

The Hubble Space Telescope Advanced Camera for Surveys Quicklook Application

Matthew Bourque¹, Sara Ogaz¹, Alex Viana², Meredith Durbin³, and Norman Grogin¹

[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] Dept. 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) instrument 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 its 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 a third-generation imaging instrument on board the Hubble Space Telescope (HST), and 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[2]; the HRC still remains unoperational.

Despite this electronics failure, the ACS instrument has been steadily acquiring astronomical images over its 15 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

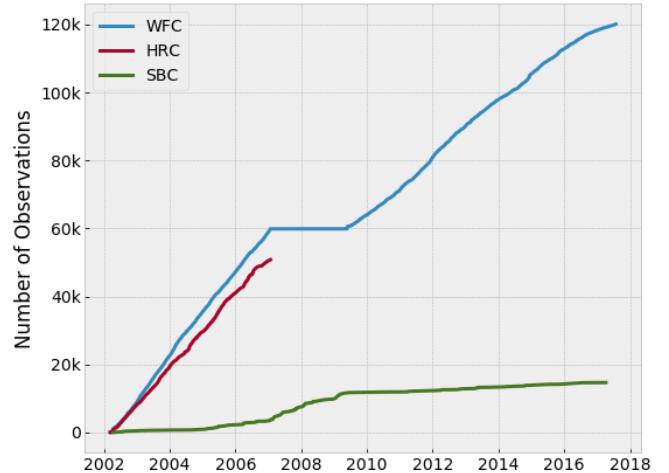


Fig. 1: The number of observations over time for each of the three detectors on ACS.

MAST, users can request and retrieve data for any publicly-available dataset via `ftp`, `sftp`, or `DVD` by `mail`². Like most 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 several major components: (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 database and web application (hereinafter referred to as the “`acsq1` library” or “`acsq1` package”). Each of these subsystems are explained in further detail in Chapter 3 (Methodology).

This paper outlines ACS Quicklook 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 related work to this project and how the `acsq1` application compares. Chapter 3 details the implementations of each of the `acsq1` components. Chapter 4 outlines the results of the project, namely 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 the ACS Quicklook application, especially the database, is heavily dependant on the underlying data structure of ACS FITS files. As such, it is important for the reader to understand this data structure and thus the next four sections are dedicated to giving an overview on the subject.

1.1.1 Filenames

Each ACS data file is named in a consistent fashion:

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

where each `<rootname>` consists of nine unique alphanumeric characters, and `<filetype>` is one of several three-character filetype options (discussed in proceeding section 1.1.4). For example, one ACS observation has the rootname `j6mf16hq_raw.fits` (Principle Investigator Gary Bernstein, observation date 2016-09-22). Each character in the 9-character `rootname` has meaning, and is discussed in section 5.2 of the Introduction to the HST Data Handbooks[3]. The `.fits` extension at the end of the filename signifies that the file is of FITS format.

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

TABLE 1: ACS/WFC FITS file extensions

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

noted that only the first eight characters of the 9-character `rootname` is truly unique; the last character, which identifies the source of data transmission from the telescope³, has several possible values, however varying values do not associate with different observations. For this reason, as will be discussed in section refsec3.5, we adopt an 8-character `rootname` as a primary key to signify unique ACS observations.

1.1.2 FITS file structure

Each ACS FITS file consists of several ‘extensions’, with each extension serving a purpose to describe 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 observation date was 2016-09-22) (discussed in the next section), 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 extension data describe a scientific observation, (2) ‘error’ (`ERR`), in which the extension data describe the uncertainty in the pixel values of the `SCI` data, and (3) ‘data quality’ (`DQ`), in which the extension data 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 that contains 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.

Over the years, there have been several tools written in various programming languages to read in FITS files and automatically convert their extension data to multi-dimensional array data types and their extension headers to dictionary data types. For this project, the `astropy.fits`

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

```

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 = 0.000000000000E+00 / right ascension of the target (deg) (J2000)
DEC_TARG = 0.000000000000E+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
FGSLOCK = '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: An example header.

python library is used extensivly to read and interact with ACS FITS files[4].

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 includes various data that describes the astronomical observation (e.g. target name, exposure time, principle investigator name, etc.), telemetry of ACS 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.

Extension headers may contain a large number of keyword/value pairs. Some extension headers contain upwards of 300 keywords, while others may contain only \sim 40 keywords.

1.1.4 FITS filetypes for ACS

As discusses in section 1.1.1, each ACS observation may result in several FITS filetypes. Each filetype describes the observation in a different way. 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. Also beyond the scope of this paper are the vast details that surround each filetype; each one has a different scientific application that is not important to understanding the ACS Quicklook project. However, to provide at least 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
- **fcl** - nominally calibrated data plus corrected for Charge Transfer Efficiency (CTE) deficits.
- **drz** - Geometric distortion-corrected data
- **drc** - Geometric distrotion-corrected plus 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 has the filetypes raw, flt, jit, jif, and spt.

1.2 Key Metadata

There are several metadata key/value pairs that are particulary important for the ACS Quicklook application, specifically the web application. 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 that were explicitly added to the database schema.

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

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. 52473.8).

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 # 1 (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 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 *acsq1* system is driven by several limitations of the FITS file structure as well as the current capabilities of MAST from specific user perspectives. Some of these limitations are described below, along with the way the ACS Quicklook application aims to address them. We also discuss the intended users of the application and the 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.

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 *sftp*, transferred to a “staging area” in which the user can log into and copy the data via *ftp* at their leisure, or sent by mail via DVD, depending on which option the user selects. In the case of any one of these options, the time between a download request and the time in which the user has fully retrieved the data may be a significant amount of time. In the fastest scenario of 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 of interest via a query to the MAST database, (2) construct a download request, (3) submit the download request, and (4) await the data.

The ACS Quicklook system attempts to circumnavigate this retrieval process by making the full data products instantly available via read-only access of the *acsq1* filesystem, as well as a subset of the data products (and corresponding metadata) instantly available to view through the 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 I/O. 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. *acsq1* attempts to mitigate this processing time by storing the valuable image metadata that users often seek in a relational database.

Queries to the *acsq1* 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 thousand files. Figure N shows that the total time required to gather this information is $\sim N$ times less when querying the *acsq1* database than when retrieving the data directly from the FITS files.

In order to build the *acsq1* database, each ACS FITS file must ultimately be opened in order to retrieve the header information. However, this operation is only performed once and needed not to be performed by the user.

1.3.3 Data redundancy

As will be discussed in section 1.3.5, the primary user base for *acsq1* 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 one day, but not the next, while some data is consistently being analyzed over time. Furthermore, instrument analysts may share the need for specific data with one or more of their colleagues. Consider a situation in which two instrument analysts require data from a proposal 11655. Traditionally, for the two instrument analysts to obtain the data they need, they would have to

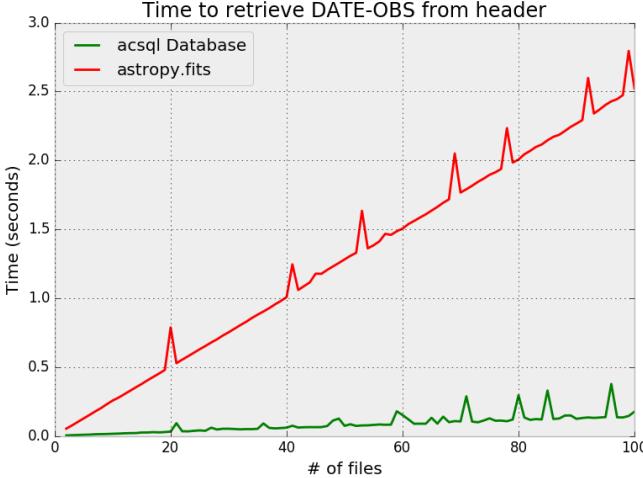


Fig. 3: The amount of time it takes to retrieve the DATE-OBS header keyword from a varying number of ACS data files via reading the FITS file using the `astropy.io.fits` module (red) and querying the `acsq1` database (green)

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 copy of this data, when in reality, only one copy is needed. As illustrated, 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; 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 know the observational parameters associated with them.

MAST has tools which allow users to discover rootnames and/or parameters for observations, such as the MAST portal, which 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 (further discussed in Chapter 3) (MAST, 2017). The MAST

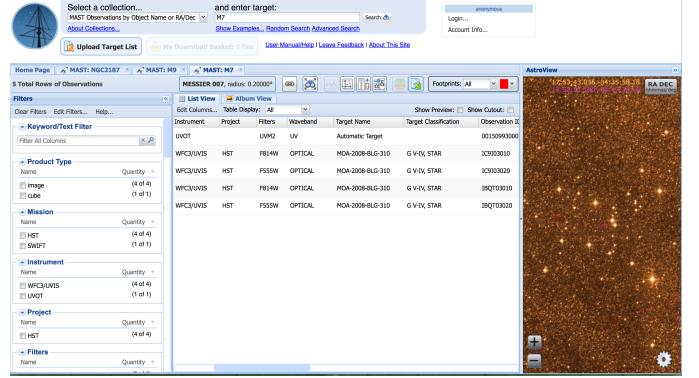


Fig. 4: An example of a query to the MAST data discovery portal.

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 of this tool is shown in Figure N.

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 observational metadata available through the MAST portal does not include all of the observational metadata available in the FITS headers; it is limited to a small subset of header keywords, mainly from the RAW and FLT filetypes, amongst other MAST-generated metadata (e.g. 'Target Classification'). Another limitation is that the result of a query is keyed off of an 'Observation ID' as opposed to a traditional 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. Similarly, this makes it difficult to use the MAST portal to query for information based on an individual observation.

The `acsq1` application aims to provide an intuitive, one-to-one correspondence between the FITS header values and the `acsq1` database schema. The column names in the header 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 header table in the database is the 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 the data to help calibrate, monitor, and characterize the various aspects and behaviors 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 instrument 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 limited to this user base; since the application is non-proprietary, other potential users exist, such as scientists and researchers external to STScI.

With the nature of ACS instrument work in mind, the `acsq1` application was designed to achieve 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.

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 `IPPPSSOTT` 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 `IPPPSSOTT` rootname.

Use case 4: Programmatically analyze images across custom datasets: The application shall allow users to 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 ACS Quicklook 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 (Bourque et al., 2016). The `wfc3ql` has been developed by the WFC3 team at STScI since WFC3 was installed on HST in 2009 during Servicing Mission 4. The project started out as 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 directory. Since then, the project has evolved to the filesystem/database/web application stack that more closely resembles that of `acsq1`.

However, 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 detailed in Chapter N). For example, unlike the `acsq1` filesystem, which contains only publicly-available data, the `wfc3ql` filesystem is essentially a copy of the MAST archive for WFC3. As such, it 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 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.

Furthermore, with `wfc3ql` being continuously developed over the past eight years by a team of several developers, there exist several features in `wfc3ql` 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 upon a cache of only publicly-available data (and thus the project can be considered open-source and non-proprietary). Secondly, `wfc3ql` currently contains several instrument calibration and monitoring routines that are built upon the `wfc3ql` automation platform/code library (known as `pyql`), while `acsq1` has yet to implement any of such routines. A third example is that the `wfc3ql` web application supports the tracking of WFC3 instrument ‘anomalies’ (i.e. anomalous features in images that should be noted to instrument team members (Gosmeyer, 2017)). These features, 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 subsystems of the ACS Quicklook system. Additionally, we discuss the programming standards and standard workflows that were employed to promote code quality such as readability, maintainability, extensibility, etc; we believe that this aspect of the project is at least equally important to the system as its individual components.

3.1 Version control

All code associated with this project (including this paper itself) is version controlled using the `git` Version Control System (VCS) (git, 2017). The `git` repository for the project is named `acsq1`. The `git` repository is also hosted on GitHub, a repository hosting service (GitHub, 2017), and is publicly available at <http://github.com/spacetelescope/acsq1/>.

Several feature branches of the code were created throughout the building of this project such that the `master` branch (which is considered the “production” branch) always contained operational code (while the code in the branches may contain unfinished implementations). Such branches include `create-database` (for implementation of the database schema), `add-logging` (for implementation of system logging), `build-ingest` (for implementation of data ingestion software), and `web-application` (for implementation of the web application). For each merge of a feature branch, a `tag` and `release` was created for the `master` branch, which allows 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, using GitHub allowed for issue tracking of bugs, features, and potential enhancements to the code repository. Current open issues of the repository can be found at <https://github.com/spacetelescope/acsq1/issues>.

