ::

18
19     from ascql.ingest.make_file_dict import get_detector
20     from ascql.ingest.make_file_dict import get_metadata_from_test_files
21     from ascql.ingest.make_file_dict import get_proposid
22     from acsql.ingest.make_file_dict import make_file_dict
23
24     make_file_dict(filename)
25     get_detector(filename)
26     get_metadata_from_test_files(rootname_path, keyword)
27     get_proposid(filename)
28
29 Dependencies
30 -----
31     External library dependencies include:
32
33     - ``astropy``
34 """
35
36 import glob
37 import logging
38 import os
39
40 from astropy.io import fits
41
42 from acsql.utils import utils
43 from acsql.utils import SETTINGS
44
45
46 def get_detector(filename):
47     """Return the ``detector`` associated with the given ``filename``,
48     if possible.
49
50     Parameters
51     -----
52     filename : str
53         The path to the file to attempt to get the ``detector`` header
54         keyword from.
55
56     Returns
57     -----
58     detector : str
59         The detector (e.g. ``'WFC'``)
60 """
61
62     if 'jst' in filename:
63         detector = fits.getval(filename, 'config', 0)
64         if detector == 'S/C': # FGS observation
65             detector = None
66         else:
67             detector = detector.lower().split('/')[-1]
68     else:
69         detector = fits.getval(filename, 'detector', 0).lower()
70
71     return detector
72
73
74 def get_metadata_from_test_files(rootname_path, keyword):
75     """Return the value of the given ``keyword`` and ``rootname_path``.
76

```

```

77 The given ``rootname_path`` is checked for various filetypes that
78 are believed to have the ``keyword`` that is sought, in order
79 of most likeliness: ``raw``, ``flt``, ``spt``, ``drz``, and
80 ``jit``. If a candidate file is found, it is used to determine
81 the value of the ``keyword`` in the primary header. If no
82 candidate file exists, or the ``keyword`` value cannot be
83 determined from the primary header, a ``value`` of ``None`` is
84 returned, essentially ending the ingestion process for the given
85 rootname.
86
87 Parameters
88 -----
89 rootname_path : str
90     The path to the rootname in the MAST cache.
91 keyword : str
92     The header keyword to determine the value of (e.g.
93     ``detector``)
94
95 Returns
96 -----
97 value : str or None
98     The header keyword value.
99 """
100
101 raw = glob.glob(os.path.join(rootname_path, '*raw.fits'))
102 flt = glob.glob(os.path.join(rootname_path, '*flt.fits'))
103 spt = glob.glob(os.path.join(rootname_path, '*spt.fits'))
104 drz = glob.glob(os.path.join(rootname_path, '*drz.fits'))
105 jit = glob.glob(os.path.join(rootname_path, '*jit.fits'))
106
107 for test_files in [raw, flt, spt, drz, jit]:
108     try:
109         test_file = test_files[0]
110         if keyword == 'detector':
111             value = get_detector(test_file)
112         elif keyword == 'proposid':
113             value = get_proposid(test_file)
114         break
115     except (IndexError, KeyError):
116         value = None
117
118 if not value:
119     logging.warning('Cannot determine {} for {}'\
120 .format(keyword, rootname_path))
121
122 return value
123
124
125 def get_proposid(filename):
126     """Return the proposal ID from the primary header of the given
127     ``filename``.
128
129     Parameters
130     -----
131     filename : str
132         The path to the file to get the ``proposid`` from.
133
134     Returns
135     -----
136     proposid : int
137         The proposal ID (e.g. ``12345``).
138 """
139
140 proposid = str(fits.getval(filename, 'proposid', 0))
141
142 return proposid
143
144
145 def make_file_dict(filename):
146     """Create a dictionary that holds information that is useful for
147     the ingestion process. This dictionary can then be passed around
148     the various functions of the module.
149
150     Parameters
151     -----
152     filename : str
153         The path to the file.

```

```

154
155     Returns
156     -----
157     file_dict : dict
158         A dictionary containing various data useful for the ingestion
159         process.
160     """
161
162     file_dict = {}
163
164     # Filename related keywords
165     file_dict['filename'] = os.path.abspath(filename)
166     file_dict['dirname'] = os.path.dirname(filename)
167     file_dict['basename'] = os.path.basename(filename)
168     file_dict['rootname'] = file_dict['basename'].split('_')[0][-1]
169     file_dict['full_rootname'] = file_dict['basename'].split('_')[0]
170     file_dict['filetype'] = file_dict['basename'].split('.fits')[0].split('_')[-1]
171     file_dict['proposid'] = file_dict['basename'][0:4]
172     file_dict['proposid_int'] = get_metadata_from_test_files(file_dict['dirname'], 'proposid')
173
174     # Metadata keywords
175     file_dict['detector'] = get_metadata_from_test_files(file_dict['dirname'], 'detector')
176     if file_dict['detector']:
177         file_dict['file_exts'] = getattr(utils, '{}_FILE_EXTS'.format(file_dict['detector'].upper()))[
178             file_dict['filetype']]
179
180     # JPEG related keywords
181     if file_dict['filetype'] in ['raw', 'flt', 'flc']:
182         file_dict['jpg_filename'] = file_dict['basename'].replace('.fits', '.jpg')
183         file_dict['jpg_dst'] = os.path.join(SETTINGS['jpeg_dir'], file_dict['proposid_int'], file_dict['
184             jpg_filename'])
185         file_dict['thumbnail_filename'] = file_dict['basename'].replace('.fits', '.thumb')
186         file_dict['thumbnail_dst'] = os.path.join(SETTINGS['thumbnail_dir'], file_dict['proposid_int'],
187             file_dict['thumbnail_filename']))
188     else:
189         file_dict['jpg_filename'] = None
190         file_dict['jpg_dst'] = None
191         file_dict['thumbnail_filename'] = None
192         file_dict['thumbnail_dst'] = None
193
194     return file_dict

```

acsq1/ingest/make_jpeg.py

```

1 """Create a "Quicklook" JPEG for the given observation.
2
3 A JPEG is created for every ``raw``, ``flt`` , and ``flc`` file and is
4 placed into the ``acsq1`` filesystem of JPEGs. The JPEGs are then
5 used by the ``acsq1`` web application to easily view ACS observaitons.
6
7 Authors
8 -----
9     Matthew Bourque
10
11 Use
12 ---
13     This module is inteneded to be imported and used by
14     ``acsq1.ingest.ingest.py`` as such:
15     :::
16
17     from acsq1.ingest.make_jpeg import make_jpeg
18     make_jpeg(file_dict)
19
20 Dependencies
21 -----
22     External library dependencies include:
23
24     - ``astropy``
25     - ``numpy``
26     - ``PIL``
27 """
28
29 import logging
30 import os
31
32 from astropy.io import fits
33 import numpy as np
34 from PIL import Image

```

```

35
36
37 def make_jpeg(file_dict):
38     """Creates a JPEG for the given file.
39
40     Parameters
41     -----
42     file_dict : dict
43         A dictionary containing various data useful for the ingestion
44         process.
45     """
46
47     logging.info(' {}: Creating JPEG'.format(file_dict['rootname']))
48
49     hdulist = fits.open(file_dict['filename'], mode='readonly')
50     data = hdulist[1].data
51
52     # If the image is full-frame WFC, add on the other extension
53     if len(hdulist) > 4 and hdulist[0].header['detector'] == 'WFC':
54         if hdulist[4].header['EXTNAME'] == 'SCI':
55             data2 = hdulist[4].data
56             height = data.shape[0] + data2.shape[0]
57             width = data.shape[1]
58             new_array = np.zeros((height, width))
59             new_array[0:int(height/2), :] = data
60             new_array[int(height/2):height, :] = data2
61             data = new_array
62
63     # Clip the top and bottom 1% of pixels.
64     top = np.percentile(data, 99)
65     data[data > top] = top
66     bottom = np.percentile(data, 1)
67     data[data < bottom] = bottom
68
69     # Scale the data.
70     data = data - data.min()
71     data = (data / data.max()) * 255.
72     data = np.flipud(data)
73     data = np.uint8(data)
74
75     # Create parent JPEG directory if necessary
76     jpg_dir = os.path.dirname(file_dict['jpg_dst'])
77     if not os.path.exists(jpg_dir):
78         os.makedirs(jpg_dir)
79         logging.info(' {}: Created directory {}'.format(file_dict['rootname'], jpg_dir))
80
81     # Write the image to a JPEG
82     image = Image.fromarray(data)
83     image.save(file_dict['jpg_dst'])
84
85     # Close the hdulist
86     hdulist.close()

```

acssql/ingest/make_thumbnail.py

```

1 """Create a "Quicklook" Thumbnail for the given observation.
2
3 A Thumbnail image is created from a given JPEG file (see module
4 documentation for ``make_jpeg.py`` for further details). A
5 thumbnail is a JPEG image reduced to 128 x 128 pixel size. The
6 thumbnails are used by the ``acssql`` web application for quickly viewing
7 many JPEGs.
8
9 Authors
10 -----
11     Matthew Bourque
12
13 Use
14 ---
15     This module is intended to be imported and used by
16     ``acssql.ingest.ingest.py`` as such:
17     :::
18
19     from acssql.ingest.make_thumbnail import make_thumbnail
20     make_thumbnail(file_dict)
21
22 Dependencies

```

```

23 -----
24     External library dependencies include:
25
26     - ``PIL``
27 """
28
29 import logging
30 import os
31 import shutil
32
33 from PIL import Image
34
35
36 def make_thumbnail(file_dict):
37     """Creates a 128 x 128 pixel 'thumbnail' JPEG for the given file.
38
39     Parameters
40     -----
41     file_dict : dict
42         A dictionary containing various data useful for the ingestion
43         process.
44 """
45
46 logging.info(' {}: Creating Thumbnail'.format(file_dict['rootname']))
47
48 # Create parent Thumbnail directory if necessary
49 thumb_dir = os.path.dirname(file_dict['thumbnail_dst'])
50 if not os.path.exists(thumb_dir):
51     try:
52         os.makedirs(thumb_dir)
53         logging.info(' {}: Created directory {}'\ \
54             .format(file_dict['rootname'], thumb_dir))
55     except FileExistsError:
56         pass
57
58 # Make a copy of the JPEG in the thumbnail directory
59 shutil.copyfile(file_dict['jpg_dst'], file_dict['thumbnail_dst'])
60
61 # Open the copied JPEG and reduce its size
62 image = Image.open(file_dict['thumbnail_dst'])
63 image.thumbnail((128, 128), Image.ANTIALIAS)
64 image.save(file_dict['thumbnail_dst'], 'JPEG')

```

acsq1/scripts/ingest_production.py

```

1 #! /usr/bin/env python
2
3 """Performs ingestion of HST/ACS data into the ``acsq1`` database and
4 filesystem.
5
6 This script is a wapper around ``acsq1.ingest.ingest.py`` to ingest
7 multiple rootnames into the system. The user may supply a list of
8 individual rootnames to ingest, or (by default) ingest whichever
9 rootnames exist in the MAST cache but yet to exist in the ``acsq1``
10 database.
11
12 See ``acsq1.ingest.ingest.py`` module docstrings for further
13 information on the ingestion process.
14
15 Authors
16 -----
17     Matthew Bourque
18
19 Use
20 ---
21     This script is inteneded to be executed from the command line as
22     such:
23     :::
24
25     python ingest_production.py [-i|--ingest_filelist]
26         ['-f|--filetype']
27
28     Parameters:
29     (Optional) [-i|--ingest_filelist] - A text file containing
30         individual rootnames to be ingested. If not supplied, this
31         module will determine which rootnames are to be ingested by
32         comparing the MAST cache against what already exists in the
33         ``acsq1`` database.

```

