The Hubble Space Telescope Advanced Camera for Surveys Quicklook Application

Matthew Bourque[1], Sara Ogaz[1], Alex Viana[2], Meredith Durbin[3], and Norman Grogin[1]

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

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

1 Introduction

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

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

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

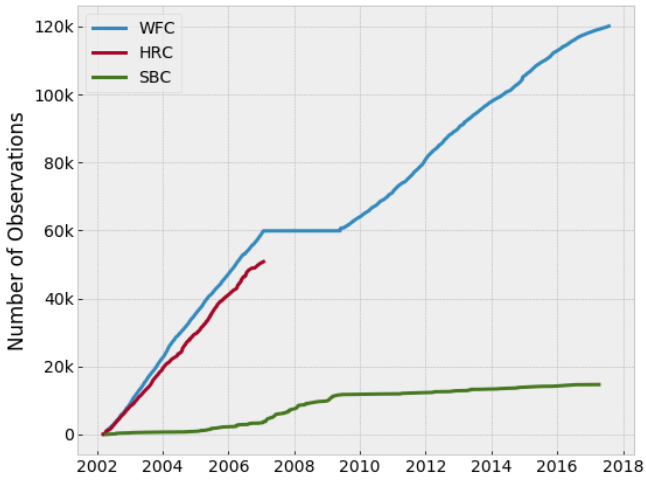


Fig. 1: *The number of observations over time for each of the three detectors on ACS.*

MAST, users can request and retrieve data for any publicly-available dataset via `ftp`, `sftp`, or DVD by mail[2]. Like is standard for other astronomical data, ACS data are stored using the Flexible Image Transport System (FITS) data format[5]. This data format has several unique characteristics, as will be discussed in section 1.1.

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

Manuscript received Month DD, YYYY

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

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

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

This paper outlines the acsql application as part of the Towson University Computer Science Masters Program Graduate Project. The remaining sections in this chapter discuss the motivation and use cases for this application, as well as details on the underlying data structure on which this project was built. Chapter 2 discusses how the acsql application compares to related work. Chapter 3 details the implementation of each acsql component. Chapter 4 outlines the results of the projectand 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 acsql, particularly the acsql database, is heavily dependent on the underlying data structure of ACS FITS files. As such, it is important for the reader to have a conceptual understanding of this data structure and thus the following sections are dedicated to providing an overview on the FITS data format and ACS-specific intricacies.

1.1.1 Filenames

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

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

Each <rootname> consists of nine unique alpha-numeric characters, and <filetype> is one of several possible three-character filetype options (discussed in section 1.1.4). For example, one ACS observation has the filename j6mf16lhq_raw.fits (Principle Investigator Gary Bernstein, observation date 2016-09-22). Each character in the 9-character rootname has a specific meaning, and is discussed in section 5.2 of the Introduction to the HST Data Handbooks[7]. The .fits extension at the end of the filename signifies that the file is of the FITS data format.

The unique 9-character rootname is associated with an individual, distinct ACS observation. However, it should be noted that only the first eight characters of the 9-character rootname are truly unique; the last character, which identifies the source of data transmission from the telescope[3], has several possible values. However, varying values of the last character do not associate with different observations. For this reason, as will be discussed in section 3.5, we adopt an 8-character rootname as a primary key for the acsql database tables to denote unique ACS observations.

1.1.2 FITS file structure

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

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

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

TABLE 1: ACS/WFC FITS file extensions

Extension	Purpose	Image Dimensions (pixels)	Data Type
0	Primary header	–	String
1	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

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

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

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

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

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

1.1.3 FITS file extension headers

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

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

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

            / DATA DESCRIPTION KEYWORDS

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

            / TARGET INFORMATION

TARGNAME= 'DARK                              ' / proposer's target name
RA_TARG =   0.000000000000E+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 investigat

            / 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: *A section of an example header, taken from the 0th extension of the file* `j6mf6lhq_raw.fits`.

1.1.4 FITS filetypes for ACS

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

- **raw** - the raw, uncalibrated data that comes directly from HST
- **flt** - nominally calibrated data
- **flc** - nominally calibrated data plus corrected for Charge Transfer Efficiency (CTE) deficits.
- **drz** - geometric distortion-corrected data
- **drc** - geometric distortion-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 results in the filetypes `raw`, `flt`, `jit`, `jif`, and `spt`.

1.2 Key Metadata

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

rootname - The 8-character unique rootname of the observation (e.g. `j9q15401`).