```

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`.

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 (van Rossum, 2001), (2) The PEP257 python guide for module and function docstring conventions (Goodger, 2001), and (3) the numpydocs documentation standard (NumPy Documentation, 2017). 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 were these conventions not followed, such as using a single line of code, even if it exceeded the recommended 80 characters, to allow for greater readability. By following these conventions, the style of the acs1l code is consistent amongst each module and attempts to reflect the style of industry-grade python code.

The PEP257 guide for docstring conventions describes standard conventions 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 acs1l code documentation. Furthermore, the numpydocs documentation convention provides some additional details on top of the PEP257 conventions and is used in many python packages including the `numpy` (numerical python) and `scipy` (scientific python) packages (van der Walt et al., 2011). Figure N shows an example of these conventions, taken from the `acsq1.ingest.ingest.get_proposal_type` function.

Another benefit to using PEP257 and numpydoc docstring conventions is that API documentation creation tools such as `sphinx` (Brandi et al., 2007) or `epydoc` (Loper, 2004) can automatically convert the docs 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 the `readthedocs`, which is an open-

The screenshot shows the `get_proposal_type` function documentation. It includes the function signature, parameters (`proposid`), return types (`proposal_type`), and a detailed description of what the function does, mentioning it scrapes from the MAST proposal status webpage. It also shows examples of how to use the function.

Fig. 6: The `readthedocs` documentation for the `acs1l` example function seen in Figure N.



Fig. 7: A representation of the directory structure within the `acs1l` filesystem, using a few observations as an example.

source, community supported tool for hosting and browsing documentation (Read the Docs, 2017). The documentation for `acs1l` is hosted at <http://acs1l.readthedocs.io/>. The output documentation as seen on `readthedocs` for the example function in figure N is provided in Figure N.

3.3 Filesystem: Archive of ACS data

The `acs1l` filesystem is a Network File System (NFS) that stores all on-orbit ACS data on disk in an organized set of directories and subdirectories hosted at STScI. Figure N shows an example of this directory structure: The parent directory is the first four characters of the 9-character rootname, which maps directly to an individual PROPOSID. The sub-directories of the parent directories are named after the full 9-character rootname such that each parent directory contains the rootname subdirectories that were observed for that particular PROPOSID. Each rootname subdirectory contains every available filetype (as described in Section 1.1.4) for the particular observation is stored.

Figure N shows how the total size of the filesystem has evolved over the lifetime of the mission; currently, the filesystem occupies ~40 TB of storage space. Note that the file sizes across the detectors and across the various filetypes

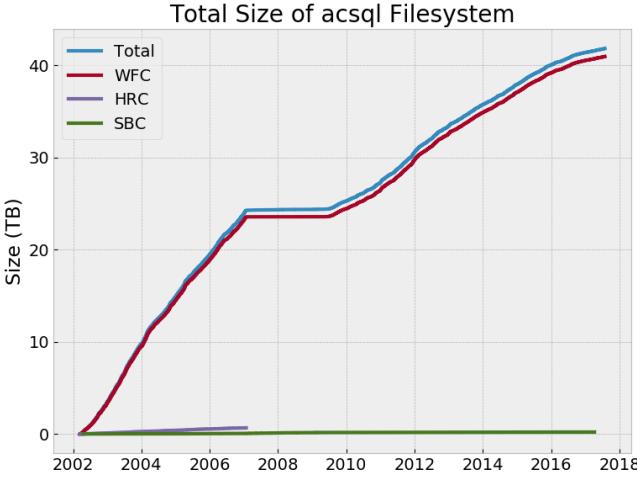


Fig. 8: The size of the `acsql` filesystem as a function of observation date.

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.).

3.4 Filesystem: Archive of JPEGs and Thumbnails

In addition to the ACS data products described in the last section, the `acsql` 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 used by the `acsql` web application to allow users to quickly and easily view the contents of the data without having to physically open the corresponding `.fits` files.

The JPEG images are generated by taking the two-dimensional data from the SCI extension(s), sigma-clipping the top and bottom 1% of the values (as to avoid large outlier values and enhance the scaling of the image), and saving the data to a a JPEG format. The thumbnail images are created by simply resizing the corresponding JPEG into a 128x128 pixel image and saving 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 `acsql` web application. An example of a JPEG image and its corresponding thumbnail is shown in Figure N.

Unlike the ACS data products portion of the filesystem, the JPEG and thumbnail portions of the filesystem are organized based on the 5-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 which to view data based on the 5-digit PROPOSID and less often on the details of the rootname. An example of this sturcture is shown in Figure N. Note that the thumbnail filesystem only contains thumbnails created from FLT filetypes, since thumbnails are only inteded for navigation and quick-viewing.

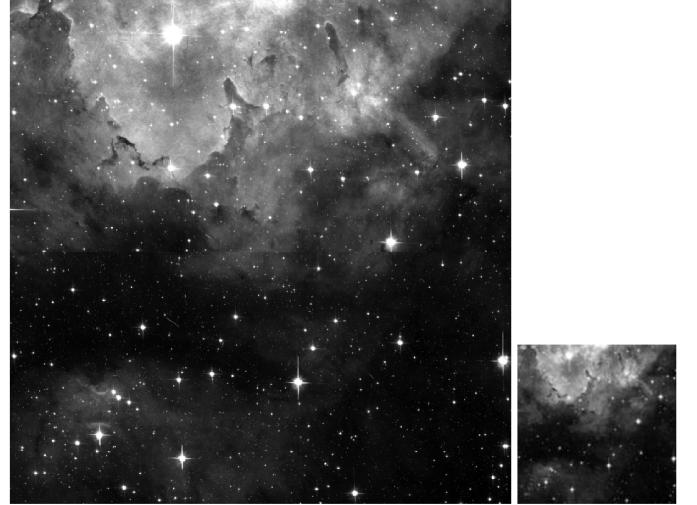


Fig. 9: An example of a JPEG image (left) and its corresponding thumbnail image using example dataset `jcs718koq`.

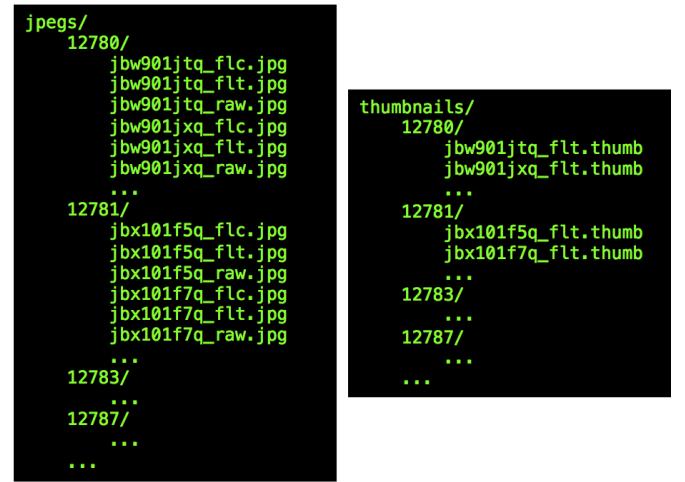


Fig. 10: A representation of the directory structure for the JPEG (left) and thumbnail (right) portion of the `acsql` filesystem, using a few observations as an example.

3.5 Database: Relational Schema

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

To accomplish this, we implemented the relational schema shown in Figure N. The `acsql` database contains 111 tables in total: one master table which contains basic information about each `rootname` that is important for the `acsql` database in general, 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.

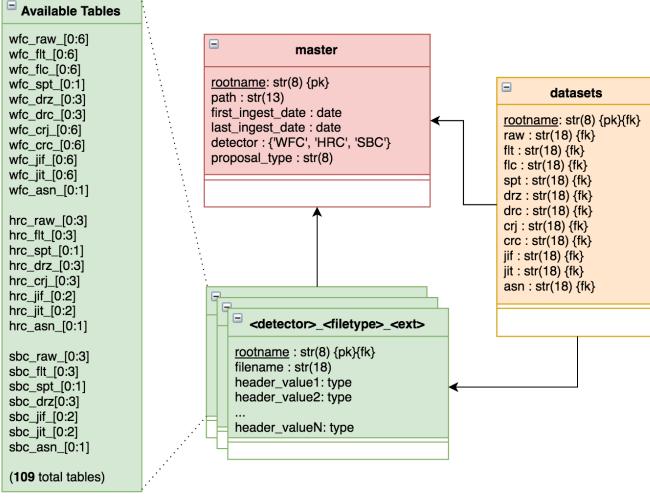


Fig. 11: The relational database schema for the acsql database.

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 `jcs718ko_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 correspondance with the header metadata key/value pairs found in observations 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 with 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) (Oracle, 2017) 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 (Bayer, 2006). As an ORM, SQLAlchemy enables python classes to be easily translated to SQL-based database tables, and vice versa. Additionally, SQLAlchemy provides python 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 `acsq1.database.database_interface.py` module). One such function is the `load_connection`, as shown in Figure N. This function creates three SQLAlchemy objects that are used to establish a connection with the acsql database: `engine`, `base`, and `session`, each described below.

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 (Lemburg, 2017). It is created from the `sqlalchemy.create_engine` method, which requires a user-supplied `connection_string`. The `connection_string` is a string that contains information about the type of database, 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 acsql, 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 acsql library (as will be discussed in section 3.9).

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 tables from 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 via explicit class definitions in `database_interface`, and are shown in Figures N and N, respectively. Each table column is defined using the `sqlalchemy.Column` object, which is a class that can be initialized with the datatype that will be stored in the column (e.g. `String`, `Float`, `Integer`, etc.) as well as parameters that set SQL-like constraints and parameters 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).

```

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

    Create an ``engine`` using an 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 ``acsqll`` 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=100000)
base = declarative_base(engine)
Session = sessionmaker(bind=engine)
session = Session()

return session, base, engine

```

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

```

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.

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 manner 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

```

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)
    ffc = 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.

```

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.

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 N. 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 N.

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

```

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.

(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`. An portion of an example text file is shown in Figure N.

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 set of example FITS files to scrape its 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` is used to add new header keywords by comparing the header contents of a given FITS file and the existing column

```

DETECTOR, String
NEXTEND, Integer
EXTEND, Bool
SIMPLE, Bool
NAXIS, Integer
LINENUM, String
GROUPS, Bool
DATE, String
EQUINOX, Float
INSTRUME, 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.

definition text files⁴.

With the implementation of the `orm_factory` and `define_columns` function, it is then trivial to create class definitions for each of the 109 header tables. An example of this is shown in Figure N, where the 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 accom-

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

plished by executing the `base.metadata.create_all()` method.

3.7 Data ingestion software: Algorithm

Another critical component of the ACS Quicklook system are the modules that are used to ingest data into the acsql database and to create the “Quicklook” JPEGs and thumbnails. By the term ‘ingest’, we refer to the following algorithm:

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 acsql 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 ingestion software (i.e. the `acsql.ingest.ingest` module), takes as input a single rootname, such that if there are multiple rootnames to be ingested, the calls to the ingestion module can be parallelized over many CPUs. The ingestion of one rootname does not depend on the ingestion of another, nor is the order of which files are ingested important. Please note that steps 3 through N are written from the perspective 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 metadata 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 N.

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. Please note that steps 5 through N are written from the perspective that a single file 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 being passed around to various 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 which 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

```
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.

converting hypens in header keys to underscores as to avoid python errors), it is rather trivial to perform and `insert` or `update` operation via the `insert_or_update` function (see Figure N).