```

34     (Optional) [-f|--filetype] - The type of file to ingest. May be
35         an individual filetype (e.g. ``flt``) or ``all`` to ingest all
36         filetypes. ``all`` is the default value.
37 """
38
39 import argparse
40 import glob
41 import logging
42 import multiprocessing
43 import os
44
45 from astropy.io import fits
46
47 from acsql.database.database_interface import Master, session
48 from acsql.ingest.ingest import ingest
49 from acsql.utils.utils import SETTINGS, setup_logging, VALID_FILETYPES
50
51
52 def get_rootnames_to_ingest():
53     """Return a list of paths to rootnames in the filesystem that need
54     to be ingested (i.e. do not already exist in the ``acsql``
55     database).
56
57     Returns
58     ------
59     rootnames_to_ingest : list
60         A list of full paths to rootnames that exist in the filesystem
61         but not in the ``acsql`` database.
62 """
63
64     logging.info('Gathering files to ingest')
65
66     # Query the database to determine which rootnames already exist
67     results = session.query(Master.rootname).all()
68     db_rootnames = set([item[0] for item in results])
69
70     # Gather list of rootnames that exist in the filesystem
71     fsys_paths = glob.glob(os.path.join(SETTINGS['filesystem'], 'j*', '*'))
72     fsys_rootnames = set([os.path.basename(item)[-1] for item in fsys_paths])
73
74     # Determine new rootnames to ingest
75     new_rootnames = fsys_rootnames - db_rootnames
76
77     # Re-retrieve the full paths
78     rootnames_to_ingest = [item for item in fsys_paths if
79                           os.path.basename(item)[-1] in new_rootnames]
80
81     logging.info('{} rootnames in database'.format(len(db_rootnames)))
82     logging.info('{} rootnames in filesystem'.format(len(fsys_rootnames)))
83     logging.info('{} rootnames to ingest'.format(len(rootnames_to_ingest)))
84
85     return rootnames_to_ingest
86
87
88 def ingest_production(filetype, ingest_filelist):
89     """Perform ingestion on the given filelist of rootnames (or if not
90     provided, any new rootnames that exist in the MAST filesystem but
91     not in the ``acsql`` database) for the given ``filetype`` (or all
92     filetypes if ``filetype`` == ``all``).
93
94     Parameters
95     ------
96     filetype : str
97         The filetype to ingest (e.g. ``flt``, or ``all``)
98     ingest_filelist : str or None
99         The path to a file that contains rootnames to ingest. If
100         ``None``, then the acsql database and MAST filesystem are
101         used to determine new rootnames to ingest.
102 """
103
104     if ingest_filelist:
105         with open(ingest_filelist) as f:
106             rootnames = f.readlines()
107             rootnames = [rootname.strip().lower() for rootname in rootnames]
108             rootnames = [os.path.join(SETTINGS['filesystem'], rootname[0:4], rootname) for rootname in
109                         rootnames]
109     else:

```

```

110     rootnames = get_rootnames_to_ingest()
111
112     pool = multiprocessing.Pool(processes=SETTINGS['ncores'])
113     filetypes = [filetype for item in rootnames]
114     mp_args = [(rootname, filetype) for rootname, filetype in zip(rootnames, filetypes)]
115     pool.starmap(ingest, mp_args)
116
117     logging.info('Process Complete.')
118
119
120 def parse_args():
121     """Parse command line arguments. Returns ``args`` object
122
123     Returns
124     ------
125     args : obj
126         An argparse object containing all of the arguments
127     """
128
129     VALID_FILETYPES.extend(['all'])
130
131     # Create help strings
132     filetype_help = 'The filetypes to ingest. Can be one of the following: '
133     filetype_help += '{}. If "all", then all '.format(VALID_FILETYPES)
134     filetype_help += 'available filetypes for each rootname will be ingested. '
135     filetype_help += 'If a specific filetype is given, then only that '
136     filetype_help += 'filetype will be ingested. "all" is the default option.'
137     ingest_filelist_help = 'A file containing a list of rootnames to ingest. '
138     ingest_filelist_help += 'If not provided, then the acsql database is used'
139     ingest_filelist_help += 'to determine which files get ingested.'
140
141     # Add arguments
142     parser = argparse.ArgumentParser()
143     parser.add_argument('-f --filetype',
144                         dest='filetype',
145                         action='store',
146                         required=False,
147                         default='all',
148                         help=filetype_help)
149     parser.add_argument('-i --ingest_filelist',
150                         dest='ingest_filelist',
151                         action='store',
152                         required=False,
153                         default=None,
154                         help=ingest_filelist_help)
155
156     # Parse args
157     args = parser.parse_args()
158
159     # Test the args
160     test_args(args)
161
162     return args
163
164
165 def test_args(args):
166     """Test the command line arguments to ensure that they are valid.
167
168     Parameters
169     ------
170     args : obj
171         An argparse objects containing all of the arguments.
172
173     Raises
174     ------
175     AssertionError
176         If any of the argument conditions fail.
177     """
178
179     # Ensure the filetype is valid
180     VALID_FILETYPES.extend(['all'])
181     assert args.filetype in VALID_FILETYPES,\n        '{} is not a valid filetype'.format(args.filetype)
182
183     # Ensure that the ingest_filelist exists
184     if args.ingest_filelist:
185         assert os.path.exists(args.ingest_filelist), \

```

```

187     '{} does not exist.'.format(args.ingest_filelist)
188
189
190 if __name__ == '__main__':
191     module = os.path.basename(__file__).strip('.py')
192     setup_logging(module)
193
194     args = parse_args()
195     ingest_production(args.filetype, args.ingest_filelist)

```

acsq1/utils/utils.py

```

1 """This module contains several functions that are useful to various
2 modules within the ``acsq1`` package. See individual function
3 docstrings for further information.
4
5 Authors
6 -----
7     Matthew Bourque
8
9 Use
10 ---
11
12     The functions within this module are intened to be imported by
13     various acsq1 modules and scripts, as such:
14     ::
15
16         from acsq1.utils.utils import insert_or_update
17         from acsq1.utils.utils import SETTINGS
18         from acsq1.utils.utils import setup_logging
19
20     There also exists static importable data:
21     ::
22
23         from acsq1.utils.utils import FILE_EXTS
24         from acsq1.utils.utils import TABLE_DEFS
25
26 Dependencies
27 -----
28     External library dependencies include:
29
30     - ``acsq1``
31     - ``astropy``
32     - ``numpy``
33     - ``sqlalchemy``
34 """
35
36 import datetime
37 import getpass
38 import glob
39 import logging
40 import os
41 import socket
42 import sys
43 import yaml
44
45 import astropy
46 import numpy
47 import sqlalchemy
48 from sqlalchemy import Table
49 from sqlalchemy.exc import DataError
50 from sqlalchemy.exc import IntegrityError
51 from sqlalchemy.exc import InternalError
52
53 import acsq1
54
55 __config__ = os.path.realpath(os.path.join(os.getcwd(),
56                                         os.path.dirname(__file__)))
57
58 # Define possible detector/filetype/extension combinations
59 WFC_FILE_EXTS = {'jif': [0, 1, 2, 3, 4, 5, 6],
60                  'jit': [0, 1, 2, 3, 4, 5, 6],
61                  'flt': [0, 1, 2, 3, 4, 5, 6],
62                  'flc': [0, 1, 2, 3, 4, 5, 6],
63                  'drz': [0, 1, 2, 3],
64                  'drc': [0, 1, 2, 3],
65                  'raw': [0, 1, 2, 3, 4, 5, 6],

```

```

66     'crj': [0, 1, 2, 3, 4, 5, 6],
67     'crc': [0, 1, 2, 3, 4, 5, 6],
68     'spt': [0, 1],
69     'asn': [0, 1] }
70
71 SBC_FILE_EXTS = {'jif': [0, 1, 2],
72                   'jit': [0, 1, 2],
73                   'flt': [0, 1, 2, 3],
74                   'drz': [0, 1, 2, 3],
75                   'raw': [0, 1, 2, 3],
76                   'spt': [0, 1],
77                   'asn': [0, 1] }
78
79 HRC_FILE_EXTS = {'jif': [0, 1, 2],
80                   'jit': [0, 1, 2],
81                   'flt': [0, 1, 2, 3],
82                   'drz': [0, 1, 2, 3],
83                   'raw': [0, 1, 2, 3],
84                   'crj': [0, 1, 2, 3],
85                   'spt': [0, 1],
86                   'asn': [0, 1] }
87
88 # Define ingestable filetypes
89 VALID_FILETYPES = ['jif', 'jit', 'flt', 'flic', 'drz', 'drc', 'raw', 'crj',
90                     'crc', 'spt', 'asn']
91
92 # Define value proposal types
93 VALID_PROPOSAL_TYPES = ['CAL/ACS', 'CAL/OTA', 'CAL/STIS', 'CAL/WFC3',
94                          'ENG/ACS', 'GO', 'GO/DD', 'GO/PAR', 'GTO/ACS',
95                          'GTO/COS', 'NASA', 'SM3/ACS', 'SM3/ERO', 'SM4/ACS',
96                          'SM4/COS', 'SM4/ERO', 'SNAP']
97
98
99 def get_settings():
100     """Returns the settings that are located in the acsql config file.
101
102     Returns
103     -----
104     settings : dict
105         A dictionary with setting key/value pairs.
106     """
107
108     with open(os.path.join(__config__, 'config.yaml'), 'r') as f:
109         settings = yaml.load(f)
110
111     return settings
112
113
114 SETTINGS = get_settings()
115
116
117 def setup_logging(module):
118     """Configures a log file that logs the execution of the given
119     module. Log files are written to the log_dir that is set in the
120     config.yaml configuration file. The filename of the log file is
121     <module>_<timestamp>.log.
122
123     Parameters
124     -----
125     module : str
126         The name of the module to log.
127     """
128
129     SETTINGS = get_settings()
130
131     # Configure logging
132     timestamp = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M')
133     filename = '{0}_{1}.log'.format(module, timestamp)
134     logfile = os.path.join(SETTINGS['log_dir'], filename)
135     logging.basicConfig(
136         filename=logfile,
137         format='%(asctime)s %(levelname)s: %(message)s',
138         datefmt='%m/%d/%Y %H:%M:%S',
139         level=logging.INFO)
140
141     # Log environment information
142     logging.info('User: {0}'.format(getpass.getuser()))

```

```

143     logging.info('System: {0}'.format(socket.gethostname()))
144     logging.info('Python Version: {0}'.format(sys.version.replace('\n', '')))
145     logging.info('Python Path: {0}'.format(sys.executable))
146     logging.info('NumPy Version: {0}'.format(numpy.__version__))
147     logging.info('NumPy Path: {0}'.format(numpy.__path__[0]))
148     logging.info('Astropy Version: {0}'.format(astropy.__version__))
149     logging.info('Astropy Path: {0}'.format(astropy.__path__[0]))
150     logging.info('SQLAlchemy Version: {0}'.format(sqlalchemy.__version__))
151     logging.info('SQLAlchemy Path: {0}'.format(sqlalchemy.__path__[0]))
152
153
154 def get_table_defs():
155     """Return a dictionary containing the columns for each database
156     table, as taken from the table_definition text files.
157
158     Returns
159     -----
160     table_defs : dict
161         A dictionary whose keys are detector/file_type/extension
162         configurations (e.g. 'wfc_flt_0') and whose values are lists
163         of column names for the corresponding table.
164     """
165
166     # Get table definition files
167     table_def_directory = os.path.realpath(os.path.join(os.getcwd(),
168                                                     os.path.dirname(__file__)))
169     table_def_directory = table_def_directory.replace('utils', 'database/table_definitions')
170     table_def_files = glob.glob(os.path.join(table_def_directory, '*.txt'))
171
172     table_defs = {}
173
174     for table_def_file in table_def_files:
175
176         configuration = os.path.basename(table_def_file).split('.txt')[0]
177         with open(table_def_file, 'r') as f:
178             contents = f.readlines()
179             contents = [item.strip() for item in contents]
180             columns = [item.split(',') [0] for item in contents]
181             table_defs[configuration] = columns
182
183     return table_defs
184
185 TABLE_DEFS = get_table_defs()
186
187
188 def insert_or_update(table, data_dict):
189     """Insert or update a record in the given ``table`` with the data
190     in the ``data_dict``.
191
192     A record is inserted if the primary key of the record does not
193     already exist in the ``table``. A record is updated if it does
194     already exist.
195
196     Parameters
197     -----
198     table : str
199         The name of the table to insert/update into.
200     data_dict : dict
201         A dictionary containing the data to insert/update.
202     """
203
204     table_obj = getattr(acsqldb.database.database_interface, table)
205     session, base, engine = acsqldb.database.database_interface.\
206         load_connection(SETTINGS['connection_string'])
207
208     # Check to see if a record exists for the rootname
209     query = session.query(table_obj)\.
210         .filter(getattr(table_obj, 'rootname') == data_dict['rootname'])
211     query_count = query.count()
212
213     # If there are no results, then perform an insert
214     if not query_count:
215         tab = Table(table.lower(), base.metadata, autoload=True)
216         insert_obj = tab.insert()
217         try:
218             insert_obj.execute(data_dict)
219         except (DataError, IntegrityError, InternalError) as e:
```

```

220     logging.warning('\tUnable to insert {} into {}: {}'.format(
221         data_dict['rootname'], table, e))
222
223     else:
224         query.update(data_dict)
225
226     session.commit()
227     session.close()
228     engine.dispose()

```

acsqsql/website/acsqsql_webapp.py