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

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

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

1.3 Motivation

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

1.3.1 Data retrieval latency

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

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

1.3.2 File I/O

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

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

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

1.3.3 Data redundancy

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

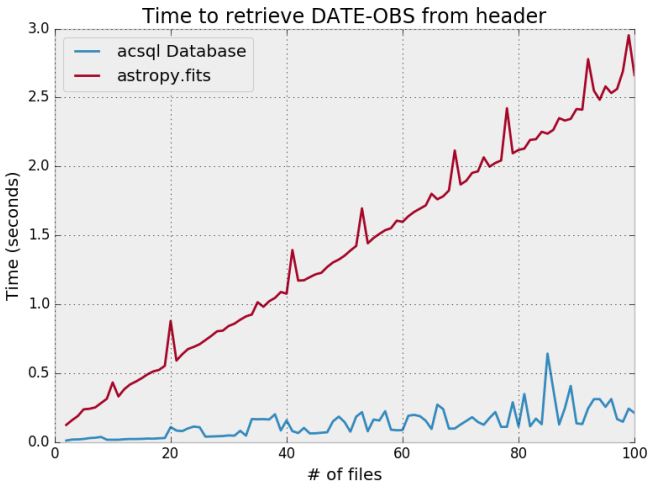


Fig. 3: *The amount of time it takes to retrieve the* `DATE-OBS` *header keyword from a varying number of ACS data files via reading the FITS file using the* `astropy.io.fits` *module (red) and querying the* `acsql` *database (blue).*

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

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

1.3.4 Data discovery

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

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

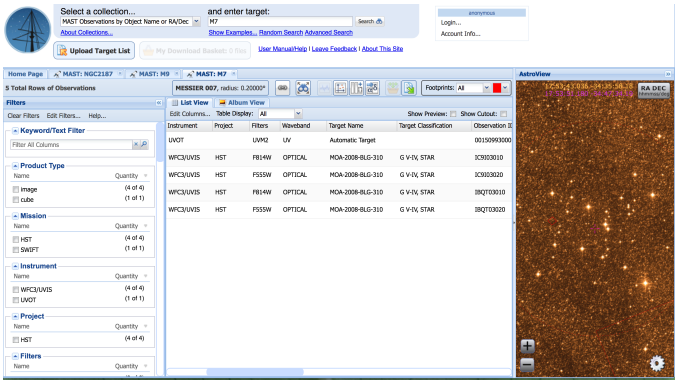


Fig. 4: *An example of a query to the MAST data discovery portal.*

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

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

1.3.5 Use Cases

The intended primary users of the `acsql` application are ACS instrument analysts that work for the ACS instrument team at STScI. ACS instrument analysts use ACS data frequently in their day-to-day work, using it to help calibrate, monitor, and characterize various aspects of the ACS instrument. On any given day, an instrument analyst may need to analyze data from a number of observing modes, time periods, or filetypes. In this regard, the `acsql` application aims to limit the amount of time and effort required by the

analysts to discover and access the data that they need to perform such work. It should be noted, though, that the acsql application is not necessarily limited to this user base; since the application contains only public data, there is potential for users external to STScI, such as scientific researchers at other academic institutions.

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

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

Use case 2: Determine which datasets exist in the ACS archive for a given set of observational parameters: The application shall allow users to determine which 9-character rootnames exist for a defined set of observational parameters via ACS header keywords.

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

Use case 4: Programmatically analyze images across custom datasets: The application shall allow users to 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 acsql 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 acsql does, including a filesystem that stores all WFC3 data on disk, a relational database that stores WFC3 FITS header information, and a web application for user interactivity with the system[11][12]. The wfc3ql application has been developed by the WFC3 team at STScI since the instrument was installed on HST in 2009 during Servicing Mission 4[2]. The genesis of the application was a single script that performed a download request to MAST for new data, created Quicklook JPEGs from the resulting images, and saved them to a centrally-located directory. Over time, the project has evolved into the filesystem-database-web stack that more closely resembles that of the acsql application.

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

With wfc3ql being continuously developed over the past eight years by a team of several developers, many features have been implemented for the application that are not currently implemented on acsql. 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 acsql, as the acsql filesystem is built using the MAST cache of only publicly-available data (and thus the acsql application can be considered open-source and non-proprietary). Secondly, wfc3ql currently contains several instrument calibration and monitoring routines built upon the wfc3ql automation platform and code library (known as pyql), while acsql has yet to implement any such routines. A third example is that the wfc3ql web application supports the tracking of WFC3 instrument image 'anomalies' (i.e. anomalous features in images that are tracked by instrument team members[13]). These features, amongst others, clearly serve as possible extensions to the acsql system; such extensions are discussed further in Chapter 5.