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 it are these filetypes that contain actual 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 high or low-valued pixels, and

saved to a .jpg format. The JPEGs are saved to the JPEG 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 the larger JPEG images via the acsql web application. Thumbnails are generated by simply opening up the corresponding flt JPEG and resizing it to 128x128 pixels. The Thumbnails are saved to the Thumbnail portion of the acsql filesystem (described in section 3.4).

This workflow is encapsulated within several modules across the acsql.scripts and acsql.ingest subpackages, as will be described in section 3.9. These modules are intended to be executed daily (as an automatically-spawned process) as to keep the ACS Quicklook system up-to-date on any public data as it becomes available.

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 by which the status of the ingestion process can be logged to an output text file and 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 has 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 N.

When the ingestion module gets executed, 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 3.9), 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 which ingestion run.

Next, the python logging module is used to configure the format of the log statements. It does this 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 code `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 Figure N.)

Calls to the logging module are strategically placed within the ingestion software to provide enough context to 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 system (i.e. an updated database table, the creation of a JPEG or Thumbnail.)

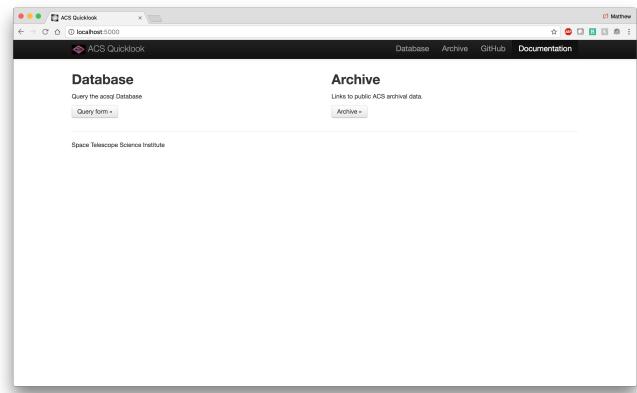


Fig. 21: The homepage of the acsql web application.

3.9 Web Application

The front-end of the acsql system is the web application. The web application is built using Python and Flask, which is a Python based web framework (Ronacher, 2010). 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 visualize these features, we show examples of some of the different webpages that make up the web application in Figures N through N, and further describe each below.

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

Figure N shows an example of the database query page. This page allows users to fill out a form that in turn gets converted to a query of the acsql database and executed when 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 or potentially frequently used by the user. As such, we expect that this page to 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 that were 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 N), (2) a CSV

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.

```

08/15/2017 11:05:26 INFO: User: bourque
08/15/2017 11:05:26 INFO: Python Version: 3.5.2 |Continuum Analytics, Inc.| (default, Jul 2 2016, 17:53:06) [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
08/15/2017 11:05:26 INFO: Python Path: /bourque/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: /bourque/envs/anaconda3/envs/astroconda3/lib/python3.5/site-packages/numpy
08/15/2017 11:05:26 INFO: Astropy Path: /bourque/envs/anaconda3/envs/astroconda3/lib/python3.5/site-packages/astropy
08/15/2017 11:05:26 INFO: SQLAlchemy Version: 1.1.4
08/15/2017 11:05:26 INFO: SQLAlchemy Path: /bourque/envs/anaconda3/envs/astroconda3/lib/python3.5/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 11:05:52 INFO: j8zh21xv: Begin ingestion
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).

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

file, which is downloaded to the user's machine, containing a comma-separated table of the selected output columns (see Figure N), or (3) a webpage showing the Thumbnail images of all of the resulting images (see Figure N).

Figure N 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 (see Figure N). Similar to the database query results shown in Figure N, the 'view proposal' webpage shows thumbnail images of every 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 N shows an example of this webpage, using proposal 14039.

In the 'view proposal' webpages, users may click on any

rootname	pr_inv_i	pr_inv_f	proposid	aperture	date_obs	time_obs	targname
j8zh50ck	j8zh50ck	Levay	Zolt	12812	WFC-FIX	2012-11-03	17:42:03
j8zh50co	j8zh50co	Levay	Zolt	12812	WFC-FIX	2012-11-03	18:03:18
j8zh51en	j8zh51en	Levay	Zolt	12812	WFC-FIX	2012-10-22	16:51:05
j8zh51er	j8zh51er	Levay	Zolt	12812	WFC-FIX	2012-10-22	17:12:20
j8zh52er	j8zh52er	Levay	Zolt	12812	WFC-FIX	2012-10-23	16:47:15
j8zh52k3	j8zh52k3	Levay	Zolt	12812	WFC-FIX	2012-10-23	17:08:30
j8zh53kw	j8zh53kw	Levay	Zolt	12812	WFC-FIX	2012-10-23	23:36:15
j8zh53jt	j8zh53jt	Levay	Zolt	12812	WFC-FIX	2012-10-24	00:51:28
j8zh54bf	j8zh54bf	Levay	Zolt	12812	WFC-FIX	2012-10-26	13:24:33
j8zh54bj	j8zh54bj	Levay	Zolt	12812	WFC-FIX	2012-10-26	13:45:48
j8zh55az	j8zh55az	Levay	Zolt	12812	WFC-FIX	2012-10-27	03:46:22
j8zh55ts	j8zh55ts	Levay	Zolt	12812	WFC-FIX	2012-10-27	04:37:37
j8zh56ht	j8zh56ht	Levay	Zolt	12812	WFC-FIX	2012-10-27	23:21:48
j8zh56lo	j8zh56lo	Levay	Zolt	12812	WFC-FIX	2012-10-28	00:30:13
j8zh57ac	j8zh57ac	Levay	Zolt	12812	WFC-FIX	2012-11-05	01:36:54
j8zh57ag	j8zh57ag	Levay	Zolt	12812	WFC-FIX	2012-11-05	01:58:09
j8zh58kp	j8zh58kp	Levay	Zolt	12812	WFC-FIX	2012-11-07	19:01:53
j8zh58at	j8zh58at	Levay	Zolt	12812	WFC-FIX	2012-11-07	19:23:08
j8hd10q	j8hd10q	Levay	Zolt	13823	WFC-FIX	2014-02-07	06:03:09
j8hd10tl	j8hd10tl	Levay	Zolt	13823	WFC-FIX	2014-02-07	06:26:17
j8hd10dr	j8hd10dr	Levay	Zolt	13823	WFC-FIX	2014-02-07	07:31:07
j8hd10dv	j8hd10dv	Levay	Zolt	13823	WFC-FIX	2014-02-07	07:54:15
j8hd206	j8hd206	Levay	Zolt	13823	WFC-FIX	2014-02-07	09:14:29
j8hd208	j8hd208	Levay	Zolt	13823	WFC-FIX	2014-02-07	09:37:37
j8hd20ee	j8hd20ee	Levay	Zolt	13823	WFC-FIX	2014-02-07	10:42:29

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

one of the thumbnail images to bring up the 'view image' webpage for that particular image. An example of this webpage is shown in Figure N, 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.

3.9.1 Implementation

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

- acsql_webapp.py
- data_containers.py
- form_options.py
- query_form.py
- query_lib.py

query_results.csv													
<pre>1 routines_proposal_inv_f.proposal.aperture,date_obs,time_obs,targname 2 jbx150ck,Levay,Zolt,12812,WFC-FIX,2012-11-03,17:43:03,ANY 3 jbx150co,Levay,Zolt,12812,WFC-FIX,2012-11-03,18:03:18,ANY 4 jbx151en,Levay,Zolt,12812,WFC-FIX,2012-10-22,16:51:05,ANY 5 jbx151er,Levay,Zolt,12812,WFC-FIX,2012-10-22,17:12:28,ANY 6 jbx152iz,Levay,Zolt,12812,WFC-FIX,2012-10-23,16:47:15,ANY 7 jbx152k3,Levay,Zolt,12812,WFC-FIX,2012-10-23,17:08:30,ANY 8 jbx153kw,Levay,Zolt,12812,WFC-FIX,2012-10-23,23:36:15,ANY 9 jbx153l7,Levay,Zolt,12812,WFC-FIX,2012-10-24,00:51:28,ANY 10 jbx154bf,Levay,Zolt,12812,WFC-FIX,2012-10-24,13:24:33,ANY 11 jbx154bj,Levay,Zolt,12812,WFC-FIX,2012-10-24,13:45:48,ANY 12 jbx155ez,Levay,Zolt,12812,WFC-FIX,2012-10-27,03:46:22,ANY 13 jbx155f3,Levay,Zolt,12812,WFC-FIX,2012-10-27,04:07:37,ANY 14 jbx156ht,Levay,Zolt,12812,WFC-FIX,2012-10-27,23:13:45,ANY 15 jbx156jw,Levay,Zolt,12812,WFC-FIX,2012-10-28,01:30:11,ANY 16 jbx157aq,Levay,Zolt,12812,WFC-FIX,2012-11-05,01:58:09,ANY 17 jbx157aq,Levay,Zolt,12812,WFC-FIX,2012-11-05,01:58:09,ANY 18 jbx158sp,Levay,Zolt,12812,WFC-FIX,2012-11-07,19:01:53,ANY 19 jbx158st,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 jchx01dr,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 jchx02ei,Levay,Zolt,13623,WFC-FIX,2014-02-07,16:42:29,ANY 27 jchx02ei,Levay,Zolt,13623,WFC-FIX,2014-02-07,17:05:37,ANY 28 jchx03id,Levay,Zolt,13623,WFC-FIX,2014-02-08,07:56:56,ANY 29 jchx03ii,Levay,Zolt,13623,WFC-FIX,2014-02-08,09:02:01,ANY 30 jchx03im,Levay,Zolt,13623,WFC-FIX,2014-02-08,09:25:09,ANY 31 jchx04jz,Levay,Zolt,13623,WFC-FIX,2014-02-11,12:04:41,ANY 32 jchx04k1,Levay,Zolt,13623,WFC-FIX,2014-02-11,12:27:49,ANY 33 jchx04k6,Levay,Zolt,13623,WFC-FIX,2014-02-11,13:33:58,ANY</pre>													
Line 1, Column 1													
Spaces: 4 Plain Text													

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

ACS Archive													
8193	9603	9830	10188	10459	10626	10907	11655	12250	12759	13407	13942		
8947	9658	9831	10189	10460	10627	10909	11658	12253	12760	13410	13945		
8948	9660	9835	10190	10461	10628	10910	11663	12254	12760	13412	13952		
8962	9662	9838	10192	10466	10629	10911	11669	12256	12761	13419	13953		
8963	9663	9837	10195	10471	10630	10913	11670	12257	12762	13420	13954		
9005	9664	9838	10196	10472	10631	10915	11675	12262	12763	13422	13955		
9006	9665	9842	10198	10473	10632	10917	11678	12270	12767	13425	13956		
9008	9668	9847	10199	10474	10633	10918	11677	12271	12768	13433	13957		
9009	9667	9851	10200	10475	10634	10919	11679	12273	12769	13435	13958		
9010	9668	9853	10201	10476	10635	10920	11681	12282	12790	13437	13959		
9011	9674	9854	10204	10486	10695	10921	11683	12286	12791	13439	13960		
9012	9675	9856	10205	10486	10696	10922	11684	12292	12811	13441	13961		
9013	9678	9857	10206	10487	10697	10923	11688	12297	12812	13442	13962		
9014	9683	9860	10207	10488	10698	10991	11689	12309	12817	13443	13964		
9015	9684	9861	10210	10491	10701	10992	11691	12310	12858	13449	13965		
9016	9688	9862	10213	10490	10703	10994	11695	12311	12866	13465	13966		
9017	9687	9863	10214	10491	10705	10996	11704	12313	12866	13461	14034		
9018	9688	9864	10216	10492	10707	10997	11710	12317	12871	13463	14035		
9019	9647	9869	10217	10493	10710	11038	11711	12323	12872	13470	14037		
9020	9648	9872	10227	10494	10711	11054	11713	12326	12875	13474	14038		
9022	9649	9873	10228	10495	10713	11055	11715	12327	12877	13476	14039		
9023	9650	9877	10229	10496	10715	11058	11718	12328	12878	13479	14056		
9024	9651	9884	10231	10497	10718	11051	11722	12331	12886	13480	14057		
9025	9652	9889	10233	10498	10720	11053	11724	12362	12884	13483	14061		
9026	9654	9890	10235	10499	10722	11055	11731	12367	12908	13485	14063		
9027	9655	9891	10237	10500	10729	11054	11734	12368	12911	13493	14072		

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

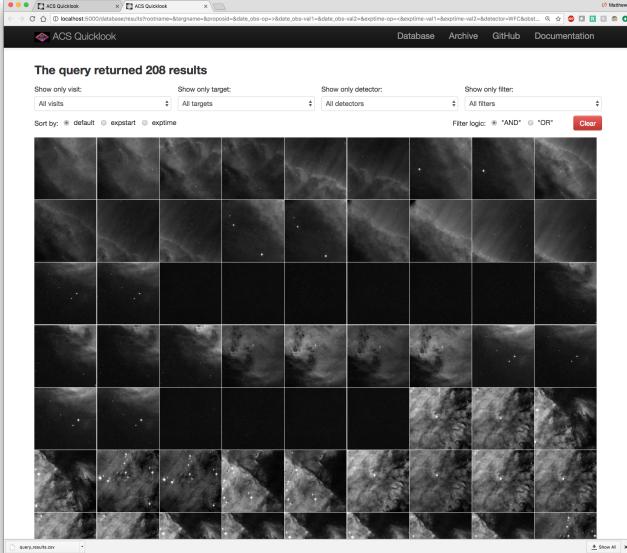


Fig. 25: The results of the database query shown in Figure N in the form of thumbnail images.

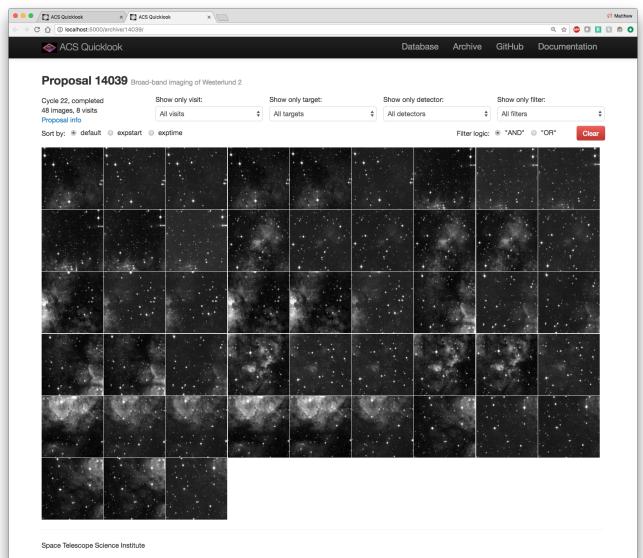


Fig. 27: An example of a ‘view proposal’ webpage, using proposal 14039.

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

- 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 acsql_webapp module is the main module for running the acsql web application; it is this module that must be executed in order to serve the web application (either on a dedicated web server, or locally on a user’s machine, accessed through the user’s localhost). This module uses the Flask web framework to receive incoming HTML

requests. It is structured such that each acsql webpage relates to an individual function within acsql_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 acsql webpages (e.g. lists of images, image metadata, paths to JPEG images on disk, etc.) is imported into acsql_webapp from the data_containers.py module. That is to say, many of the functions within the acsql_webapp module calls functions from data_containers to gather data to eventually pass

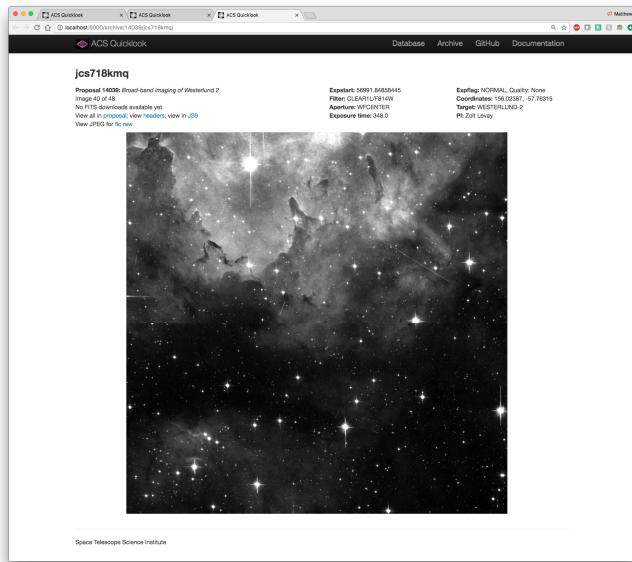


Fig. 28: An example of a ‘view image’ webpage, using dataset jcs718kmq.

on to the HTML template. This design choice was made in order to separate the data from the functionality, which is often regarded as a best practice in software engineering.

Along those same lines, the `form_options.py` module is simply 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 that 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`). 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 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 (bootstrap).

3.10 acsq1 Package

All code associated with the `acsq1` project is contained within a single git repository (also named `acsq1`), which we refer to as the “`acsq1 Library`”, or “`acsq1 package`”. The package layout is shown below:

```
acsq1/
    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
    acsq1/
        __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
            acsq1_webapp.py
            data_containers.py
            form_options.py
            query_form.py
            query_lib.py
            static/
                css/
                    *.css
                img/
                    jpeg
                    thumbnails
                js/
                    *.js
            templates/
                *.html
```

We now provide a brief description of each package component:

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 paper.

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

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

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

requirements.rst: A list of acsql package dependencies, used by Read the Docs to build a virtual machine that constructs the resulting html doc pages (see Section 3.2).

source/: A subdirectory that contains 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 that contains 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 that indicates 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 that contains several examples of queries that can be used with the acsql database.

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

table_definitions: A subdirectory containing all of the `<detector>_<filetype>_<extension>` 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 acsql system, 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 4.7).

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 acsql filesystem, as well as storage place for possible future instrument calibration and monitoring routines.

ingest_production: The Python module for ingesting new ACS data as it becomes publicly available, intended to be executed periodically.

utils/: The utils subpackage, containing Python modules that are useful for general acsql 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 acsql database credentials. Specifically, it contains values for the acsql 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 acsql 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 various 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).

For further details on each Python module within the `acsq1` package, readers are encouraged to view the official API documentation hosted at <http://acsq1.readthedocs.io/>.

4 RESULTS

The results of the system implementation (described in Chapter 2) were several project deliverables (each described further in the sections below):

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

The project deliverables will primarily be used by members of the ACS instrument team at STScI, but may also be used by ACS users external to STScI.

4.1 Filesystem Deliverable

As described in Sections 3.3 and 3.4, the `acsq1` filesystem is comprised of two major parts: (1) A filesystem that stores the archive of publicly-available ACS data (i.e. the 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 very similar way

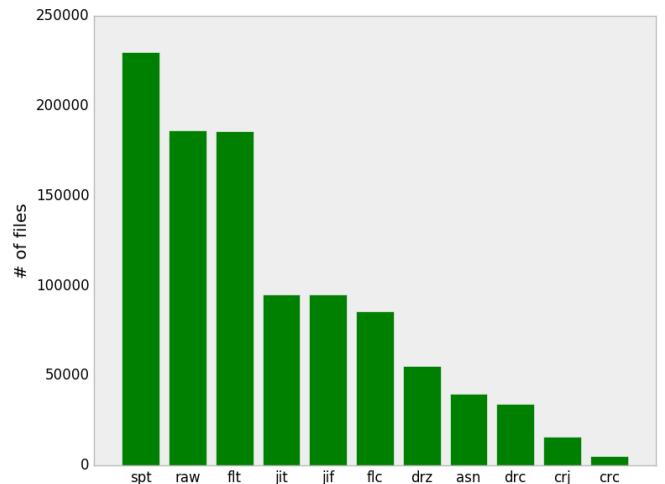


Fig. 29: The number of files in the `acsq1` filesystem by filetype.

to that shown in Figure N⁶. 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 N shows how this breaks down by individual filetype. We see that `spt`, `raw`, and `fit` make up the majority of the files, which is not surprising considering that these filetypes occur for nearly every observing mode. On the other hand, we see a small amount of `crj` and `crc` files, which is also unsurprising considering that these filetypes are only triggered for specific observing modes.

As per 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 3.10). The location of the JPEG/Thumbnail 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 `acsq1` 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 the `ingest_production.py` and `database_interface` modules. All 111 tables of `acsq1` database are up to date as of the time of this writing (September 2017), and it is intended that the database will be kept up-to-date via periodic executions of `ingest_production.py`.

6. We omit the mention of the parent directory structure and server names here as well as in the `acsq1` code repository as to avoid exposing possible sensitive information.

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

Figure N 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 `acsq1` web application consists of a series of Python modules and static files. Currently, the web application is not yet 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 `acsq1` 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 Sections 3.1 and 3.10, the `acsq1` git repository serves as the software package that contains all of the code and supporting materials needed to operate the `acsq1` system. The software package is open-source, and is available to download directly from the GitHub repository (<http://github.com/acsq1/>). All future enhancements and bug fixes to the `acsq1` project 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 `acsq1` package, but are also hosted on `readthedocs`, available at <http://acsq1.readthedocs.io/>. This is considered the official code documentation.

5 DISCUSSION

5.1 Possible enhancements to the `acsq1` system

Though these currently are the only two main features, we expect that the web application may be a source of significant ongoing development, as the number of possible features implemented on top of the `acsq1` database and `acsq1` filesystem are plentiful.

5.2 Possible simplifications based on MAST archive

5.3 Possible extensions to other instruments or missions

6 CONCLUSION

The conclusion goes here.

⁸ At the time of this writing, work is being done to build an web server that is internal to STScI to allow ACS instrument team members to more easily access the `acsq1` web application. We expect this web server to be operation within the next few months.

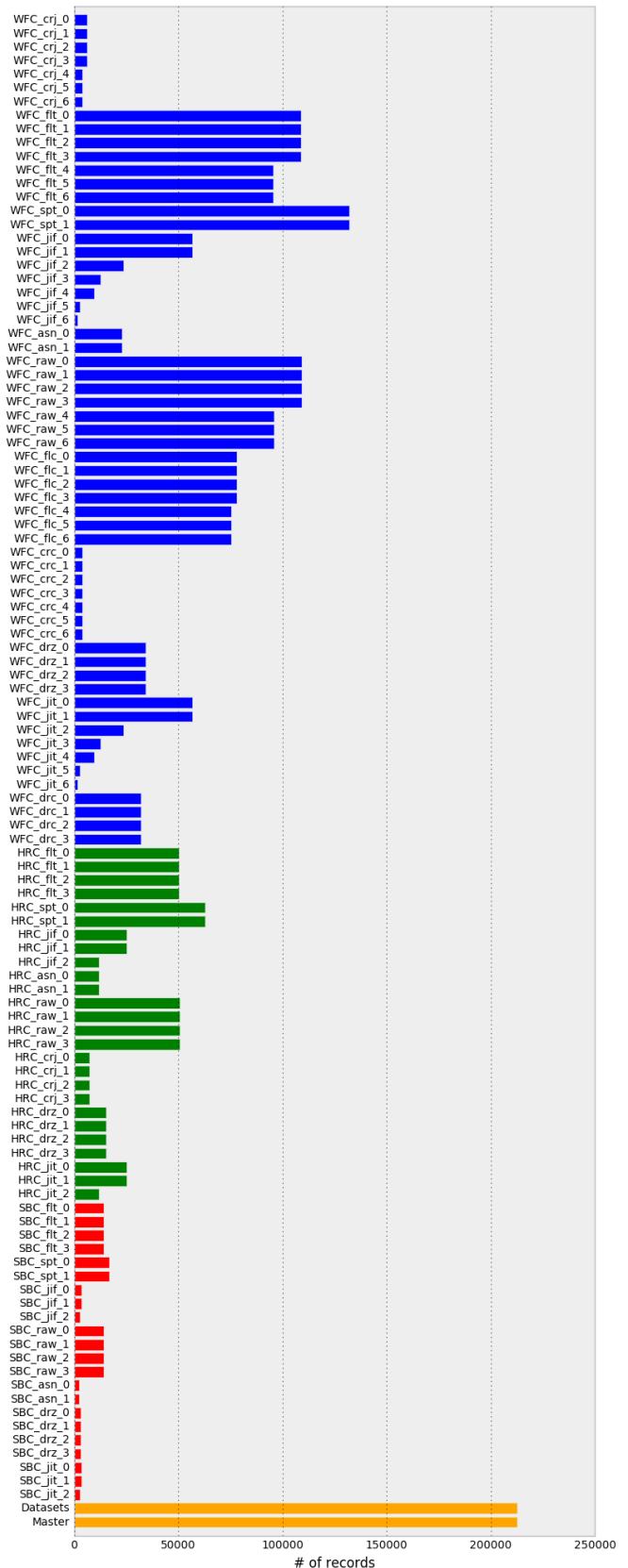


Fig. 30: The number of records in each of the `acsq1` database tables.

ACKNOWLEDGMENTS

The authors would like to thanks various members of the STScI staff that were extremely valuable for help-

ing make this project come together: members of the WFC3/Quicklook team, for their years of ongoing development and support of the WFC3/Quicklook project, without which there would be no inspiration to create this similar project (Varun Bajaj, Ariel Bowers, Michael Dulue, Meredith Durbin, Jules Fowler, Catherine Gosmeyer, Heather Gunning, Harish Khandrika, Catherine Martlin, Abhi Rajan, Clare Shanahan, Ben Sunnquist, Alex Viana); members of the Instruments Division, who support the creation and maintenance of an internal web server (Francesca Boffi, Norman Grogan); Members of the Information Technology and Services Division who provided valuable resources such as the database server and disk space used to store project materials; and finally, Named Person, 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] Smith, E., et al., 2011, *Introduction to the HST Data Handbooks*, Version 8.0 (Baltimore: STScI)
- [6] Robitaille, T.P., et al., 2013, *Astropy: A community Python package for astronomy*, *Astronomy & Astrophysics*, 558, A33.
- [7] *STScI: Space Telescope Science Institute*, [Online; accessed 2017-09-16], available at www.stsci.edu.
- [8] *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
- [9] Bayer, M., 2006, *SQLAlchemy: The database toolkit for Python*, [Online; accessed 2017-02-21], available at <http://www.sqlalchemy.org/>.
- [10] *Bootstrap*, [Online; accessed 2017-09-04], available at <http://getbootstrap.com/>.
- [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] Brandi, G., et al., 2007, *Sphinx: Python Documentation Generator*, available at <http://www.sphinx-doc.org/en/stable/>.
- [14] *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>.
- [15] *git*, [Online; accessed 2017-08-05], available at <https://git-scm.com>.
- [16] *GitHub*, [Online; accessed 2017-08-05], available at <https://github.com>.
- [17] Goodger, D., 2001, *PEP 257 – Docstring Conventions*, Python Developer’s Guide, available at <https://www.python.org/dev/peps/pep-0257/>.
- [18] 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/>.
- [19] Loper, E., 2004, *Epydoc: Generating API Documentation in Python*, Proceedings of the Second Annual Python Conference, available at <http://epydoc.sourceforge.net/>.
- [20] *MySQL 5.6 Reference Manual*, Oracle, [Online; accessed 2017-08-13], available at <https://dev.mysql.com/doc/refman/5.6/en/>
- [21] *Read the Docs*, [Online; accessed 2017-08-05], available at <https://readthedocs.org>.
- [22] Ronacher, A., et al., 2010, [Online; accessed 2017-09-04], available at <http://flask.pocoo.org/>
- [23] *The MAST Portal*, [Online; accessed 2017-09-15], available at <https://mast.stsci.edu/>
- [24] 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.
- [25] 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/>.
- [26] *wtforms*, [Online; accessed 2017-09-04], available at <https://github.com/wtforms/wtforms/>.

APPENDIX A

ACSQL CODE

Table of contents.

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' (http://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 'acsq1' database. Please ask [ @bourque] (http://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