```

1  #! /usr/bin/env python
2
3 """This module serves as the ``acsqsql`` web application, which allows
4 users to view image data and interact with the ``acsqsql`` database.
5
6 The module is build using the ``flask`` python web framework. See
7 accompanying ``data_containers``, ``query_form``, and ``form_choices``
8 modules for further information.
9
10 Authors
11 -----
12
13     - Matthew Bourque
14     - Meredith Durbin
15
16 Use
17 ---
18
19     This module is intended to be executed on a web server, but it can
20     also be run locally:
21     :::
22
23     python acsqsql_webapp.py
24     <go to localhost:5000 in a browser>
25
26 Dependencies
27 -----
28
29     - ``acsqsql``
30     - ``flask``
31     - ``numpy``
32 """
33
34 from collections import OrderedDict
35 import glob
36 import os
37
38 from flask import Flask, render_template, request, Response
39 import numpy as np
40
41 from acsqsql.utils.utils import SETTINGS
42 from acsqsql.website.data_containers import get_view_image_dict
43 from acsqsql.website.data_containers import get_view_proposal_dict
44 from acsqsql.website.data_containers import get_view_query_results_dict
45 from acsqsql.website.query_form import get_query_form
46 from acsqsql.website.query_lib import generate_csv
47 from acsqsql.website.query_lib import get_query_results
48
49 app = Flask(__name__)
50
51
52 @app.route('/archive/')
53 def archive():
54     """Returns webpage containing links to all ACS archive proposals.
55
56     Returns
57     -----
58     template : obj
59         The ``archive.html`` template.
60     """
61
62     # Get list of all proposal numbers
63     proposal_list = glob.glob(os.path.join(SETTINGS['jpeg_dir'], '*'))
64     proposal_list = sorted([int(os.path.basename(item)) for item in proposal_list])
65
66     # rearrange list so that it appears in multiple columns

```

```

67     ncols = 12
68     if len(proposal_list) % ncols != 0:
69         proposal_list.extend([''] * (ncols - (len(proposal_list) % ncols)))
70     proposal_array = np.asarray(proposal_list).reshape(ncols, int(len(proposal_list) / ncols)).T
71
72     return render_template('archive.html', proposal_array=proposal_array)
73
74
75 @app.route('/database/')
76 @app.route('/database/results')
77 def database():
78     """Returns webpage containing a query form for querying the
79     ``acs`` database.
80
81     Returns
82     ------
83     template : obj
84         The ``database.html`` webpage.
85     """
86
87     query_form = get_query_form(request.args)
88
89     if request.query_string:
90         if query_form.validate():
91             query_form_dict = request.args.to_dict(flat=False)
92             query_results_dict = get_query_results(query_form_dict)
93
94             results = query_results_dict['query_results']
95             num_results = query_results_dict['num_results']
96             output_format = query_results_dict['output_format']
97             output_columns = query_results_dict['output_columns']
98
99             # If something went wrong with the query
100            if num_results is None:
101                template = render_template(
102                    'database_error.html',
103                    form=query_form,
104                    msg=num_results)
105
106            # If the query returned no results
107            elif num_results == 0:
108                results = True
109                template = render_template(
110                    'database_table.html',
111                    results=results,
112                    num_results=num_results)
113
114            # If the query returned results
115            else:
116
117                # For HTML table output format
118                if output_format == ['table']:
119                    template = render_template(
120                        'database_table.html',
121                        results=results,
122                        num_results=num_results,
123                        output_columns=output_columns)
124
125                # For CSV output format
126                elif output_format == ['csv']:
127                    template = Response(generate_csv(output_columns, results), mimetype='text/csv')
128                    template.headers['Content-Disposition'] = 'attachment; filename=query_results.csv'
129
130                # For Thumbnail output format
131                elif output_format == ['thumbnails']:
132                    thumbnail_dict = get_view_query_results_dict(query_results_dict)
133                    template = render_template('view_query_results.html', thumbnail_dict=thumbnail_dict)
134
135            else:
136                template = render_template('database_error.html', form=query_form)
137
138        return template
139
140    # Form was not validated
141    else:
142        return render_template('database_error.html', form=query_form)
143

```

```

144     else:
145         return render_template('database.html', form=query_form)
146
147
148 def handle_500(trace):
149     """Handle 500 error.
150
151     Parameters
152     -----
153     trace : str
154         The traceback of the error.
155
156     Returns
157     -----
158     template : obj
159         The ``500.html`` template.
160     """
161
162     trace_html = trace.replace('\n', '<br>')
163
164     return render_template('500.html', trace_html=trace_html)
165
166
167 @app.route('/')
168 def main():
169     """Generates the ``acsq1`` website homepage.
170
171     Returns
172     -----
173     template : obj
174         The ``index.html`` template.
175     """
176
177     return render_template('index.html')
178
179
180 @app.errorhandler(404)
181 def page_not_found(error):
182     """Redirects any nonexistent URL to 404 page.
183
184     Parameters
185     -----
186     error : obj
187         The ``error`` thrown.
188
189     Returns
190     -----
191     template : obj
192         The ``404.html`` template.
193     """
194
195     return render_template('404.html'), 404
196
197
198 # @app.route('/archive/<proposal>/<filename>/<fits_type>/header/')
199 # def view_header(proposal, filename, fits_type):
200 #     """
201 #     """
202
203 #     header_dict = get_view_header_dict()
204 #     return render_template('header.html', header_dict=header_dict)
205
206
207 @app.route('/archive/<proposal>/<filename>/')
208 @app.route('/archive/<proposal>/<filename>/<fits_type>/')
209 def view_image(proposal, filename, fits_type='flt'):
210     """Returns webpage for viewing a single JPEG image, along with some
211     useful metadata and links to additional information/downloads.
212
213     If an invalid ``fits_type`` is supplied, a 404 page is returned.
214
215     Parameters
216     -----
217     proposal : str
218         The proposal ID (e.g. ``'12345'``).
219     filename : str
220         The 9-character IPPSSOOT rootname (e.g. ``'jcye04zsq'``).

```

```

221     fits_type : str
222         The JPEG FITS type to view. Can either be ``raw'', ``flt'', or
223         ``flc''.
224
225     Returns
226     -----
227     template : obj
228         The ``view_image.html'' template.
229     """
230
231     if fits_type in ['raw', 'flt', 'flc']:
232         image_dict = get_view_image_dict(proposal, filename, fits_type)
233         return render_template('view_image.html', image_dict=image_dict)
234     else:
235         return render_template('404.html'), 404
236
237
238 @app.route('/archive/<proposal>/')
239 def view_proposal(proposal):
240     """Returns webpage for viewing all thumbnails for a given
241     ``proposal'', along with some metadata and links to additional
242     information.
243
244     If an invalid proposal is supplied, a 404 page is returned.
245
246     Parameters
247     -----
248     proposal : str
249         The proposal ID (e.g. ``'12345' '').
250
251     Returns
252     -----
253     template : obj
254         The ``view_proposal.html'' template.
255     """
256
257     proposal_list = glob.glob(os.path.join(SETTINGS['jpeg_dir'], '*'))
258     proposal_list = [item.split('/')[-1] for item in proposal_list]
259
260     if proposal in proposal_list:
261         proposal_dict = get_view_proposal_dict(proposal)
262         return render_template('view_proposal.html', proposal_dict=proposal_dict)
263     else:
264         return render_template('404.html'), 404
265
266
267 if __name__ == '__main__':
268
269     app.run()

```

acsq1/website/data_containers.py

```

1     """Various functions for creating and returning various data to be used
2     by the ``acsq1'' web application.
3
4     This module contains functions to obtain image and proposal metadata
5     for use by the ``acsq1'' web application. See the ``acsq1_webapp''
6     module for further information about the web application.
7
8     Authors
9     -----
10
11     - Matthew Bourque
12     - Meredith Durbin
13
14     Use
15     ---
16
17     This module is intended to be imported and used by ``acsq1_webapp''
18     as such:
19     :::
20
21     from acsq1.website.data_containers import get_view_image_dict
22     from acsq1.website.data_containers import get_view_proposal_dict
23
24     image_dict = get_view_image_dict(proposal, filename, fits_type)
25     proposal_dict = get_view_proposal_dict(proposal)
26

```

```

27 Dependencies
28 -----
29
30     - ``acsq1``
31 """
32
33
34 import glob
35 import html
36 import os
37 import requests
38
39 from acsq1.database import database_interface
40 from acsq1.database.database_interface import Master
41 from acsq1.utils.utils import SETTINGS
42
43
44 def _get_image_lists(data_dict, fits_type):
45     """Add a list of JPEG and Thumbnail paths to the ``data_dict`` dictionary.
46
47     Parameters
48     -----
49     data_dict : dict
50         A dictionary containing data used to render a webpage.
51     fits_type : str
52         The FITS type. Can either be ``raw``, ``flt``, or ``flc``.
53
54     Returns
55     -----
56     data_dict : dict
57         A dictionary containing data used to render a webpage.
58 """
59
60
61 jpeg_proposal_path = os.path.join('static/img/jpeg', data_dict['proposal_id'])
62 thumb_proposal_path = os.path.join('static/img/thumbnails', data_dict['proposal_id'])
63
64 data_dict['jpeg'] = sorted(glob.glob(os.path.join(jpeg_proposal_path, '*flt.jpg')))
65 data_dict['thumb'] = sorted(glob.glob(os.path.join(thumb_proposal_path, '*flt.thumb')))
66
67 return data_dict
68
69
70 def _get_metadata_from_database(data_dict):
71     """Add observation metadata (e.g. ``aperture``, ``exptime``, etc.) to the ``data_dict`` by querying the ``acsq1`` database.
72
73     Parameters
74     -----
75     data_dict : dict
76         A dictionary containing data used to render a webpage.
77
78     Returns
79     -----
80     data_dict : dict
81         A dictionary containing data used to render a webpage.
82 """
83
84
85 session = getattr(database_interface, 'session')
86
87 results = []
88 for rootname in data_dict['rootnames']:
89
90     detector = session.query(Master.detector) \
91         .filter(Master.rootname == rootname).one()[0]
92
93     table = getattr(database_interface, '{}_raw_0'.format(detector))
94     result = session.query(
95         table.aperture, table.exptime, table.filter1, table.filter2,
96         table.targname, table.date_obs, table.time_obs, table.exptstart,
97         table.expflag, table.quality, table.ra_targ, table.dec_targ,
98         table.pr_inv_f, table.pr_inv_l).filter(table.rootname == rootname).one()
99     result = [item for item in result]
100    result.append(detector)
101    results.append(result)
102
103 session.close()

```

```

104     # Parse the results
105     data_dict['detectors'] = [detector for item in results]
106     data_dict['apertures'] = [item[0] for item in results]
107     data_dict['exptimes'] = [item[1] for item in results]
108     data_dict['filter1s'] = [item[2] for item in results]
109     data_dict['filter2s'] = [item[3] for item in results]
110     data_dict['targnames'] = [item[4] for item in results]
111     data_dict['dateobss'] = [item[5] for item in results]
112     data_dict['timeobss'] = [item[6] for item in results]
113     data_dict['expstarts'] = [item[7] for item in results]
114     data_dict['expflags'] = [item[8] for item in results]
115     data_dict['qualitys'] = [item[9] for item in results]
116     data_dict['ras'] = [item[10] for item in results]
117     data_dict['decs'] = [item[11] for item in results]
118     data_dict['pi_firsts'] = [item[12] for item in results]
119     data_dict['pi lasts'] = [item[13] for item in results]
120
121     return data_dict
122
123
124
125 def _get_buttons_dict(data_dict):
126     """Add data used for various buttons on the ``/archive/<proposal>``
127     page or ``/database/results/`` page to the ``data_dict``.
128
129     Parameters
130     -----
131     data_dict : dict
132         A dictionary containing data for the given
133         ``/archive/<proposal>`` or ``/database/results/`` page, such as
134         ``visits`` and ``targnames``.
135
136     Returns
137     -----
138     data_dict : dict
139         A dictionary containing data for the given
140         ``/archive/<proposal>`` or ``/database/results/`` page, such as
141         ``visits`` and ``targnames``.
142     """
143
144     data_dict['buttons'] = {}
145     data_dict['buttons']['detector'] = sorted(set(data_dict['detectors']))
146     data_dict['buttons']['visit'] = sorted(set(data_dict['visits']))
147     data_dict['buttons']['target'] = sorted(set(data_dict['targnames']))
148     data_dict['buttons']['filter'] = sorted(set([
149         '{} / {}'.format(filter1, filter2)
150         for filter1, filter2
151         in zip(data_dict['filter1s'], data_dict['filter2s'])]))
152
153     return data_dict
154
155
156 def _get_proposal_status(data_dict):
157     """Add proposal status information (e.g. ``proposal_title``,
158     ``cycle``, etc.) to the ``data_dict``.
159
160     The proposal status information is scraped from the proposal
161     status webpage.
162
163     Parameters
164     -----
165     data_dict : dict
166         A dictionary containing data used to render a webpage.
167
168     Returns
169     -----
170     data_dict : dict
171         A dictionary containing data used to render a webpage.
172     """
173
174     data_dict['status_page'] = (
175         'http://www.stsci.edu/cgi-bin/get-proposal-info?id={}'
176         '&submit=Go&observatory=HST').format(data_dict['proposal_id'])
177
178     req = requests.get(data_dict['status_page'], timeout=3)
179
180     if req.ok:

```