```



```

90 include acsql/database/table_definitions/wfc_jif_1.txt
91 include acsql/database/table_definitions/wfc_jif_2.txt
92 include acsql/database/table_definitions/wfc_jif_3.txt
93 include acsql/database/table_definitions/wfc_jif_4.txt
94 include acsql/database/table_definitions/wfc_jif_5.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'),

```

```

154 ]
docs/source/database.rst

1 =====
2 Database
3 =====
4
5 The ``database`` subpackage contains various modules for constructing and interacting with the ``acsqll``
6 Database.
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 Indices and tables
22 -----
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 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 data_containers
14 -----
15 .. automodule:: website.data_containers
16     :members:
17     :undoc-members:
18     :show-inheritance:
19
20 form_options
21 -----
22 .. automodule:: website.form_options
23     :members:
24     :undoc-members:
25     :show-inheritance:
26
27 query_form
28 -----
29 .. automodule:: website.query_form
30     :members:
31     :undoc-members:
32     :show-inheritance:
33
34 query_lib
35 -----
36 .. automodule:: website.query_lib
37     :members:
38     :undoc-members:
39     :show-inheritance:
40

```

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
32         from acsq1.database.database_interface import base
33         from acsq1.database.database_interface import engine
34         from acsq1.database.database_interface import session
35         from acsq1.database.database_interface import Master
36         from acsq1.database.database_interface import Datasets
37         from acsq1.database.database_interface import <header_table>
38
39 Dependencies
40 -----
41     External library dependencies include:
42
43     - ``acsq1``
44     - ``pymysql``
45     - ``sqlalchemy``
46 """

```

```

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: {}:{}'
114                             .format(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
122 def get_special_column(keyword):
123     """Treat specific keywords separately.
124

```

```

125 Parameters
126 -----
127 keyword : str
128     The header keyword.
129
130 Returns
131 -----
132 Column : obj
133     A SQLAlchemy Column object for the given ``keyword``.
134 """
135
136 if keyword in ['RULEFILE', 'PROPTTL1', 'TARDESCR', 'QUALCOM2']:
137     return Column(String(500))
138 elif keyword in ['FWERROR', 'FW2ERROR']:
139     return Column(String(100))
140 else:
141     return Column(String(50))
142
143
144 def load_connection(connection_string):
145     """Return ``session``, ``base`` and ``engine`` objects for
146     connecting to the ``acsqll`` database.
147
148     Create an ``engine`` using an given ``connection_string``. Create a
149     ``base`` class and ``session`` class from the ``engine``. Create an
150     instance of the ``session`` class. Return the ``session``,
151     ``base`` and ``engine`` instances.
152
153     Parameters
154 -----
155     connection_string : str
156         The connection string to connect to the ``acsqll`` database. The
157         connection string should take the form:
158         ``dialect+driver://username:password@host:port/database``
159
160     Returns
161 -----
162     session : session object
163         Provides a holding zone for all objects loaded or associated
164         with the database.
165     base : base object
166         Provides a base class for declarative class definitions.
167     engine : engine object
168         Provides a source of database connectivity and behavior.
169 """
170
171     engine = create_engine(connection_string, echo=False, pool_timeout=100000)
172     base = declarative_base(engine)
173     Session = sessionmaker(bind=engine)
174     session = Session()
175
176     return session, base, engine
177
178
179 session, base, engine = load_connection(SETTINGS['connection_string'])
180
181
182 def orm_factory(class_name):
183     """Create a SQLAlchemy ORM Classes with the given ``class_name``.
184
185     Parameters
186 -----
187     class_name : str
188         The name of the class to be created
189
190     Returns
191 -----
192     class : obj
193         The SQLAlchemy ORM
194 """
195
196     data_dict = {}
197     data_dict['__tablename__'] = class_name.lower()
198     data_dict['rootname'] = Column(String(8), ForeignKey('master.rootname'),
199                                     primary_key=True, index=True,
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
23     - ``acsq1``

```

```

24     - ``numpy``
25     - ``stak`` (`https://github.com/spacetelescope/stak`)
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.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' \
61                           .format(detector, ftype))
62
63     for ext in file_exts[ftype]:
64
65         filename = 'table_definitions/{}_{:.txt}'.format(detector,
66                                                       ftype, ext)
67         hsel = Hselect(all_files, '*', extension=(ext,))
68
69         print('Making file {}'.format(filename))
70         with open(filename, 'w') as f:
71             for col in hsel.table.itercols():
72                 column_name = col.name
73                 if column_name in ['ROOTNAME', 'Filename', 'FILENAME', 'Ext']:
74                     continue
75                 elif col.dtype in [np.dtype('S68'), np.dtype('S80')]:
76                     ptype = 'String'
77                 elif col.dtype in [np.int64]:
78                     ptype = 'Integer'
79                 elif col.dtype in [bool]:
80                     ptype = 'Bool'
81                 elif col.dtype in [np.float64]:
82                     ptype = 'Float'
83                 else:
84                     print('Could not find type match: {}:{}' .format(
85                           column_name, col.dtype))
86
87                 # If the column has a hyphen, switch it to underscore
88                 if '-' in column_name:
89                     column_name = column_name.replace('-', '_')
90
91                 f.write('{}, {}\n'.format(column_name, ptype))
92
93
94 if __name__ == '__main__':
95
96     make_tabledefs('wfc')
97     make_tabledefs('sbc')
98     make_tabledefs('hrc')

```

acsql/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
39     query = queries.filter_distribution()
40
41 Each function will print the results to the screen, but the user
42 may also perform the query and handle the results themselves, e.g.:
43 :::
44
45     results = query.all()
46
47 Dependencies
48 -----
49 External library dependencies include:
50
51 - ``acsq1``
52 - ``sqlalchemy``
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').
76
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
171     query = session.query(WFC_raw_0.rootname, WFC_raw_0.filename,
172                           WFC_raw_0.targname) \
173                           .filter(WFC_raw_0.targname == targname)
174     query_results = query.all()
175
176     print('\nQuery performed:\n\n{}\n'.format(str(query)))
177
178     for result in query_results:
179         print(result)
180
181     return query
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
205     calibration_keyword_obj = getattr(WFC_raw_0, calibration_keyword)
206     query = session.query(WFC_raw_0.filename) \
207                           .filter(calibration_keyword_obj == value)
208     query_results = query.all()
209
210     print('\nQuery performed:\n\n{}\n'.format(str(query)))
211
212     for result in query_results:
213         print(result[0])
214
215     return query
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     Returns
310     -----
311     query : obj
312         The query object that contains attributes and methods for
313         performing the query.
314
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     logfile: The path to an ``acsqll.ingest.ingest.py`` log to be
35         used to determine new header keywords.
36
37 Dependencies
38 -----
39 External library dependencies include:
40
41     - ``acsqll``
42     - ``numpy``
43     - ``stak`` (''https://github.com/spacetelescope/stak'')
44
45 Notes
46 -----
47     The ``stak.Hselect`` dependency still depends on Python 2 at the
48     time of this writing.
49 """
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 '{}, {},'
128                 .format(keyword, table) in f][0]
129     test_file = test_file.split(',')[0]
130
131     # Use hselect to determine data type
132     try:
133         hsel = Hselect(test_file, keyword, extension=(int(table[-1]),))
134         dtype = hsel.table[keyword].dtype
135     except:
136         print('Cannot determine datatype for {}. Defaulting to String'
137             .format(keyword))
138         dtype = np.dtype('S80')
139
140     if dtype in [np.dtype('S68'), np.dtype('S80')]:
141         col_type = 'String'
142         db_type = 'VARCHAR(50)'
143     elif dtype in [np.int64]:
144         col_type = 'Integer'
145         db_type = 'INTEGER'
146     elif dtype in [bool]:
147         col_type = 'Bool'
148         db_type = 'BOOLEAN'
149     elif dtype in [np.float64]:
150         col_type = 'Float'
151         db_type = 'FLOAT'
152     else:
153         print('Could not find type match: {}:{}' .format(keyword, dtype))
154
155     # Update the appropriate table definitions file
156     tabledefs_file = 'table_definitions/{}.txt'.format(table.lower())
157     with open(tabledefs_file, 'r') as f:
158         data = f.readlines()
159     existing_keywords = [item.strip().split(',') [0] for item in data]
160     with open(tabledefs_file, 'a') as f:
161         if keyword not in existing_keywords:
162             f.write('{} , {} \n'.format(keyword, col_type))
163     print('Updated {} with {}'.format(tabledefs_file, keyword))
164
165     # Add ALTER TABLE command to list of commands
166     command = 'ALTER TABLE {} ADD {} {} ;' .format(table.lower(),
167                                                     keyword.lower(), db_type)
168     command_list.append(command)
169
170 # Print out the ALTER TABLE commands
171 print('\n\nALTER TABLE commands to execute for database:\n')
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 ``acsq1`` 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     try:
123         insert_obj.execute(data_dict)
124     except IntegrityError as e:
125         logging.warning('{}: Unable to insert {} into datasets table: {}'\
126                         .format(file_dict['full_rootname'], file_dict['basename'], e))
127
128 # If there are results, add the filename to the existing entry
129 else:
130     data_dict = query.one().__dict__
131     del data_dict['sa_instance_state']
132     data_dict[file_dict['filetype']] = file_dict['basename']
133     try:
134         query.update(data_dict)
135     except IntegrityError as e:
136         logging.warning('{}: Unable to update {} in datasets table: {}'\
137                         .format(file_dict['full_rootname'], file_dict['basename'], e))
138
139 session.commit()
140 session.close()
141 engine.dispose()
142
143 logging.info('{}: Updated datasets table for {}'\
144                 .format(file_dict['rootname'], file_dict['filetype']))
145
146

```

```

147 def update_header_table(file_dict, ext):
148     """Insert/update an entry for the file in the appropriate header
149     table (e.g. ``wfc_raw_0``).
150
151     The header table that get updated depend on the detector, filetype,
152     and extension.
153
154     Parameters
155     -----
156     file_dict : dict
157         A dictionary containing various data useful for the ingestion
158         process.
159     ext : int
160         The header extension.
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 = "{}_{}_{}{}".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 {}'.format(
200                         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(
207                         file_dict['rootname'], table))
208
209         except VerifyError as e:
210             logging.warning('\tUnable to insert {} into {}: {}'.format(
211                             file_dict['rootname'], table, e))
212
213
214 def update_master_table(rootname_path):
215     """Insert/update an entry in the ``master`` table for the given
216     file.
217
218     Parameters
219     -----
220     rootname_path
221         The path to the rootname directory in the MAST cache.
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'),
235                  'proposal_type': proposal_type}
236     insert_or_update('Master', data_dict)
237     logging.info('{}: Updated master table.'.format(rootname))
238
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))

```

acsqll/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 ``acsq1.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 acs1.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 acs1.utils import utils
43 from acs1.utils.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 'jit' 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 beleived 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`` form.
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()))[file_dict['filetype']]
178
179 # JPEG related keywords
180 if file_dict['filetype'] in ['raw', 'flt', 'flc']:
181     file_dict['jpg_filename'] = file_dict['basename'].replace('.fits', '.jpg')
182     file_dict['jpg_dst'] = os.path.join(SETTINGS['jpeg_dir'], file_dict['proposid_int'], file_dict['jpg_filename'])
183     file_dict['thumbnail_filename'] = file_dict['basename'].replace('.fits', '.thumb')
184     file_dict['thumbnail_dst'] = os.path.join(SETTINGS['thumbnail_dir'], file_dict['proposid_int'], file_dict['thumbnail_filename'])
185 else:
186     file_dict['jpg_filename'] = None
187     file_dict['jpg_dst'] = None
188     file_dict['thumbnail_filename'] = None
189     file_dict['thumbnail_dst'] = None
190
191 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 observations.
6
7 Authors
8 -----
9   Matthew Bourque
10
11 Use
12 ---
13   This module is intended to be imported and used by
14   ``acsq1.ingest.ingest.py`` as such:
15   ::

16       from acsq1.ingest.make_jpeg import make_jpeg
17       make_jpeg(file_dict)
18
19 Dependencies
20 -----
21   External library dependencies include:
22
23   - ``astropy``
24   - ``numpy``
25   - ``PIL``
26 """
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.

```

```

54 """
55
56     logging.info(' {}: Creating JPEG'.format(file_dict['rootname']))
57
58     hdulist = fits.open(file_dict['filename'], mode='readonly')
59     data = hdulist[1].data
60
61     # If the image is full-frame WFC, add on the other extension
62     if len(hdulist) > 4 and hdulist[0].header['detector'] == 'WFC':
63         if hdulist[4].header['EXTNAME'] == 'SCI':
64             data2 = hdulist[4].data
65             height = data.shape[0] + data2.shape[0]
66             width = data.shape[1]
67             new_array = np.zeros((height, width))
68             new_array[0:int(height/2), :] = data
69             new_array[int(height/2):height, :] = data2
70             data = new_array
71
72     # Clip the top and bottom 1% of pixels.
73     top = np.percentile(data, 99)
74     data[data > top] = top
75     bottom = np.percentile(data, 1)
76     data[data < bottom] = bottom
77
78     # Scale the data.
79     data = data - data.min()
80     data = (data / data.max()) * 255.
81     data = np.flipud(data)
82     data = np.uint8(data)
83
84     # Create parent JPEG directory if necessary
85     jpg_dir = os.path.dirname(file_dict['jpg_dst'])
86     if not os.path.exists(jpg_dir):
87         os.makedirs(jpg_dir)
88         logging.info(' {}: Created directory {}'.format(file_dict['rootname'], jpg_dir))
89
90     # Write the image to a JPEG
91     image = Image.fromarray(data)
92     image.save(file_dict['jpg_dst'])
93
94     # Close the hdulist
95     hdulist.close()

```

acsq1/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 ``acsq1`` 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     ``acsq1.ingest.ingest.py`` as such:
17     ::

18         from acsq1.ingest.make_thumbnail import make_thumbnail
19         make_thumbnail(file_dict)
20
21 Dependencies
22 -----
23     External library dependencies include:
24
25     - ``PIL``
26
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 {}{}'.format(
54                 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 acsq1.database.database_interface import Master, session
48 from acsq1.ingest.ingest import ingest
49 from acsq1.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 ``acsq1``
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 ``acsq1`` 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 ``acsq1`` 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 acsq1 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]
110     else:
111         rootnames = get_rootnames_to_ingest()
112
113     pool = multiprocessing.Pool(processes=SETTINGS['ncores'])
114     filetypes = [filetype for item in rootnames]
115     mp_args = [(rootname, filetype) for rootname, filetype in zip(rootnames, filetypes)]
116     pool.starmap(ingest, mp_args)
117
118     logging.info('Process Complete.')
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 acsq1 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),\n            '{} does not exist.'.format(args.ingest_filelist)
186
187
188 if __name__ == '__main__':
189
190     module = os.path.basename(__file__).strip('.py')
191     setup_logging(module)
192
193     args = parse_args()
194     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 acsql
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', 'flc', '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
130     SETTINGS = get_settings()
131
132     # Configure logging
133     timestamp = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M')
134     filename = '{0}_{1}.log'.format(module, timestamp)
135     logfile = os.path.join(SETTINGS['log_dir'], filename)
136     logging.basicConfig(
137         filename=logfile,
138         format='%(asctime)s %(levelname)s: %(message)s',
139         datefmt='%m/%d/%Y %H:%M:%S',
140         level=logging.INFO)
141
142     # Log environment information
143     logging.info('User: {0}'.format(getpass.getuser()))
144     logging.info('System: {0}'.format(socket.gethostname()))
145     logging.info('Python Version: {0}'.format(sys.version.replace('\n', '')))
146     logging.info('Python Path: {0}'.format(sys.executable))
147     logging.info('NumPy Version: {0}'.format(numpy.__version__))
148     logging.info('NumPy Path: {0}'.format(numpy.__path__[0]))
149     logging.info('Astropy Version: {0}'.format(astropy.__version__))
150     logging.info('Astropy Path: {0}'.format(astropy.__path__[0]))
151     logging.info('SQLAlchemy Version: {0}'.format(sqlalchemy.__version__))
152     logging.info('SQLAlchemy Path: {0}'.format(sqlalchemy.__path__[0]))
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(acssql.database.database_interface, table)
205     session, base, engine = acssql.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/acssql_webapp.py

```

1 #! /usr/bin/env python
2
3 """This module serves as the ``acssql`` web application, which allows
4 users to view image data and interact with the ``acssql`` 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 acssql_webapp.py
24     <go to localhost:5000 in a browser>
25
26 Dependencies
27 -----
28
29     - ``'acssql'``
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 acsql.utils.utils import SETTINGS
42 from acsql.website.data_containers import get_view_image_dict
43 from acsql.website.data_containers import get_view_proposal_dict
44 from acsql.website.data_containers import get_view_query_results_dict
45 from acsql.website.query_form import get_query_form
46 from acsql.website.query_lib import generate_csv
47 from acsql.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     ``acsql`` 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 ``acsq`` 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 IPPPSSOOT 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/jpegs/', data_dict['proposal_id'])
62 thumb_proposal_path = os.path.join('static/img/thumbnails/', data_dict['proposal_id'])
63
64 data_dict['jpegs'] = sorted(glob.glob(os.path.join(jpeg_proposal_path, '*flt.jpg')))
65 data_dict['thumbs'] = 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.)
72     to the ``data_dict`` by querying the ``acsqll`` database.
73
74     Parameters
75     -----
76     data_dict : dict
77         A dictionary containing data used to render a webpage.
78
79     Returns
80     -----
81     data_dict : dict
82         A dictionary containing data used to render a webpage.
83
84     """
85
86     session = getattr(database_interface, 'session')
87
88     results = []
89     for rootname in data_dict['rootnames']:
90
91         detector = session.query(Master.detector) \
92             .filter(Master.rootname == rootname).one()[0]
93
94         table = getattr(database_interface, '{}_raw_0'.format(detector))
95         result = session.query(
96             table.aperture, table.exptime, table.filter1, table.filter2,
97             table.targname, table.date_obs, table.time_obs, table.expsstart,
98             table.expflag, table.quality, table.ra_targ, table.dec_targ,
99             table.pr_inv_f, table.pr_inv_l).filter(table.rootname == rootname).one()
100        result = [item for item in result]
101        result.append(detector)
102        results.append(result)
103
104    session.close()
105
106    # Parse the results
107    data_dict['detectors'] = [detector for item in results]
108    data_dict['apertures'] = [item[0] for item in results]
109    data_dict['exptimes'] = [item[1] for item in results]
110    data_dict['filter1s'] = [item[2] for item in results]
111    data_dict['filter2s'] = [item[3] for item in results]
112    data_dict['targnames'] = [item[4] for item in results]
113    data_dict['dateobss'] = [item[5] for item in results]
114    data_dict['timeobss'] = [item[6] for item in results]
115    data_dict['expstarts'] = [item[7] for item in results]
116    data_dict['expflags'] = [item[8] for item in results]
117    data_dict['qualities'] = [item[9] for item in results]
118    data_dict['ras'] = [item[10] for item in results]
119    data_dict['decs'] = [item[11] for item in results]
120    data_dict['pi_firsts'] = [item[12] for item in results]
121    data_dict['pi lasts'] = [item[13] for item in results]
122
123    return data_dict
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
145     data_dict['buttons'] = {}
146     data_dict['buttons']['detector'] = sorted(set(data_dict['detectors']))
147     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
182         status_string = req.content.decode()
183         data_dict['proposal_title'] = html.unescape(status_string.\
184             split('<b>Title:</b>')[1].\\
185             split('<br>')[0])
186         data_dict['cycle'] = html.unescape(status_string.\\
187             split('<b>Cycle:</b>')[1].\\
188             split('<br>')[0])
189         data_dict['schedule'] = html.unescape(status_string.\\
190             split('/proposal-help-HST.html#')[2].\\
191             split('>')[0])
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['qualitys'][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}

```