```

181     status_string = req.content.decode()
182     data_dict['proposal_title'] = html.unescape(status_string.\
183         split('<b>Title:</b>')[1].\
184         split('<br>')[0])
185     data_dict['cycle'] = html.unescape(status_string.\
186         split('<b>Cycle:</b>')[1].\
187         split('<br>')[0])
188     data_dict['schedule'] = html.unescape(status_string.\
189         split('/proposal-help-HST.html#')[2].\
190         split('>')[0])
191
192
193 else:
194     print('Request failed: {}'.format(data_dict['status_page']))
195     data_dict['proposal_title'] = 'proposal title unavailable'
196     data_dict['cycle'] = None
197     data_dict['schedule'] = None
198
199 return data_dict
200
201
202 def _initialize_data_dict(proposal, fits_type='flt'):
203     """Create and return a dictionary containing commonly used data
204     amongst ``/archive/<proposal>`` and
205     ``/archive/<proposal>/<filename>`` webpages.
206
207     Parameters
208     -----
209     proposal : str
210         The proposal number (e.g. ``12345``).
211     fits_type : str
212         The FITS type. Can be ``raw``, ``flt``, or ``flc``.
213
214     Returns
215     -----
216     data_dict : dict
217         A dictionary containing data used to render a webpage.
218     """
219
220     data_dict = {}
221     data_dict['proposal_id'] = proposal
222     data_dict = _get_image_lists(data_dict, fits_type)
223     data_dict['rootnames'] = [os.path.basename(item).split('_')[0][-1] for item in data_dict['jpeg']]
224     data_dict['filenames'] = [os.path.basename(item).split('_')[0] for item in data_dict['jpeg']]
225     data_dict['num_images'] = len(data_dict['jpeg'])
226     data_dict = _get_proposal_status(data_dict)
227
228 return data_dict
229
230
231 # def get_view_header_dict(filename, fits_type='flt'):
232 #     """
233 #     """
234
235 #     header_dict = {}
236 #     header_dict['filename'] = filename
237 #     header_dict['fits_type'] = fits_type.upper()
238
239 #     return header_dict
240
241 def get_view_image_dict(proposal, filename, fits_type='flt'):
242     """Return a dictionary containing data used for the
243     ``/archive/<proposal>/<filename>`` webpage.
244
245     Parameters
246     -----
247     proposal : str
248         The proposal number (e.g. ``12345``).
249     filename : str
250         The 9-character IPPSSOOT rootname (e.g. ``jcye04zsq``.)
251     fits_type : str
252         The JPEG FITS type to view. Can either be ``raw``, ``flt``, or
253         ``flc``.
254
255     Returns
256     -----
257     image_dict : dict

```

```

258     A dictionary containing data used for the
259     ``/archive/<proposal>/<filename>`` webpage.
260 """
261
262 image_dict = _initialize_data_dict(proposal, fits_type)
263 image_dict['fits_type'] = fits_type.upper()
264 image_dict['filename'] = filename
265 image_dict['rootname'] = filename[:-1]
266 image_dict = _get_metadata_from_database(image_dict)
267 image_dict['index'] = image_dict['filenames'].index(image_dict['filename'])
268 image_dict['page'] = image_dict['index'] + 1
269 image_dict['expstart'] = image_dict['expstarts'][image_dict['index']]
270 image_dict['filter1'] = image_dict['filter1s'][image_dict['index']]
271 image_dict['filter2'] = image_dict['filter2s'][image_dict['index']]
272 image_dict['aperture'] = image_dict['apertures'][image_dict['index']]
273 image_dict['exptime'] = image_dict['exptimes'][image_dict['index']]
274 image_dict['expflag'] = image_dict['expflags'][image_dict['index']]
275 image_dict['quality'] = image_dict['qualities'][image_dict['index']]
276 image_dict['ra'] = image_dict['ras'][image_dict['index']]
277 image_dict['dec'] = image_dict['decs'][image_dict['index']]
278 image_dict['targname'] = image_dict['targnames'][image_dict['index']]
279 image_dict['pi_first_name'] = image_dict['pi_firsts'][image_dict['index']]
280 image_dict['pi_last_name'] = image_dict['pi_lasts'][image_dict['index']]
281 image_dict['view_url'] = 'archive/{}/{}{}.format(image_dict['proposal_id'], image_dict['filename'],
282 fits_type)
283 image_dict['fits_links'] = {}
284 image_dict['first'] = image_dict['index'] == 0
285 image_dict['last'] = image_dict['index'] == image_dict['num_images'] - 1
286
287 # Determine path to JPEG
288 jpeg_path = '/static/img/jpeg/{}/{}/{}.jpg'.format(image_dict['proposal_id'], image_dict['filename'],
289 fits_type)
290 jpeg_path_abs = os.path.join(SETTINGS['jpeg_dir'], image_dict['proposal_id'], '{}_{}.jpg'.format(
291 image_dict['filename'], fits_type))
292 if os.path.exists(jpeg_path_abs):
293     image_dict['image'] = jpeg_path
294 else:
295     image_dict['image'] = None
296
297 # Determine next and previous images, if possible
298 if not image_dict['last']:
299     image_dict['next'] = {'proposal': image_dict['proposal_id'], 'filename': image_dict['filenames'][image_dict['index'] + 1], 'fits_type': fits_type}
300 if not image_dict['first']:
301     image_dict['prev'] = {'proposal': image_dict['proposal_id'], 'filename': image_dict['filenames'][image_dict['index'] - 1], 'fits_type': fits_type}
302
303 # Determine other available JPEGs for given observation
304 jpeg_types = glob.glob(jpeg_path_abs.replace('{}.jpg'.format(fits_type), '*.jpg'))
305 jpeg_types = [os.path.basename(item).split('_')[-1].split('.')[0] for item in jpeg_types]
306 jpeg_types = [item for item in jpeg_types if item.upper() != image_dict['fits_type']]
307 image_dict['available_jpegs'] = {}
308 for jpeg_type in jpeg_types:
309     image_dict['available_jpegs'][jpeg_type] = image_dict['view_url'].replace(fits_type, jpeg_type)
310
311 # For downloading the files
312 # image_dict['proposal_name'] = image_dict['filename'][0:4]
313 # image_dict['fits_links']['FLT'] = os.path.join(
314 #     SETTINGS['filesystem'],
315 #     image_dict['proposal_name'],
316 #     image_dict['filename'],
317 #     '{}_flt.fits'.format(image_dict['filename']))
318
319 return image_dict
320
321
322 def get_view_proposal_dict(proposal):
323     """Return a dictionary containing data used for the
324     ``/archive/<proposal>`` webpage.
325
326 Parameters
327 -----
328 proposal : str
329     The proposal number (e.g. ``'12345'``).
330
331 Returns
332 -----
333

```

```

330     proposal_dict : dict
331         A dictionary containing data used for the
332         ``/archive/<proposal>/`` webpage.
333     """
334
335     proposal_dict = _initialize_data_dict(proposal)
336     proposal_dict['visits'] = [os.path.basename(item).split('_')[0][4:6].upper() for item in proposal_dict['jpeg']]
337     proposal_dict['num_visits'] = len(set(proposal_dict['visits']))
338     proposal_dict = _get_metadata_from_database(proposal_dict)
339     proposal_dict = _get_buttons_dict(proposal_dict)
340     proposal_dict['viewlinks'] = ['/archive/{}/{}/.format(proposal_dict['proposal_id'], filename) for
341     filename in proposal_dict['filenames']]
342
343     return proposal_dict
344
345 def get_view_query_results_dict(query_results_dict):
346     """Return a dictionary containing data used for the
347     ``/database/results/`` webpage.
348
349     Parameters
350     -----
351     query_results_dict : dict
352         A dictionary containing the results of the query performed
353         through the ``/database/`` webpage, along with some additional
354         metadata.
355
356     Returns
357     -----
358     thumbnail_dict : dict
359         A dictionary containing data used for the ``/database/results/``
360         webpage.
361     """
362
363     query_results = query_results_dict['query_results']
364
365     thumbnail_dict = {}
366     thumbnail_dict['num_images'] = query_results_dict['num_results']
367     thumbnail_dict['rootnames'] = [item[3] for item in query_results]
368     thumbnail_dict['filenames'] = [item[4].split('_')[0] for item in query_results]
369     thumbnail_dict['detectors'] = [item[5] for item in query_results]
370     thumbnail_dict['expstarts'] = [item[6] for item in query_results]
371     thumbnail_dict['filter1s'] = [item[7] for item in query_results]
372     thumbnail_dict['filter2s'] = [item[8] for item in query_results]
373     thumbnail_dict['exptimes'] = [item[9] for item in query_results]
374     thumbnail_dict['targnames'] = [item[10] for item in query_results]
375     thumbnail_dict['proposal_ids'] = [item[11] for item in query_results]
376     thumbnail_dict['visits'] = [item[4:6] for item in thumbnail_dict['rootnames']]
377     thumbnail_dict = _get_buttons_dict(thumbnail_dict)
378     thumbnail_dict['thumbs'] = ['static/img-thumbnails/{}/{}_flt.thumb'.format(proposid, filename)
379         for proposid, filename in zip(thumbnail_dict['proposal_ids'], thumbnail_dict['filenames'])]
380     thumbnail_dict['viewlinks'] = ['/archive/{}/{}/.format(proposid, filename)
381         for proposid, filename in zip(thumbnail_dict['proposal_ids'], thumbnail_dict['filenames'])]
382
383     return thumbnail_dict

```

acsq1/website/form_options.py

```

1 """Provides a dictionary containing ``acsq1`` database query form data
2 for use by the ``acsq1`` web application.
3
4 Authors
5 -----
6
7 - Matthew Bourque
8 - Meredith Durbin
9
10 Use
11 ---
12
13 The dictionary contained in this module is intended to be imported
14 as such:
15 ::

16
17     from acsq1.website.form_options import FORM_OPTIONS
18
19 """

```

```

20 APERTURES = ['WFC', 'WFC-FIX', 'WFC1', 'WFC1-1K', 'WFC1-2K', 'WFC1-512',
21   'WFC1-CTE', 'WFC1-FIX', 'WFC1-IRAMP', 'WFC1-IRAMPQ', 'WFC1-MRAMP',
22   'WFC1-MRAMPQ', 'WFC1-POL0UV', 'WFC1-POL0V', 'WFC1-POL120UV', 'WFC1-POL120V',
23   'WFC1-POL60UV', 'WFC1-POL60V', 'WFC1A-1K', 'WFC1A-2K', 'WFC1A-512', 'WFC1B-1K',
24   'WFC1B-2K', 'WFC1B-512', 'WFC2', 'WFC2-2K', 'WFC2-FIX', 'WFC2-MRAMP',
25   'WFC2-MRAMP', 'WFC2-ORAMP', 'WFC2-ORAMPQ', 'WFC2C-1K', 'WFC2C-2K', 'WFC2C-512', 'WFC2D-1K',
26   'WFC2D-2K', 'WFC2D-512', 'WFCENTER', 'HRC', 'HRC-512', 'HRC-ACQ', 'HRC-CORON1.8',
27   'HRC-CORON3.0', 'HRC-FIX', 'HRC-OCCULT0.8', 'HRC-SUB1.8', 'SBC', 'SBC-FIX']
28 DETECTORS = ['WFC', 'HRC', 'SBC']
29 FILTER1S = ['F115LP', 'F122M', 'F125LP', 'F140LP', 'F150LP', 'F165LP',
30   'F475W', 'F502N', 'F550M', 'F555W', 'F606W', 'F625W', 'F658N', 'F775W',
31   'F850LP', 'F892N', 'G800L', 'POL0UV', 'POL60UV', 'POL120UV', 'PR110L',
32   'PR130L', 'CLEAR1L', 'CLEAR1S', 'BLOCK1', 'BLOCK3', 'BLOCK4', 'NotCmded']
33 FILTER2S = ['F220W', 'F250W', 'F330W', 'F344N', 'F435W', 'F660N', 'F814W',
34   'FR388N', 'FR423N', 'FR462N', 'FR459M', 'FR505N', 'FR551N', 'FR601N',
35   'FR647M', 'FR656N', 'FR716N', 'FR782N', 'FR853N', 'FR914M', 'FR931N',
36   'FR1016N', 'POL0V', 'POL60V', 'POL120V', 'PR200L', 'CLEAR2L', 'CLEAR2S',
37   'NotCmded']
38 IMAGETYPES = ['BIAS', 'DARK', 'FLAT', 'EXT']
39 OBSTYPES = ['IMAGING', 'SPECTROSCOPIC', 'CORONOGRAPHIC', 'INTERNAL']
40 OUTPUT_COLUMNS = [('rootname', 'Rootname'), ('detector', 'Detector'),
41   ('proposal_type', 'Proposal Type'), ('pr_inv_l', 'PI Last Name'),
42   ('pr_inv_f', 'PI First Name'), ('proposid', 'Proposal ID'), ('filter1', 'Filter1'),
43   ('filter2', 'Filter2'), ('aperture', 'Aperture'), ('expstart', 'Expstart'),
44   ('date_obs', 'Date of Observation'), ('time_obs', 'Time of Observation'),
45   ('targname', 'Target Name'), ('ra_targ', 'Target RA'), ('dec_targ', 'Target Dec'),
46   ('obstype', 'Observation Type'), ('obsmode', 'Observation Mode'),
47   ('subarray', 'Subarray'), ('imagetyp', 'Image Type'), ('asn_id', 'Association ID')]
48 OUTPUT_FORMAT = [('table', 'HTML table'), ('csv', 'CSV'), ('thumbnails', 'Thumbnails')]
49 PROPOSAL_TYPES = ['GO', 'GTO/ACS', 'CAL/ACS', 'SM3/ACS', 'SM3/ERO', 'SNAP',
50   'GO/PAR', 'GO/DD', 'GTO/COS', 'CAL/OTA', 'ENG/ACS', 'NASA', 'SM4/ACS',
51   'SM4/ERO', 'SM4/COS', 'CAL/WFC3', 'CAL/STIS']
52
53
54 FORM_OPTIONS = {}
55 FORM_OPTIONS['aperture'] = [(aperture, aperture) for aperture in APERTURES]
56 FORM_OPTIONS['detector'] = [(detector, detector) for detector in DETECTORS]
57 FORM_OPTIONS['filter1'] = [(filter1, filter1) for filter1 in FILTER1S]
58 FORM_OPTIONS['filter2'] = [(filter2, filter2) for filter2 in FILTER2S]
59 FORM_OPTIONS['imagetyp'] = [(imagetyp, imagetyp) for imagetyp in IMAGETYPES]
60 FORM_OPTIONS['obstype'] = [(obstype, obstype) for obstype in OBSTYPES]
61 FORM_OPTIONS['output_columns'] = OUTPUT_COLUMNS
62 FORM_OPTIONS['output_format'] = OUTPUT_FORMAT
63 FORM_OPTIONS['proposal_type'] = [(prop_type, prop_type) for prop_type in PROPOSAL_TYPES]
```

acsq1/website/query_form.py