```

297     if not image_dict['first']:
298         image_dict['prev'] = {'proposal': image_dict['proposal_id'], 'filename': image_dict['filenames'][image_dict['index'] - 1], 'fits_type': fits_type}
299
300     # Determine other available JPEGs for given observation
301     jpeg_types = glob.glob(jpeg_path_abs.replace('{}.jpg'.format(fits_type), '*.jpg'))
302     jpeg_types = [os.path.basename(item).split('_')[-1].split('.')[0] for item in jpeg_types]
303     jpeg_types = [item for item in jpeg_types if item.upper() != image_dict['fits_type']]
304     image_dict['available_jpegs'] = {}
305     for jpeg_type in jpeg_types:
306         image_dict['available_jpegs'][jpeg_type] = image_dict['view_url'].replace(fits_type, jpeg_type)
307
308     # For downloading the files
309     # image_dict['proposal_name'] = image_dict['filename'][0:4]
310     # image_dict['fits_links']['FLT'] = os.path.join(
311     #     SETTINGS['filesystem'],
312     #     image_dict['proposal_name'],
313     #     image_dict['filename'],
314     #     '{}_flt.fits'.format(image_dict['filename'])))
315
316     return image_dict
317
318
319 def get_view_proposal_dict(proposal):
320     """Return a dictionary containing data used for the
321     ``/archive/<proposal>/`` webpage.
322
323     Parameters
324     -----
325     proposal : str
326         The proposal number (e.g. ``12345``).
327
328     Returns
329     -----
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['jpegs']]
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'] = ['/{}/{}/{}'.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

```

acsqsql/website/form_options.py

```

1 """Provides a dictionary containing ``acsqsql`` database query form data
2 for use by the ``acsqsql`` 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 acsqql.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-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', 'G800I', '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 inteded 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 """
31
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 acsq1.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``.
```

```

55 """
56
57     op = operator_form
58     val1 = DateField('Date Observed',
59                     [validators.Optional()],
60                     description='YYYY-MM-DD',
61                     format='%Y-%m-%d')
62     val2 = DateField('dateobs2',
63                     [validators.Optional()],
64                     description='YYYY-MM-DD',
65                     format='%Y-%m-%d')
66
67
68 class ExptimeForm(Form):
69     """Creates a ``ExptimeForm`` object that allows for ``exptime``
70     input in a form field.
71
72     Parameters
73     -----
74     Form : obj
75         The ``Form`` object from ``wtforms``.
76
77     op = operator_form
78     val1 = DecimalField('Exposure Time', [validators.Optional()])
79     val2 = DecimalField('exptime2', [validators.Optional()])
80
81
82 class MultiCheckboxField(SelectMultipleField):
83     """A multiple-select, except displays a list of checkboxes.
84
85     Parameters
86     -----
87     SelectMultipleField : obj
88         The ``SelectMultipleField`` object from ``wtforms``.
89
90
91     widget = widgets.ListWidget(prefix_label=False)
92     option_widget = widgets.CheckboxInput()
93
94
95     def is_field_value(form, fieldname, value, negate=False):
96         """Helper function to check if the given field in the given form is
97         of a specified value.
98
99         Parameters
100        -----
101        form: obj
102            The form to test on
103        fieldname : str
104            The fieldname to test value against. If not found an Exception
105            is raised.
106        value : str
107            Value to test for.
108        negate : boolean
109            True/False to invert the result.
110
111
112        field = form._fields.get(fieldname)
113        if field is None:
114            raise Exception('Invalid field "%s"' % fieldname)
115        test = value == field.data
116        test = not test if negate else test
117
118        return test
119
120
121 class RequiredIf(validators.Required):
122     """Custom validator to enforce requires only if another field
123     matches a specified value. the ``negate`` allows for inverting
124     the result.
125
126     Parameters
127     -----
128     validators.Required : obj
129         The ``validators.Required`` object from ``wtforms``.
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
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 ``acsqll`` 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
153     # 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 # Combine data returned from fields with operator dropdowns
208 for operator_key in operator_keys:
209     operator_dict = {}
210     for key, value in list(query_form_dict.items()):
211         if key == operator_key + '-op':
212             operator_dict['op'] = value[0]
213         elif key == operator_key + '-val1':
214             operator_dict['val1'] = value[0]
215         elif key == operator_key + '-val2':
216             operator_dict['val2'] = value[0]
217     if len(operator_dict) > 1:
218         query_form_dict[operator_key] = operator_dict
219
220 return query_form_dict
221
222
223
224 def generate_csv(output_columns, results):
225     """Create a CSV file of the database query output.
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
309     # Put results in a dictionary
310     query_results_dict = {}
311     query_results_dict['num_results'] = num_results
312     query_results_dict['query_results'] = query_results
313     query_results_dict['output_format'] = output_format
314     query_results_dict['output_columns'] = output_columns
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     ``acsq1`` database.
322
323     Returns
324     -----
325     session : obj
326         A ``SQLAlchemy`` ``session`` object which serves as a
327         connection to the ``acsq1`` database.
328
329
330     engine = create_engine(SETTINGS['connection_string'], echo=False, pool_timeout=100000)
331     Session = sessionmaker(bind=engine)
332     session = Session()
333
334     return session
335
336
337 def _merge_query(wfc_query, hrc_query, sbc_query):
338     """Merge the results from the queries from each table
339
340     Parameters
341     -----
342     wfc_query : obj
343         The ``SQLAlchemy`` ``query`` object for request columns of the
344         WFC table.
345     hrc_query : obj
346         The ``SQLAlchemy`` ``query`` object for request columns of the
347         HRC table.
348     sbc_query : obj
349         The ``SQLAlchemy`` ``query`` object for request columns of the
350         SBC table.
351
352     Returns
353     -----
354     query : obj
355         The ``SQLAlchemy`` ``query`` object with merging applied.
356
357
358     # Turn off the query if the table is not needed
359     if wfc_query:
360         if str(wfc_query.statement).find('WHERE') == -1:

```

```

361         wfc_query = False
362     if hrc_query:
363         if str(hrc_query.statement).find('WHERE') == -1:
364             hrc_query = False
365     if sbc_query:
366         if str(sbc_query.statement).find('WHERE') == -1:
367             sbc_query = False
368
369     # If combination needed
370     if wfc_query and hrc_query and sbc_query:
371         query = wfc_query.union_all(hrc_query, sbc_query)
372     elif wfc_query and not hrc_query and sbc_query:
373         query = wfc_query.union_all(sbc_query)
374     elif wfc_query and hrc_query and not sbc_query:
375         query = wfc_query.union_all(hrc_query)
376     elif not wfc_query and hrc_query and sbc_query:
377         query = hrc_query.union_all(sbc_query)
378
379     # If no combination needed
380     elif wfc_query and not hrc_query and not sbc_query:
381         query = wfc_query
382     elif not wfc_query and hrc_query and not sbc_query:
383         query = hrc_query
384     elif not wfc_query and not hrc_query and sbc_query:
385         query = sbc_query
386
387     else:
388         query = None
389
390     return query

```

acsq1/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 %}

```

acsq1/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()}}
12    </button>
13    <button type="button" id="check_all_{{field.name}}" class="btn">Select all {{field.label.text.lower()}}
14    }s</button>
15    <button type="button" id="uncheck_all_{{field.name}}" class="btn">Deselect all {{field.label.text.lower()}}
16    }s</button>
17 {% endif %}
18
19 {% elif field.type == 'SelectMultipleField' %}
20 <div class="{{ field.description[-5:] }} formfield">
21     <h5>{{ field.label.text }}</h5>
22     {{ field(style="width:100%;", placeholder=field.description[:-6], **kwargs)|safe }}

```

```

21  {% else %}
22  <div class="{{ field.description[-5:] }} formfield">
23    <h5>{{ field.label.text }}</h5>
24    {{ field(style="width:calc(100% - 1em);width:-moz-calc(100% - 1em);
25      width:-webkit-calc(100% - 1em);width:-o-calc(100% - 1em);width:-ms-calc(100% - 1em);",
26      placeholder=field.description[:-6], **kwargs)|safe }}
27  {% endif %}
28 </div>
29 {% endmacro %}
30
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 }}&ampnbsp
38     {% elif subfield.name.endswith('val1') %}
39       {{ subfield(style="width:calc(100% - 5.75em);display:inline;", placeholder=subfield.description, **kwargs)|safe }}
40     {% elif subfield.name.endswith('val2') %}
41       {{ subfield(style="width:calc(100% - 5.75em);display:inline;float:right;margin-right:3px;display:none
42      ;", placeholder=subfield.description, **kwargs)|safe }}
43     {% endif %}
44   {% endfor %}
45 </div>
46 {% 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     <script src="/static/js/modernizr-2.6.2.min.js"></script>
29     <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js"></script>
30     <script>window.jQuery || document.write('<script src="/static/js/jquery-1.10.1.min.js"><\!></script>')
31   </script>
32
33 </head>
34 <body>
35   <div class="navbar navbar-inverse navbar-fixed-top">
36     <div class="navbar-inner">
37       <div class="container">
38         <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
39           <span class="icon-bar"></span>
40           <span class="icon-bar"></span>
41           <span class="icon-bar"></span>
42         </a>
43         <a class="brand pull-left no-overflow" href="/"> ACS Quicklook</a>
45         <div class="nav-collapse collapse">
46           <ul class="nav pull-right">
47             <li><a href="/database/"><font size="4">Database</font></a></li>
48             <li><a href="/archive/"><font size="4">Archive</font></a></li>
49             <li><a href="https://github.com/spacetelescope/acsql" target="_blank"><font
50               size="4">GitHub</font></a></li>
51             <li><a href="http://acsql.readthedocs.io" target="_blank"><font size="4">
52               Documentation</font></a></li>
53           </ul>
54         </div>
55       </div>
56     </div>
57
58     <div class="container">
59
60       {%- block content %} {%- endblock %}
61
62       <hr>
63
64       <footer>
65         <p>Space Telescope Science Institute</p>
66       </footer>
67
68     </div>
69
70     <script src="/static/js/bootstrap.min.js"></script>
71     <script src="/static/js/plugins.js"></script>
72     <script src="/static/js/main.js"></script>
73   </body>
74 </html>

```

acsql/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 acsql 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
27  </form>