```

1 """Contains class objects for building a query form for querying the
2 ``acsq1`` database through the ``acsq1`` web application.
3
4 Many of the class objects are subclasses or extensions from components
5 provided by the ``wtforms`` library. Hard coded data such as form
6 options are imported from the ``form_options`` module.
7
8 Authors
9 -----
10
11 - Matthew Bourque
12 - Meredith Durbin
13
14 Use
15 ---
16
17 This module is intended to be imported and used by the
18 ``acsq1_webapp`` module as such:
19 :::
20
21     from acsq1.website.query_form import get_query_form
22     query_form = get_query_form()
23
24 Dependencies
25 -----
26
27 - ``acsq1``
28 - ``wtforms``
29 - ``wtforms_components``
30 """

```

```

32 from wtforms import DateField
33 from wtforms import DecimalField
34 from wtforms import Form
35 from wtforms import FormField
36 from wtforms import RadioField
37 from wtforms import SelectField
38 from wtforms import SelectMultipleField
39 from wtforms import TextField
40 from wtforms import validators
41 from wtforms import widgets
42 from wtforms_components.fields import IntegerField
43
44 from acsql.website.form_options import FORM_OPTIONS
45
46
47 operator_form = SelectField('Operator',
48     [validators.Optional()],
49     choices=[('=', '='), ('<', '<'), ('>', '>'), ('between','between')],
50     default='=')
51
52
53 class CheckboxField(SelectMultipleField):
54     """Like a ``SelectField``, except displays a list of checkbox
55     buttons.
56
57     Parameters
58     -----
59     SelectMultipleField : obj
60         The ``SelectMultipleField`` object from ``wtforms``
61     """
62
63     widget = widgets.ListWidget(prefix_label=False)
64     option_widget = widgets.CheckboxInput()
65
66
67 class DateForm(Form):
68     """Creates a ``DateForm`` object that allows for date input in a
69     form field.
70
71     Parameters
72     -----
73     Form : obj
74         The ``Form`` object from ``wtforms``.
75     """
76
77     op = operator_form
78     val1 = DateField('Date Observed',
79         [validators.Optional()],
80         description='YYYY-MM-DD',
81         format='%Y-%m-%d')
82     val2 = DateField('dateobs2',
83         [validators.Optional()],
84         description='YYYY-MM-DD',
85         format='%Y-%m-%d')
86
87
88 class ExptimeForm(Form):
89     """Creates a ``ExptimeForm`` object that allows for ``exptime``
90     input in a form field.
91
92     Parameters
93     -----
94     Form : obj
95         The ``Form`` object from ``wtforms``.
96     """
97     op = operator_form
98     val1 = DecimalField('Exposure Time', [validators.Optional()])
99     val2 = DecimalField('exptime2', [validators.Optional()])
100
101
102 class MultiCheckboxField(SelectMultipleField):
103     """A multiple-select, except displays a list of checkboxes.
104
105     Parameters
106     -----
107     SelectMultipleField : obj
108         The ``SelectMultipleField`` object from ``wtforms``
```

```

109 """
110
111     widget = widgets.ListWidget(prefix_label=False)
112     option_widget = widgets.CheckboxInput()
113
114
115 def is_field_value(form, fieldname, value, negate=False):
116     """Helper function to check if the given field in the given form is
117     of a specified value.
118
119     Parameters
120     -----
121     form: obj
122         The form to test on
123     fieldname : str
124         The fieldname to test value against. If not found an Exception
125         is raised.
126     value : str
127         Value to test for.
128     negate : boolean
129         True/False to invert the result.
130 """
131
132     field = form._fields.get(fieldname)
133     if field is None:
134         raise Exception('Invalid field "%s"' % fieldname)
135     test = value == field.data
136     test = not test if negate else test
137
138     return test
139
140
141 class RequiredIf(validators.Required):
142     """Custom validator to enforce requires only if another field
143     matches a specified value. the ``negate`` allows for inverting
144     the result.
145
146     Parameters
147     -----
148     validators.Required : obj
149         The ``validators.Required`` object from ``wtforms``.
150 """
151
152     def __init__(self, other_fieldname, value, negate, *args, **kwargs):
153         self.other_fieldname = other_fieldname
154         self.negate = negate
155         self.value = value
156         super(RequiredIf, self).__init__(*args, **kwargs)
157
158     def __call__(self, form, field):
159         if is_field_value(form, self.other_fieldname, self.value, self.negate):
160             super(RequiredIf, self).__call__(form, field)
161
162
163 class QueryForm(Form):
164     """Form for querying the ``acsq1`` database.
165
166     Parameters
167     -----
168     Form : obj
169         The ``Form`` object from ``wtforms``.
170 """
171
172     rootname = TextField('Rootname',
173                         [validators.Optional()],
174                         description='Single rootname (IPPPSSOOT) or comma-separated list span6')
175     targname = TextField('Target Name',
176                         [validators.Optional()],
177                         description='ex. OMEGACEN, NGC-3603, IRAS05129+5128; single or comma-separated span6')
178     proposid = IntegerField('Proposal ID',
179                            [validators.Optional()],
180                            validators.NumberRange(min=8183, max=19999,
181                            message='Please enter a valid proposal ID')],
182                            description='span4')
183     date_obs = FormField(DateForm,
184                          'Date Observed',
185                          description='span4')

```

```

186     exptime = FormField(ExptimeForm,
187         'Exposure Time',
188         description='span4')
189     proposal_type = CheckboxField('Proposal Type',
190         [validators.Optional()],
191         description='span3',
192         choices=FORM_OPTIONS['proposal_type'])
193     detector = CheckboxField('Detector',
194         [validators.Optional()],
195         description='span3',
196         choices=FORM_OPTIONS['detector'])
197     obstype = CheckboxField('Observation Type',
198         [validators.Optional()],
199         description='span3',
200         choices=FORM_OPTIONS['obstype'])
201     aperture = SelectMultipleField('Aperture',
202         [validators.Optional()],
203         choices=FORM_OPTIONS['aperture'],
204         description='span3')
205     filter1 = SelectMultipleField('Filter1',
206         [validators.Optional()],
207         description='span3',
208         choices=FORM_OPTIONS['filter1'])
209     filter2 = SelectMultipleField('Filter2',
210         [validators.Optional()],
211         description='span3',
212         choices=FORM_OPTIONS['filter2'])
213     imagetyp = SelectMultipleField('Image Type',
214         [validators.Optional()],
215         choices=FORM_OPTIONS['imagetyp'],
216         description='span3')
217     pr_inv_l = TextField('PI Last Name',
218         [validators.Optional()],
219         description='Single or comma-separated span3')
220     pr_inv_f = TextField('PI First Name',
221         [validators.Optional()],
222         description='Single or comma-separated span3')
223     output_columns = MultiCheckboxField('Output Columns',
224         [RequiredIf('output_format', 'thumbnails', True,
225             message='Please select at least one output column.']),
226         choices=FORM_OPTIONS['output_columns'])
227     output_format = RadioField('Output Format',
228         [validators.Required(message='Please select an output format.']),
229         choices=FORM_OPTIONS['output_format'],
230         default='thumbnails', description='span3')
231
232
233 def get_query_form(Form):
234     """Return the ``QueryForm`` object that contains query form
235     components
236
237     Parameters
238     -----
239     Form : obj
240         A request form.
241
242     Returns
243     -----
244     query_form : obj
245         The ``QueryForm`` object, which contains the various components
246         to build the ACS Database query form.
247
248
249     query_form = QueryForm(Form)
250
251     return query_form

```

acsq1/website/query_lib.py

```

1 """Contains various functions to support the ``/database/`` webpage of
2 the ``acsq1`` web application.
3
4 Functions include those that parse, build, validate, and return
5 ``SQLAlchemy`` ``query`` objects in order to perform a database query
6 through the web application.
7
8 Authors
9 -----
10
11 - Matthew Bourque
12 - Meredith Durbin
13
14 Use
15 ---
16
17 This module is intended to be imported and used by the
18 ``acsq1_webapp`` module as such:
19 :::
20
21     from query_lib import generate_csv
22     from query_lib import get_query_results
23
24     generate_csv(output_columns, results)
25     get_query_results(query_form_dict)
26
27 Dependencies
28 -----
29
30 - ``acsq1``
31 - ``sqlalchemy``
32 """
33
34 from sqlalchemy import create_engine
35 from sqlalchemy import literal_column
36 from sqlalchemy import or_
37 from sqlalchemy.orm import sessionmaker
38
39 from acsq1.database.database_interface import Master
40 from acsq1.database.database_interface import WFC_raw_0
41 from acsq1.database.database_interface import HRC_raw_0
42 from acsq1.database.database_interface import SBC_raw_0
43 from acsq1.utils.utils import SETTINGS
44
45
46 def _apply_query_filter(table, key, values, query):
47     """Apply a filter to the given ``query`` based on the ``table``,
48     ``key``, and ``value``.
49
50     Parameters
51     -----
52     table : obj
53         The ``SQLAlchemy`` table object associated with the table to
54         apply the filter to.
55     key : str
56         The keyword for the column to filter on.
57     values : obj
58         The requested values of the ``key``.
59     query : obj
60         The ``SQLAlchemy`` ``query`` object to perform the filtering
61         on.
62
63     Returns
64     -----
65     query : obj
66         The ``SQLAlchemy`` ``query`` object with filter applied.
67 """
68
69 # Fields that accept comma-separated lists and wildcards
70 csv_keys = ['rootname', 'targname', 'pr_inv_l', 'pr_inv_f']
71
72 # Fields that allow operators
73 operator_keys = ['date_obs', 'exptime']
74
75 # Parse the key/value pairs for comma-separated values
76 if key in csv_keys:

```

```

77     parsed_value = values[0].replace(' ', '').split(',')
78     parsed_value = [item.replace('*', '%') for item in parsed_value]
79     conditions = [getattr(table, key).like(val) for val in parsed_value]
80     query = query.filter(or_(*conditions))
81
82     # Parse the key/value pairs for operator keys
83     elif key in operator_keys:
84         if values['op'] == 'between':
85             if len(values) == 3:
86                 if float(values['val1'].replace('-', '')) < float(values['val2'].replace('-', '')):
87                     query = query.filter(getattr(table, key).between(values['val1'], values['val2']))
88                 elif float(values['val1'].replace('-', '')) > float(values['val2'].replace('-', '')):
89                     query = query.filter(getattr(table, key).between(values['val2'], values['val1']))
90                 elif float(values['val1'].replace('-', '')) == float(values['val2'].replace('-', '')):
91                     raise ValueError('Values submitted for field "{}" must be different.'.format(key))
92             else:
93                 raise ValueError('Invalid values submitted for field "{}".format(key)')
94         else:
95             query = query.filter(getattr(table, key).op('=')(values['val1']))
96     else:
97         query = query.filter(getattr(table, key).op(values['op'])(values['val1']))
98
99     # Else the filtering is straightforward
100    query = query.filter(getattr(table, key).in_(values))
101
102    return query
103
104
105 def _build_queries(output_columns):
106     """Builds queries of appropriate tables and columns
107
108     Parameters
109     -----
110     output_columns : list
111         List of columns desired for query output
112
113     Returns
114     -----
115     wfc_query : obj
116         Query object for requested columns in WFC database tables.
117     hrc_query : obj
118         Query object for requested columns in HRC database tables.
119     sbc_query : obj
120         Query object for requested columns in SBC database tables.
121     """
122
123     # Determine which columns belong to which tables
124     master_cols = [getattr(Master, col) for col in output_columns if hasattr(Master, col)]
125     wfc_cols = [getattr(WFC_raw_0, col) for col in output_columns if hasattr(WFC_raw_0, col)]
126     hrc_cols = [getattr(HRC_raw_0, col) for col in output_columns if hasattr(HRC_raw_0, col)]
127     sbc_cols = [getattr(SBC_raw_0, col) for col in output_columns if hasattr(SBC_raw_0, col)]
128
129     # Determine which columns are unique to a specific table
130     wfc_only = [col for col in output_columns if hasattr(WFC_raw_0, col)
131                  and not hasattr(Master, col)
132                  and not hasattr(HRC_raw_0, col)
133                  and not hasattr(SBC_raw_0, col)]
134     hrc_only = [col for col in output_columns if hasattr(HRC_raw_0, col)
135                  and not hasattr(Master, col)
136                  and not hasattr(WFC_raw_0, col)
137                  and not hasattr(SBC_raw_0, col)]
138     sbc_only = [col for col in output_columns if hasattr(SBC_raw_0, col)
139                  and not hasattr(Master, col)
140                  and not hasattr(WFC_raw_0, col)
141                  and not hasattr(HRC_raw_0, col)]
142
143     # Combine columns amongst tables
144     master_wfc = master_cols + wfc_cols + [literal_column('---').label(col) for col in hrc_only] + \
145                 [literal_column('---').label(col) for col in sbc_only]
146     master_hrc = master_cols + hrc_cols + [literal_column('---').label(col) for col in wfc_only] + \
147                 [literal_column('---').label(col) for col in sbc_only]
148     master_sbc = master_cols + sbc_cols + [literal_column('---').label(col) for col in wfc_only] + \
149                 [literal_column('---').label(col) for col in hrc_only]
150
151     if len(master_cols) == 0:
152         # For WFC queries

```

```

154     if len(wfc_cols) == 0:
155         wfc_query = False
156     else:
157         session = _get_session()
158         wfc_query = session.query(*master_wfc)
159         session.close()
160
161     # For HRC queries
162     if len(hrc_cols) == 0:
163         hrc_query = False
164     else:
165         session = _get_session()
166         hrc_query = session.query(*master_hrc)
167         session.close()
168
169     # For SBC queries
170     if len(sbc_cols) == 0:
171         sbc_query = False
172     else:
173         session = _get_session()
174         sbc_query = session.query(*master_sbc)
175         session.close()
176
177 else:
178     session = _get_session()
179     wfc_query = session.query(*master_wfc).join(WFC_raw_0)
180     hrc_query = session.query(*master_hrc).join(HRC_raw_0)
181     sbc_query = session.query(*master_sbc).join(SBC_raw_0)
182
183 return wfc_query, hrc_query, sbc_query
184
185
186 def _convert_query_form_dict(query_form_dict):
187     """Converts raw output from ``form.to_dict()`` to a format that is
188     more useable for ``acsqll`` database queries.
189
190     Parameters
191     -----
192     query_form_dict : dict
193         The dictionary returned by ``form.to_dict()``.
194
195     Returns
196     -----
197     query_form_dict : dict
198         A new dictionary with blank entries removed and operator key
199         entries reformatted.
200     """
201
202     # Keys that allow operators (e.g. greater than)
203     operator_keys = ['date_obs', 'exptime']
204
205     # Remove blank entries from form data
206     query_form_dict = {key: value for key, value in list(query_form_dict.items()) if value != ['']}
207
208     # Combine data returned from fields with operator dropdowns
209     for operator_key in operator_keys:
210         operator_dict = {}
211         for key, value in list(query_form_dict.items()):
212             if key == operator_key + '-op':
213                 operator_dict['op'] = value[0]
214             elif key == operator_key + '-val1':
215                 operator_dict['val1'] = value[0]
216             elif key == operator_key + '-val2':
217                 operator_dict['val2'] = value[0]
218         if len(operator_dict) > 1:
219             query_form_dict[operator_key] = operator_dict
220
221     return query_form_dict
222
223
224 def generate_csv(output_columns, results):
225     """Create a CSV file of the database query ouput.
226
227     Parameters
228     -----
229     output_columns : list
230         A list of columns desired for the output file.

```

```

231     results : list
232         A list of results from the database query
233
234
235     header = ','.join(output_columns) + '\n'
236     yield header
237
238     for result in results:
239         if len(result) == 1:
240             yield str(result[0]) + '\n'
241         else:
242             yield ','.join(map(str, result)) + '\n'
243
244
245 def get_query_results(query_form_dict):
246     """Returns a dictionary with the results of the requested query
247     along with some additional metadata. Calls on several internal
248     functions to build and perform the query in order to abstract
249     out its complexity.
250
251     Parameters
252     -----
253     query_form_dict : dict
254         A dictionary containing information about the requested query,
255         such as the ``output_format``, ``output_columns`` and the
256         requested values.
257
258     Returns
259     -----
260     query_results_dict : dict
261         A dictionary containing the query results as well as some
262         metadata such as ``output_format`` and number of results.
263
264
265     # Determine output format
266     output_format = query_form_dict.pop('output_format')
267     if output_format == ['thumbnails']:
268         output_columns = ['rootname', 'filename', 'detector',
269                           'proposal_type', 'expstart', 'filter1',
270                           'filter2', 'exptime', 'targname', 'proposid']
271     if 'output_columns' in query_form_dict:
272         query_form_dict.pop('output_format', None)
273     else:
274         output_columns = query_form_dict.pop('output_columns')
275
276     # Remove blank entries from form data
277     query_form_dict = _convert_query_form_dict(query_form_dict)
278
279     # Build the query
280     wfc_query, hrc_query, sbc_query = _build_queries(output_columns)
281
282     # Perform filtering on the query
283     for key, value in list(query_form_dict.items()):
284         if hasattr(Master, key):
285             if wfc_query:
286                 wfc_query = _apply_query_filter(Master, key, value, wfc_query)
287             if hrc_query:
288                 hrc_query = _apply_query_filter(Master, key, value, hrc_query)
289             if sbc_query:
290                 sbc_query = _apply_query_filter(Master, key, value, sbc_query)
291         if wfc_query and hasattr(WFC_raw_0, key) and not hasattr(Master, key):
292             wfc_query = _apply_query_filter(WFC_raw_0, key, value, wfc_query)
293         if hrc_query and hasattr(HRC_raw_0, key) and not hasattr(Master, key):
294             hrc_query = _apply_query_filter(HRC_raw_0, key, value, hrc_query)
295         if sbc_query and hasattr(SBC_raw_0, key) and not hasattr(Master, key):
296             sbc_query = _apply_query_filter(SBC_raw_0, key, value, sbc_query)
297
298     # Combine the results
299     query = _merge_query(wfc_query, hrc_query, sbc_query)
300
301     # Perform the query
302     if query:
303         query_results = query.all()
304         num_results = len(query_results)
305     else:
306         query_results = False
307         num_results = 0

```

```

308     # Put results in a dictionary
309     query_results_dict = {}
310     query_results_dict['num_results'] = num_results
311     query_results_dict['query_results'] = query_results
312     query_results_dict['output_format'] = output_format
313     query_results_dict['output_columns'] = output_columns
314
315
316     return query_results_dict
317
318
319 def _get_session():
320     """Return a ``session`` object to be used as a connection to the
321     ``acsqll`` database.
322
323     Returns
324     -----
325     session : obj
326         A ``SQLAlchemy`` ``session`` object which serves as a
327         connection to the ``acsqll`` database.
328
329     """
330
331     engine = create_engine(SETTINGS['connection_string'], echo=False, pool_timeout=100000)
332     Session = sessionmaker(bind=engine)
333     session = Session()
334
335
336     return session
337
338
339 def _merge_query(wfc_query, hrc_query, sbc_query):
340     """Merge the results from the queries from each table
341
342     Parameters
343     -----
344     wfc_query : obj
345         The ``SQLAlchemy`` ``query`` object for request columns of the
346         WFC table.
347     hrc_query : obj
348         The ``SQLAlchemy`` ``query`` object for request columns of the
349         HRC table.
350     sbc_query : obj
351         The ``SQLAlchemy`` ``query`` object for request columns of the
352         SBC table.
353
354     Returns
355     -----
356     query : obj
357         The ``SQLAlchemy`` ``query`` object with merging applied.
358
359     """
360
361     # Turn off the query if the table is not needed
362     if wfc_query:
363         if str(wfc_query.statement).find('WHERE') == -1:
364             wfc_query = False
365     if hrc_query:
366         if str(hrc_query.statement).find('WHERE') == -1:
367             hrc_query = False
368     if sbc_query:
369         if str(sbc_query.statement).find('WHERE') == -1:
370             sbc_query = False
371
372     # If combination needed
373     if wfc_query and hrc_query and sbc_query:
374         query = wfc_query.union_all(hrc_query, sbc_query)
375     elif wfc_query and not hrc_query and sbc_query:
376         query = wfc_query.union_all(sbc_query)
377     elif wfc_query and hrc_query and not sbc_query:
378         query = wfc_query.union_all(hrc_query)
379     elif not wfc_query and hrc_query and sbc_query:
380         query = hrc_query.union_all(sbc_query)
381
382     # If no combination needed
383     elif wfc_query and not hrc_query and not sbc_query:
384         query = wfc_query
385     elif not wfc_query and hrc_query and not sbc_query:
386         query = hrc_query
387     elif not wfc_query and not hrc_query and sbc_query:
388

```

```
385         query = sbc_query
386
387     else:
388         query = None
389
390     return query
```

acsqsl/website/templates/404.html

```
1 {# extends "base.html" #}
2 {# block content #}
3
4 <h3>Oops! The page you're looking for doesn't seem to exist.</h3>
5
6 <center></center>
7
8 {# endblock #}
```

acsq1/website/templates/500.html

```
1  {% extends "base.html" %}  
2  {% block content %}  
3  
4 <h3>Oops! Something went wrong.</h3>  
5  
6 <pre>{{trace_html|safe}}</pre>  
7  
8 {% endblock %}
```

acsqsl/website/templates/_formhelpers.html

```
1  {%- macro render_field(field) %}  
2  {%- if field.type == 'MultiCheckboxField' %}  
3  <div class="span12">  
4      <h5>{{ field.label.text }}</h5>  
5      {{ field(id=field.name, **kwargs)|safe }}  
6  </div>  
7  {%- if field.label.text.endswith('s') %}  
8  <div class="span12">  
9      <button type="button" id="check_all_{{field.name}}" class="btn">Select all {{field.label.text.lower()}}  
10     }</button>  
11     <button type="button" id="uncheck_all_{{field.name}}" class="btn">Deselect all {{field.label.text.lower()}}</button>  
12 {%- else %}  
13 <div class="span12">  
14     <button type="button" id="check_all_{{field.name}}" class="btn">Select all {{field.label.text.lower()}}s</button>  
15     <button type="button" id="uncheck_all_{{field.name}}" class="btn">Deselect all {{field.label.text.lower()}}s</button>  
16 {%- endif %}  
17 {%- elif field.type == 'SelectMultipleField' %}  
18 <div class="{{ field.description[-5:] }} formfield">  
19     <h5>{{ field.label.text }}</h5>  
20     {{ field(style="width:100%;", placeholder=field.description[:-6], **kwargs)|safe }}  
21  
22 {%- else %}  
23 <div class="{{ field.description[-5:] }} formfield">  
24     <h5>{{ field.label.text }}</h5>  
25     {{ field(style="width:calc(100% - 1em);width:-moz-calc(100% - 1em);  
26         width:-webkit-calc(100% - 1em);width:-o-calc(100% - 1em);width:-ms-calc(100% - 1em);",  
27         placeholder=field.description[:-6], **kwargs)|safe }}  
28 {%- endif %}  
29 </div>  
30 {%- endmacro %}  
31  
32 {%- macro render_formfield(field) %}  
33 <div class="{{ field.description[-5:] }} formfield">  
34     <h5>{{ field.label.text }}</h5>  
35     {%- for subfield in field %}  
36         {%- if subfield.name.endswith('op') %}  
37             {{ subfield(style="width:4em;display:inline;", **kwargs)|safe }}&nbsp;  
38         {%- elif subfield.name.endswith('val1') %}  
39             {{ subfield(style="width:calc(100% - 5.75em);display:inline;", placeholder=subfield.description, **  
40             kwargs)|safe }}  
41             {%- elif subfield.name.endswith('val2') %}  
42                 {{ subfield(style="width:calc(100% - 5.75em);display:inline;float:right;margin-right:3px;display:none  
43             ;", placeholder=subfield.description, **kwargs)|safe }}  
44         {%- endif %}
```