```

```

25     { % endfor %}
26
27     <div class="span12" style="margin-top:2em;">
28         <center>
29             <button type="submit" value="Submit" class="btn btn-lg btn-primary">Submit</button>&ampnbsp&
nbsp;&nbsps;
30             <button type="reset" value="Reset" class="btn btn-lg btn-danger">Reset</button>
31         </center>
32     </div>
33     </form>
34 </div>
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     { % endif %}
57 { % endfor %}
58 { % 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 { % 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                 {% for col in output_columns %}
59                     <th>{{ col }}</th>
60                 {% endfor %}
61             </thead>
62             <tbody>
63                 {% for line in results %}
64                     <tr>
65                         {% for value in line %}
66                             <td>{{ value }}</td>
67                         {% endfor %}
68                     </tr>
69                 {% endfor %}
70             </tbody>
71         </table>
72     </div>
73     {% endif %}
74
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 >
```

```

82 <script>
83 $(document).ready(function() {
84     $('#results').DataTable( {
85         "lengthMenu": [[10, 25, 50, 100, 500, 1000, -1], [10, 25, 50, 100, 500, 1000, "All"]],
86         stateSave: true,
87         "language": {
88             "lengthMenu": "Display _MENU_ results per page",
89             "zeroRecords": "Nothing found",
90             "info": "Showing page _PAGE_ of _PAGES_",
91             "infoEmpty": "No results available",
92             "infoFiltered": "(filtered from _MAX_ total results)"
93         }
94     } );
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 <script>
24 $(document).ready(function() {
25     $('#scriptstatus').DataTable( {
26         "paging": false,
27         "bFilter": false,
28         stateSave: true
29     } );
30 })
31
32 </script>

```

```

29     } );
30 });
31 </script>
32 {%- 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</b> {{image_dict.proposal_title}}</i><br>
16             Image {{image_dict.page}} of {{image_dict.num_images}}<br>
17
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
28             <!-- Additional information links -->
29             <br>
30             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>
31             {% if image_dict.available_jpegs %}
32                 View JPEG for
33                 {% for key, link in image_dict.available_jpegs.items() %}
34                     <a href="/archive/{{image_dict.proposal_id}}/{{image_dict.filename}}/{{key}}"> {{key}}</a>
35                 {% endfor %}
36             {% endif %}
37             </p>
38     </div>
39     <div class="span3">
40         <p>
41             <b>Expstart:</b> {{image_dict.expstart}}<br>
42             <b>Filter:</b> {{image_dict.filter1}}/{{image_dict.filter2}}<br>
43             <b>Aperture:</b> {{image_dict.aperture}}<br>
44             <b>Exposure time:</b> {{image_dict.exptime}}<br>
45         </p>
46     </div>
47     <div class="span3">
48         <p>
49             <b>Expflag:</b> {{image_dict.expflag}}, Quality: {{image_dict.quality}}<br>
50             <b>Coordinates:</b> {{image_dict.ra}}, {{image_dict.dec}}<br>
51             <b>Target:</b> {{image_dict.targname}}<br>
52             <b>PI:</b> {{image_dict.pi_first_name}} {{image_dict.pi_last_name}}<br>
53         </p>
54     </div>
55 </div>
56
57     <!-- Other available JPEGs -->
58
59
60     <!-- Image -->
61     <div class="row">
62         <div class="fleximage">
63             {% if image_dict.image %}
64                 <div id="wrapper" style="width:100%; text-align:center">
65                     
66                 </div>
67             {% else %}
68                 <br><br><br>

```

```

69         <center><strong><p>There is no JPEG to display for FITS type {{image_dict.fits_type}}</p></strong></center>
70     {%
71     % endif %
72   </div>
73 </div>
74 <div style="height:1em;"></div>
75
76 <!-- Enable left and right keystrokes -->
77 <script type="text/javascript">
78   {%
79   % if not image_dict.last %}
80   $("body").keydown(function(e) {
81     if(e.keyCode == 39) { // right
82       window.location = "{{ url_for('view_image',**image_dict.next) }}";
83     }
84   });
85   {%
86   % if not image_dict.first %}
87   $("body").keydown(function(e) {
88     if(e.keyCode == 37) { // left
89       window.location = "{{ url_for('view_image',**image_dict.prev) }}";
90     }
91   });
92   {%
93   % endif %
94 </script>
95
96 {%
97   % endblock %
98 }
```

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   {%
12   % if proposal_dict.num_images %}
13   <h3>Proposal {{proposal_dict.proposal_id}} {%
14   % if proposal_dict.proposal_title %}<small>{{proposal_dict.proposal_title}}{%
15   % endif %}</small></h3>
16   {%
17   % if proposal_dict.buttons %}
18   <form id="filtering" style="margin-bottom:0;">
19
20     <!-- Filter -->
21     <div id="filterOptions" class="container">
22       <div class="form-group">
23         {%
24         % if proposal_dict.cycle %}Cycle {{proposal_dict.cycle}}, {{proposal_dict.schedule}}{%
25         br>{%
26         % endif %}
27         {{proposal_dict.num_images}} images{%
28         % if proposal_dict.num_visits %}, {{proposal_dict.
29         num_visits}} visits{%
30         % endif %}
31         <br>
32         <a href="{{proposal_dict.status_page}}" target="_blank">Proposal info</a>
33       </div>
34       {%
35       % for key,buttonlist in proposal_dict.buttons.items() %}
36       <div class="form-group">
37         <label for="{{key}}>Show only {{key}}:</label>
38         <select class="form-control" id="{{key}}>
39           <option value="all">All {{key}}s</option>
40           {%
41           % for b in buttonlist %}
42             <option value="{{b}}>{{b}}</option>
43           {%
44           % endfor %}
45         </select>
46       </div>
47       {%
48       % endfor %}
49     </div>
50
51     <!-- Sort By -->
52     <div class="row" id="sortOptions">
53       <div class="span6">
54         Sort by:&nbsp;&nbsp;
55         <label class="radio inline"><input type="radio" name="sort" id="unsort" checked>default
56       </label>
57         <label class="radio inline"><input type="radio" name="sort" id="sort-expstart">expstart
58       </label>
59     </div>
60   </div>
```

```

43         <label class="radio inline"><input type="radio" name="sort" id="sort-exptime">exptime</
44     </label>
45     </div>
46     <div class="span5 text-right">
47     {%
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"<
51     %}
52     </div>
53     <div class="span1 text-right">
54       <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
55     </div>
56   </div>
57 {%
58   endif %
59
60   <!-- Loading animation -->
61   <div id="loading">
62     <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
63       <div class="cssload-cssload-loader-line-wrap-wrap">
64         <div class="cssload-loader-line-wrap"></div>
65       </div>
66       <div class="cssload-cssload-loader-line-wrap-wrap">
67         <div class="cssload-loader-line-wrap"></div>
68       </div>
69       <div class="cssload-cssload-loader-line-wrap-wrap">
70         <div class="cssload-loader-line-wrap"></div>
71       </div>
72       <div class="cssload-cssload-loader-line-wrap-wrap">
73         <div class="cssload-loader-line-wrap"></div>
74       </div>
75       <div class="cssload-cssload-loader-line-wrap-wrap">
76         <div class="cssload-loader-line-wrap"></div>
77       </div>
78     <br>
79     Loading thumbnails...
80   </div>
81
82   <!-- Thumbnails -->
83   <div id="onload">
84     <div id="thumbnail-array">
85       {%
86         for i in range(proposal_dict.num_images) %
87           <div class="thumb" page="{{i+1}}" detector="{{proposal_dict.detectors[i]}}"
88             filter1="{{proposal_dict.filter1s[i]}} filter2="{{proposal_dict.filter2s[i]}}"
89             expstart="{{proposal_dict.exstarts[i]}} exptime="{{proposal_dict.extimes[i]}}"
90             targname="{{proposal_dict.targnames[i]}} proposid="{{proposal_dict.proposal_id}}"
91             visit="{{proposal_dict.visits[i]}} style="background:url(../../{{proposal_dict.thumbs[i]}}) no-repeat;">
92             <div class="overlay">
93               <strong>{{proposal_dict.filenames[i]}}</strong><br><font size="0.5">
94                 <b>Detector:</b> {{proposal_dict.detectors[i]}}<br>
95                 <b>Filter:</b> {{proposal_dict.filter1s[i]}} / {{proposal_dict.filter2s[i]}}
96               </div>
97               {%
98                 for j in range(proposal_dict.num_images) %
99                   <b>Exptime:</b> {{proposal_dict.extimes[i]}}<br>
100                  <b>Target:</b> {{proposal_dict.targnames[i]}}<br></font>
101                </div>
102                <a href="{{proposal_dict.viewlinks[i]}}" target="_blank">
103                  
104                </a>
105              {%
106                endfor %
107              }
108            </div>
109          {%
110            else %
111              <p>
112                <h5>There are no images to display.</h5>
113              </p>
114            %}
115          {%
116            endif %
117          }
118        </div>
119      </div>
120
121      <script type="text/javascript">
122        function preloader(){
123          document.getElementById("loading").style.display = "none";
124          document.getElementById("onload").style.display = "block";
125        }
126      </script>
127    
```

```

113     }
114     window.onload = preloader;
115 </script>
116
117 <script src="/static/js/tinysort.min.js"></script>
118 <script src="/static/js/thumbnails.js"></script>
119
120 {%- endblock %}

```

acsq1/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;">
15
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                 </div>
41                 <div class="span5 text-right">
42                     {% if thumbnail_dict.buttons|length > 2 %}
43                         Filter logic:&nbsp;&nbsp;
44                         <label class="radio inline"><input type="radio" name="logic" value="and" checked>"AND"<
45                     </label>
46                         <label class="radio inline"><input type="radio" name="logic" value="or">"OR"</label>
47                     {% endif %}
48                 </div>
49                 <div class="span1 text-right">
50                     <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
51                 </div>
52             </form>
53         {% endif %}
54
55         <!-- Loading animation -->
56         <div id="loading">
57             <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
58                 <div class="cssload-cssload-loader-line-wrap-wrap">
59                     <div class="cssload-loader-line-wrap"></div>
60                 </div>
61                 <div class="cssload-cssload-loader-line-wrap-wrap">
62                     <div class="cssload-loader-line-wrap"></div>
63                 </div>
64                 <div class="cssload-cssload-loader-line-wrap-wrap">
65                     <div class="cssload-loader-line-wrap"></div>
66                 </div>
67             </div>
68         </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]}}"
81 filter1="{{thumbnail_dict.filter1s[i]}}" filter2="{{thumbnail_dict.filter2s[i]}}" expstart="{{
82 thumbnail_dict.expstarts[i]}}" exptime="{{thumbnail_dict.exptimes[i]}}" targname="{{thumbnail_dict.
83 targnames[i]}}" proposid="{{thumbnail_dict.proposal_ids[i]}}" visit="{{thumbnail_dict.visits[i]}}"
84 style="background:url(..{{thumbnail_dict.thumbs[i]}}) no-repeat;">
85                  <div class="overlay">
86                      <strong>{{thumbnail_dict.filenames[i]}}</strong><br><font size="0.5">
87                          <b>Detector:</b> {{thumbnail_dict.detectors[i]}}<br>
88                          <b>Filter:</b> {{thumbnail_dict.filter1s[i]}} / {{thumbnail_dict.filter2s[i]
89 }}<br>
90                          <b>Exptime:</b> {{thumbnail_dict.exptimes[i]}}<br>
91                          <b>Target:</b> {{thumbnail_dict.targnames[i]}}<br></font>
92                  </div>
93                  <a href="{{thumbnail_dict.viewlinks[i]}}" target="_blank">
94                      
95                  </a>
96          {% endfor %}
97      </div>
98  </div>
99  {% else %}
100     <p>
101         <h5>There are no images to display.</h5>
102     </p>
103  {% endif %}
104 </div>
105
106 <script type="text/javascript">
107     function preloader(){
108         document.getElementById("loading").style.display = "none";
109         document.getElementById("onload").style.display = "block";
110     }
111     window.onload = preloader;
112 </script>
113
114 {% endblock %}

```