```

43     { % endfor %}
44 </div>
45 { % endmacro %}

acsq1/website/templates/archive.html

1 {%- extends "base.html" %}
2 {%- block content %}
3
4 <div class="row">
5     <div class="span12">
6         <h3>ACS Archive</h3>
7         <table class="table">
8             {%- for row in proposal_array %}
9                 <tr>
10                     {%- for proposal in row %}
11                         <td><a href="/archive/{{proposal}}/" target="_blank">{{proposal}}</a></td>
12                     {%- endfor %}
13                 </tr>
14             {%- endfor %}
15         </table>
16     </div>
17 </div>
18
19 {%- endblock %}

```

acsq1/website/templates/base.html

```

1 <!DOCTYPE html>
2 <html class="no-js">
3     <head>
4         <meta charset="utf-8">
5         <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
6         <title>ACS Quicklook</title>
7         <meta name="description" content="">
8         <meta name="viewport" content="width=device-width">
9         <link rel="stylesheet" href="/static/css/bootstrap.min.css">
10        <link rel="icon" type="image/png" href="/static/img/favicon.png">
11
12        <style>
13            body {
14                padding-top: 60px;
15                padding-bottom: 40px;
16            }
17            @media (min-width:980px) and (max-width:1199px) {
18                .no-overflow {
19                    width:11.5vw;
20                    overflow:hidden;
21                    white-space:nowrap;
22                    text-overflow:ellipsis;
23                }
24            }
25        </style>
26
27        <link rel="stylesheet" href="/static/css/bootstrap-responsive.min.css">
28
29        <script src="../../static/js/modernizr-2.6.2.min.js"></script>
30        <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js"></script>
31        <script>window.jQuery || document.write('<script src="/static/js/jquery-1.10.1.min.js"></script>')
32    </script>
33
34    </head>
35    <body>
36        <div class="navbar navbar-inverse navbar-fixed-top">
37            <div class="navbar-inner">
38                <div class="container">
39                    <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
40                        <span class="icon-bar"></span>
41                        <span class="icon-bar"></span>
42                        <span class="icon-bar"></span>
43                    </a>
44                    <a class="brand pull-left no-overflow" href="/"> ACS Quicklook</a>
46                    <div class="nav-collapse collapse">
47                        <ul class="nav pull-right">
48                            <li><a href="/database/"><font size="4">Database</font></a></li>
49                            <li><a href="/archive/"><font size="4">Archive</font></a></li>
50                            <li><a href="https://github.com/spacetelescope/acsq1" target="_blank"><font
51 size="4">GitHub</font></a></li>

```

```

49             <li><a href="http://acsq1.readthedocs.io" target="_blank"><font size="4">
50 Documentation</font></a></li>
51         </ul>
52     </div>
53 </div>
54 </div>
55
56 <div class="container">
57
58     {%
59     block content %}
60     {%
61     endblock %}
62
63     <hr>
64
65     <footer>
66         <p>Space Telescope Science Institute</p>
67     </footer>
68
69 </div>
70
71 <script src="/static/js/bootstrap.min.js"></script>
72 <script src="/static/js/plugins.js"></script>
73 <script src="/static/js/main.js"></script>
74 </body>
75 </html>

```

acsq1/website/templates/database.html

```

1  {% extends "base.html" %}
2  {% block content %}
3
4  <link rel="stylesheet" href="/static/css/database.css">
5
6 <div class="row">
7     <div class="span12">
8         <h3>ACS Database Query Form</h3>
9         <p>This form enables users to query the acsq1 database of all ACS images.
10            Only a subset of the full database is represented here; if the options available do not meet
11            your needs,
12            you may enter a hard-coded MySQL command below.
13        </p>
14    </div>
15
16    {% from "_formhelpers.html" import render_field, render_formfield %}
17
18    <form id="queryform" method="get" action="results" target="_blank">
19
20        {% for field in form %}
21            {% if field.type == 'FormField' %}
22                {{ render_formfield(field) }}
23            {% else %}
24                {{ render_field(field) }}
25            {% endif %}
26        {% endfor %}
27
28        <div class="span12" style="margin-top:2em;">
29            <center>
30                <button type="submit" value="Submit" class="btn btn-lg btn-primary">Submit</button>&ampnbsp&
nbsp;&nbsp;
31                <button type="reset" value="Reset" class="btn btn-lg btn-danger">Reset</button>
32            </center>
33        </div>
34    </form>
35
36    {% for field in form %}
37        {% if field.type == 'MultiCheckboxField' %}
38            <script>
39                $('#check_all_{{field.name}}').click(function() {
40                    $('input[name={{field.name}}]').prop('checked', true);
41                });
42                $('#uncheck_all_{{field.name}}').click(function() {
43                    $('input[name={{field.name}}]').prop('checked', false);
44                });
45            </script>
46        {% elif field.type == 'FormField' %}
47            <script>

```

```

48     $'#{field.name}-op').change(function() {
49         if ($(this).val() == 'between') {
50             $'#{field.name}-val2').show();
51         } else {
52             $'#{field.name}-val2').hide();
53         }
54     });
55     </script>
56 {%
57 % endfor %}
58 {%
59 % endblock %}

```

acsq1/website/templates/database_error.html

```

1  {% extends "base.html" %}
2  {% block content %}
3
4  <div class="row">
5      <div class="span12">
6          <h4>There was a problem with your request. Please try again.</h4>
7      </div>
8      <div class="span12">
9
10         {% if msg %}
11             {{ msg }}
12         {% else %}
13             <ul>
14                 {% for field, errors in form.errors.items() %}
15                     <li>
16                         {% if form[field].type == 'FormField' %}
17                             {{ form[field].label.text }}:
18                             {% for err in errors.values() %}
19                                 {{ ', '.join(err) }}
20                             {% endfor %}
21                         {% else %}
22                             {{ form[field].label.text }}: {{ ', '.join(errors) }}
23                         {% endif %}
24                     </li>
25                 {% endfor %}
26             {% endif %}
27
28         </div>
29
30     </div>
31
32 {%
33 % endblock %}

```

acsq1/website/templates/database_table.html

```

1  {% extends "base.html" %}
2  {% block content %}
3
4  <link rel="stylesheet" href="//cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css">
5  <link rel="stylesheet" href="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables_
   bootstrap.css">
6  <link rel="stylesheet" href="/static/css/database.css">
7  <link rel="stylesheet" href="/static/css/loader.css">
8
9  <style type="text/css">
10 div#content { display: none; }
11 div#loading { text-align:center; z-index: 1000; }
12 </style>
13
14 <div class="row">
15
16     <!-- Print how many results there are -->
17     {% if results %}
18         <div class="span12">
19             {% if num_results == 0 %}
20                 <h4>The query returned no results.</h4>
21             {% elif num_results == 1 %}
22                 <h4>The query returned 1 result.</h4>
23             {% else %}
24                 <h4>The query returned {{ num_results }} results.</h4>
25             {% endif %}
26
27             <!-- If there are results to show -->
28             {% if num_results > 0 %}
29

```

```

30     <!-- Loading animation -->
31     <div id="loading">
32       <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
33         <div class="cssload-cssload-loader-line-wrap-wrap">
34           <div class="cssload-loader-line-wrap"></div>
35         </div>
36         <div class="cssload-cssload-loader-line-wrap-wrap">
37           <div class="cssload-loader-line-wrap"></div>
38         </div>
39         <div class="cssload-cssload-loader-line-wrap-wrap">
40           <div class="cssload-loader-line-wrap"></div>
41         </div>
42         <div class="cssload-cssload-loader-line-wrap-wrap">
43           <div class="cssload-loader-line-wrap"></div>
44         </div>
45         <div class="cssload-cssload-loader-line-wrap-wrap">
46           <div class="cssload-loader-line-wrap"></div>
47         </div>
48       </div>
49       <br>
50       Loading table...
51     </div>
52
53     <!-- HTML table -->
54     <div id="content">
55       <hr>
56       <table class="table hover" id="results">
57         <thead>
58           {%
59             for col in output_columns %}
60             <th>{{ col }}</th>
61           {% endfor %}
62         </thead>
63         <tbody>
64           {%
65             for line in results %}
66             <tr>
67               {%
68                 for value in line %}
69                 <td>{{ value }}</td>
70               {% endfor %}
71             </tr>
72           {% endfor %}
73         </tbody>
74       </table>
75     </div>
76   {% endif %}
77
78 </div>
79
80 <script src="//cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js"></script>
81 <script src="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.bootstrap.js"></script>
82   >
83
84 <script>
85 $(document).ready(function() {
86   $('#results').DataTable( {
87     "lengthMenu": [[10, 25, 50, 100, 500, 1000, -1], [10, 25, 50, 100, 500, 1000, "All"]],
88     stateSave: true,
89     "language": {
90       "lengthMenu": "Display _MENU_ results per page",
91       "zeroRecords": "Nothing found",
92       "info": "Showing page _PAGE_ of _PAGES_",
93       "infoEmpty": "No results available",
94       "infoFiltered": "(filtered from _MAX_ total results)"
95     }
96   });
97
98 function preloader() {
99   document.getElementById("loading").style.display = "none";
100  document.getElementById("content").style.display = "block";
101 }
102 window.onload = preloader;
103
104 </script>
105

```

```
106 {%- endblock %}
```

acsq1/website/templates/header.html

```
1 {%- extends "base.html" %} 
2 {%- block content %} 
3 
4 <div class="row">
5     <div class="span12">
6         <h3>{{header_dict.filename}} {{header_dict.fits_type}} header</h3>
7         {% if header_dict.header %}
8             {% for ext, head in header_dict.header.items() if ext.startswith('Extension') %}
9                 <h4>{{ ext }}</h4>
10                <p>{{ head|safe }}</p>
11            {% endfor %}
12        {% else %}
13            <p>There is no header to display.</p>
14        {% endif %}
15    </div>
16 </div>
17 
18 {%- endblock %}
```

acsq1/website/templates/index.html

```
1 {%- extends "base.html" %} 
2 {%- block content %} 
3 
4 <link rel="stylesheet" href="//cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css">
5 <link rel="stylesheet" href="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.
6 bootstrap.css">
7 
8 <div class="row">
9     <div class="span6">
10        <h2>Database</h2>
11        <p>Query the acsql Database</p>
12        <p><a class="btn" href="/database/">Query form &raquo;</a></p>
13    </div>
14    <div class="span6">
15        <h2>Archive</h2>
16        <p>Links to public ACS archival data.</p>
17        <p><a class="btn" href="/archive/">Archive &raquo;</a>&ampnbsp</p>
18    </div>
19 
20 <script src="//cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js"></script>
21 <script src="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.bootstrap.js"></script>
22 >
23 
24 <script>
25 $(document).ready(function() {
26     $('#scriptstatus').DataTable( {
27         "paging": false,
28         "bFilter": false,
29         stateSave: true
30     } );
31 });
32 </script>
33 {%- endblock %}
```

acsq1/website/templates/view_image.html

```
1 {%- extends "base.html" %} 
2 {%- block content %} 
3 
4 <!-- Image info -->
5 <div class="row">
6 
7     <!-- Filename title -->
8     <div class="span12">
9         <h3>{{image_dict.filename}}</h3>
10    </div>
11 
12    <!-- Image metadata -->
13    <div class="span6">
14        <p>
15            <b>Proposal</b> {{image_dict.proposal_id}}</a>:</b> <i>{{image_dict.proposal_title}}</i><br>
16            Image {{image_dict.page}} of {{image_dict.num_images}}<br>
17        </p>
18    </div>
```

```

18     <!-- FITS download links -->
19     {% if (not image_dict.fits_links) or (image_dict.fits_links|length == 0) %}
20         No FITS downloads available yet
21     {% else %}
22         Downloads:
23         {% for key, link in image_dict.fits_links.items() %}
24             <a href="{{link}} target="_blank" download>{{key}}</a>
25         {% endfor %}
26     {% endif %}

27     <!-- Additional information links -->
28     <br>
29     View all in <a href="/archive/{{image_dict.proposal_id}}/">proposal</a>; view <a href="{{image_dict.view_url}}/headers/">headers</a>; view in <a href="{{image_dict.view_url}}/js9/">JS9</a><br>
30     {% if image_dict.available_jpegs %}
31         View JPEG for
32         {% for key, link in image_dict.available_jpegs.items() %}
33             <a href="/archive/{{image_dict.proposal_id}}/{{image_dict.filename}}/{{key}}"> {{key}}</a>
34         {% endfor %}
35     {% endif %}
36     </p>
37 </div>
38 <div class="span3">
39     <p>
40         <b>Expstart:</b> {{image_dict.expstart}}<br>
41         <b>Filter:</b> {{image_dict.filter1}}/{{image_dict.filter2}}<br>
42         <b>Aperture:</b> {{image_dict.aperture}}<br>
43         <b>Exposure time:</b> {{image_dict.exptime}}<br>
44     </p>
45 </div>
46 <div class="span3">
47     <p>
48         <b>Expflag:</b> {{image_dict.expflag}}, Quality: {{image_dict.quality}}<br>
49         <b>Coordinates:</b> {{image_dict.ra}}, {{image_dict.dec}}<br>
50         <b>Target:</b> {{image_dict.targname}}<br>
51         <b>PI:</b> {{image_dict.pi_first_name}} {{image_dict.pi_last_name}}<br>
52     </p>
53 </div>
54 </div>

55 <!-- Other available JPEGs -->
56
57
58
59 <!-- Image -->
60 <div class="row">
61     <div class="fleximage">
62         {% if image_dict.image %}
63             <div id="wrapper" style="width:100%; text-align:center">
64                 
65             </div>
66         {% else %}
67             <br><br>
68             <center><strong><p>There is no JPEG to display for FITS type {{image_dict.fits_type}}</p></strong></center>
69         {% endif %}
70     </div>
71 </div>

72 <div style="height:1em;"></div>
73
74 <!-- Enable left and right keystrokes -->
75 <script type="text/javascript">
76     {% if not image_dict.last %}
77         $("body").keydown(function(e) {
78             if(e.keyCode == 39) { // right
79                 window.location = "{{ url_for('view_image', **image_dict.next) }}";
80             }
81         });
82     {% endif %}
83     {% if not image_dict.first %}
84         $("body").keydown(function(e) {
85             if(e.keyCode == 37) { // left
86                 window.location = "{{ url_for('view_image', **image_dict.prev) }}";
87             }
88         });
89     {% endif %}
90
91 
```

```

92 </script>
93
94 {%- endblock %}

acsq1/website/templates/view_proposal.html

1  {% extends "base.html" %}
2  {% block content %}
3
4  <link rel="stylesheet" href="/static/css/loader.css">
5  <link rel="stylesheet" href="/static/css/view_proposal.css">
6
7  <div class="row">
8      <div class="span12">
9
10     <!-- Header -->
11     {% if proposal_dict.num_images %}
12         <h3>Proposal {{proposal_dict.proposal_id}} {% if proposal_dict.proposal_title %}<small>{{proposal_dict.proposal_title}}{% endif %}</small></h3>
13     {% if proposal_dict.buttons %}
14         <form id="filtering" style="margin-bottom:0;">
15
16             <!-- Filter -->
17             <div id="filterOptions" class="container">
18                 <div class="form-group">
19                     {% if proposal_dict.cycle %}Cycle {{proposal_dict.cycle}}, {{proposal_dict.schedule}}<br>{% endif %}
20                     {{proposal_dict.num_images}} images{{% if proposal_dict.num_visits %}, {{proposal_dict.num_visits}} visits{{% endif %}}
21                     <br>
22                     <a href="{{proposal_dict.status_page}}" target="_blank">Proposal info</a>
23                 </div>
24                 {% for key,buttonlist in proposal_dict.buttons.items() %}
25                     <div class="form-group">
26                         <label for="{{key}}>Show only {{key}}:</label>
27                         <select class="form-control" id="{{key}}>
28                             <option value="all">All {{key}}s</option>
29                             {% for b in buttonlist %}
30                                 <option value="{{b}}>{{b}}</option>
31                             {% endfor %}
32                         </select>
33                     </div>
34                     {% endfor %}
35                 </div>
36
37             <!-- Sort By -->
38             <div class="row" id="sortOptions">
39                 <div class="span6">
40                     Sort by:&nbsp;&nbsp;
41                     <label class="radio inline"><input type="radio" name="sort" id="unsort" checked>default
42                 </label>
43                     <label class="radio inline"><input type="radio" name="sort" id="sort-expstart">expstart
44                 </label>
45                     <label class="radio inline"><input type="radio" name="sort" id="sort-exptime">exptime</label>
46                 </div>
47                 <div class="span5 text-right">
48                     {% if proposal_dict.buttons|length > 2 %}
49                         Filter logic:&nbsp;&nbsp;
50                         <label class="radio inline"><input type="radio" name="logic" value="and" checked>"AND"</label>
51                         <label class="radio inline"><input type="radio" name="logic" value="or">"OR"</label>
52                     {% endif %}
53                 </div>
54                 <div class="span1 text-right">
55                     <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
56                 </div>
57             </div>
58         </form>
59     {% endif %}
60
61     <!-- Loading animation -->
62     <div id="loading">
63         <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
64             <div class="cssload-cssload-loader-line-wrap-wrap">
65                 <div class="cssload-loader-line-wrap"></div>
66             </div>
67             <div class="cssload-cssload-loader-line-wrap-wrap">

```

```

66         <div class="cssload-loader-line-wrap"></div>
67     </div>
68     <div class="cssload-cssload-loader-line-wrap-wrap">
69         <div class="cssload-loader-line-wrap"></div>
70     </div>
71     <div class="cssload-cssload-loader-line-wrap-wrap">
72         <div class="cssload-loader-line-wrap"></div>
73     </div>
74     <div class="cssload-cssload-loader-line-wrap-wrap">
75         <div class="cssload-loader-line-wrap"></div>
76     </div>
77 </div>
78 <br>
79 Loading thumbnails...
80 </div>

82     <!-- Thumbnails -->
83     <div id="onload">
84         <div id="thumbnail-array">
85             {% for i in range(proposal_dict.num_images) %}
86                 <div class="thumb" page="{{i+1}}" detector="{{proposal_dict.detectors[i]}} filter1=
87 {{proposal_dict.filter1s[i]}} filter2="{{proposal_dict.filter2s[i]}} expstart="{{proposal_dict.
88 expstarts[i]}} exptime="{{proposal_dict.exptimes[i]}} targname="{{proposal_dict.targnames[i]}}"
89 proposid="{{proposal_dict.proposal_id}} visit="{{proposal_dict.visits[i]}} style="background:url
90 (../../{{proposal_dict.thumbs[i]}}) no-repeat;">
91                     <div class="overlay">
92                         <strong>{{proposal_dict.filenames[i]}}</strong><br><font size="0.5">
93                         <b>Detector:</b> {{proposal_dict.detectors[i]}}<br>
94                         <b>Filter:</b> {{proposal_dict.filter1s[i]}} / {{proposal_dict.filter2s[i]
95 }}<br>
96                         <b>Exptime:</b> {{proposal_dict.exptimes[i]}}<br>
97                         <b>Target:</b> {{proposal_dict.targnames[i]}}<br></font>
98                     </div>
99                     <a href="{{proposal_dict.viewlinks[i]}}" target="_blank">
100                         
101                     </a>
102                 </div>
103             {% endfor %}
104         </div>
105     {% else %}
106         <p>
107             <h5>There are no images to display.</h5>
108         </p>
109     {% endif %}
110     </div>
111 </div>

112 <script type="text/javascript">
113     function preloader(){
114         document.getElementById("loading").style.display = "none";
115         document.getElementById("onload").style.display = "block";
116     }
117     window.onload = preloader;
118 </script>
119 <script src="/static/js/tinysort.min.js"></script>
120 <script src="/static/js/thumbnails.js"></script>
121
122 {% endblock %}

```

acsqsl/website/templates/view_query_results.html

```

1  {% extends "base.html" %}
2  {% block content %}
3
4  <link rel="stylesheet" href="/static/css/loader.css">
5  <link rel="stylesheet" href="/static/css/view_proposal.css">
6
7  <div class="row">
8      <div class="span12">
9
10         <!-- Header -->
11         {% if thumbnail_dict.num_images %}
12             <h3>The query returned {{thumbnail_dict.num_images}} results</h3>
13             {% if thumbnail_dict.buttons %}
14                 <form id="filtering" style="margin-bottom:0;">

```

```

16     <!-- Filter -->
17     <div id="filterOptions" class="container">
18         {# for key,buttonlist in thumbnail_dict.buttons.items() #}
19             <div class="form-group">
20                 <label for="{{key}}>Show only {{key}}:</label>
21                 <select class="form-control" id="{{key}}>
22                     <option value="all">All {{key}}s</option>
23                     {# for b in buttonlist #}
24                         <option value="{{b}}>{{b}}</option>
25                     {# endfor #}
26                 </select>
27             </div>
28         {# endfor #}
29     </div>
30
31     <!-- Sort By -->
32     <div class="row" id="sortOptions">
33         <div class="span6">
34             Sort by:&nbsp;&nbsp;
35             <label class="radio inline"><input type="radio" name="sort" id="unsort" checked>default
36         </label>
37             <label class="radio inline"><input type="radio" name="sort" id="sort-expstart">expstart
38         </label>
39             <label class="radio inline"><input type="radio" name="sort" id="sort-exptime">exptime</
40         label>
41             </div>
42             <div class="span5 text-right">
43                 {# if thumbnail_dict.buttons|length > 2 #}
44                 Filter logic:&nbsp;&nbsp;
45                 <label class="radio inline"><input type="radio" name="logic" value="and" checked>"AND"<
46             /label>
47                 <label class="radio inline"><input type="radio" name="logic" value="or">"OR"</label>
48             {# endif #}
49         </div>
50         <div class="span1 text-right">
51             <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
52         </div>
53     </form>
54     {# endif #}
55
56     <!-- Loading animation -->
57     <div id="loading">
58         <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
59             <div class="cssload-cssload-loader-line-wrap-wrap">
60                 <div class="cssload-loader-line-wrap"></div>
61             </div>
62             <div class="cssload-cssload-loader-line-wrap-wrap">
63                 <div class="cssload-loader-line-wrap"></div>
64             </div>
65             <div class="cssload-cssload-loader-line-wrap-wrap">
66                 <div class="cssload-loader-line-wrap"></div>
67             </div>
68             <div class="cssload-cssload-loader-line-wrap-wrap">
69                 <div class="cssload-loader-line-wrap"></div>
70             </div>
71         </div>
72         <br>
73         Loading thumbnails...
74     </div>
75
76     <!-- Thumbnails -->
77     <div id="onload">
78         <div id="thumbnail-array">
79             {# for i in range(thumbnail_dict.num_images) #}
80                 <div class="thumb" page="{{i+1}}" detector="{{thumbnail_dict.detectors[i]}}"
filter1="{{thumbnail_dict.filter1s[i]}}" filter2="{{thumbnail_dict.filter2s[i]}}" expstart="{{
thumbnail_dict.expstarts[i]}}" exptime="{{thumbnail_dict.exptimes[i]}}" targname="{{thumbnail_dict.
targnames[i]}}" proposid="{{thumbnail_dict.proposal_ids[i]}}" visit="{{thumbnail_dict.visits[i]}}"
style="background:url(/../{{thumbnail_dict.thumbs[i]}}) no-repeat;">
                    <div class="overlay">
                        <strong>{{thumbnail_dict.filenames[i]}}</strong><br><font size="0.5">
                        <b>Detector:</b> {{thumbnail_dict.detectors[i]}}<br>
                        <b>Filter:</b> {{thumbnail_dict.filter1s[i]}} / {{thumbnail_dict.filter2s[i]}}
                    </div>
81             </div>
82         {# endfor #}
83     </div>
84 
```

```

] } }<br>
      <b>Exptime:</b> {{thumbnail_dict.exptimes[i]}}<br>
      <b>Target:</b> {{thumbnail_dict.targnames[i]}}<br></font>
    </div>
    <a href="{{thumbnail_dict.viewlinks[i]}}" target="_blank">
      
    </a>
  </div>
  {# endfor %}
</div>
</div>
{# else %}
<p>
  <h5>There are no images to display.</h5>
</p>
{# endif %}
</div>
</div>
<script type="text/javascript">
  function preloader(){
    document.getElementById("loading").style.display = "none";
    document.getElementById("onload").style.display = "block";
  }
  window.onload = preloader;
</script>
<script src="/static/js/tinysort.min.js"></script>
<script src="/static/js/thumbnails.js"></script>
{# endblock %}

```