3 Methodology and Implementation

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

3.1 Version control

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

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

Additionally, GitHub allowed for issue tracking for bugs, questions, and potential enhancements to the code

repository. Current open issues of the repository can be found at https://github.com/spacetelescope/acsql/issues.

3.2 Programming and Documentation Standards

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

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

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

Another benefit of using `PEP257` and `numpydoc` docstring conventions is that API documentation creation tools such as `sphinx`[20] or `epydoc`[21] can automatically convert the docstrings into other output formats such as `HTML` and `PDF`. For this project, we use `sphinx` to convert API documentation to `HTML` and host the webpages online using `readthedocs`, an open-source, community supported tool

```
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 `acsql.ingest.ingest`.*

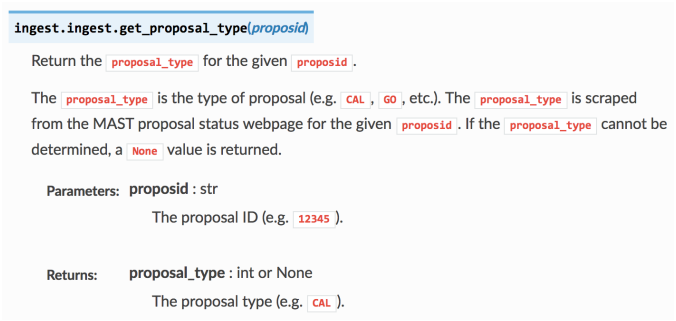


Fig. 6: *The `readthedocs` documentation for the `acsql` example function seen in Figure 5.*

for hosting and browsing documentation[22]. The output documentation as seen on `readthedocs` for the example function in figure 5 is provided in Figure 6. The documentation for `acsql` is available at http://acsql.readthedocs.io/.

3.3 Filesystem: Archive of ACS data

The `acsql` 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 7 shows an example of this directory structure. The parent directory is the first four characters of the 9-character `rootname`, which has a one-to-one correspondence with an individual `PROPOSID`. The subdirectories of the parent directories are named after the full 9-character `rootname` such that each parent directory contains a `rootname` subdirectory for each `rootname` that were observed for the particular `PROPOSID`. Furthermore, each `rootname` subdirectory contains every available filetype (as described in Section 1.1.4) for the particular observation.

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

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

Figure 8 shows how the total size of the filesystem has evolved over the lifetime of the ACS mission; currently, the

filesystem occupies ∼40 TB of storage space. Note that the file sizes across the detectors and across the various filetypes may vary depending on the nature of the particular observation (for example, full-frame observations result in larger file sizes than subarray observations, calibrated filetypes have larger file sizes than un-calibrated filetypes, etc.).

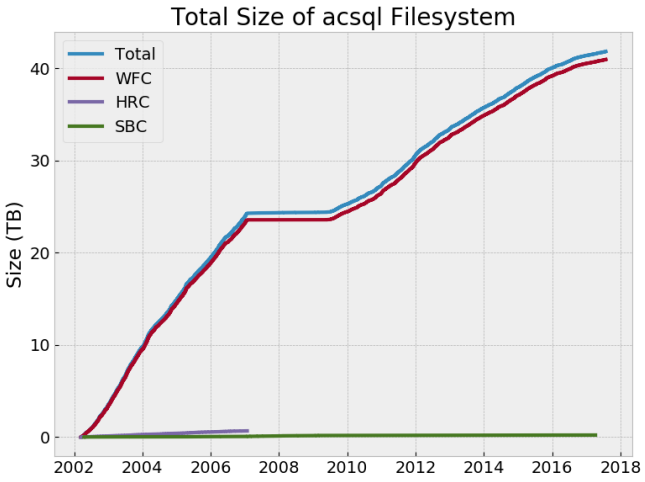


Fig. 8: *The size of the* `acsql` *filesystem as a function of observation date.*

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

3.4 Filesystem: Archive of JPEGs and Thumbnails

In addition to the ACS data products described in section 3.3, 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 utilized by the `acsql` web application to allow users to quickly and easily view ACS data without having to physically open the corresponding FITS files.

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

Unlike the ACS data products portion of the filesystem (section 3.3, the JPEG and thumbnail portions of the filesystem are organized based on the four-or-five digit `PROPOSID` of the corresponding observation instead of the first four

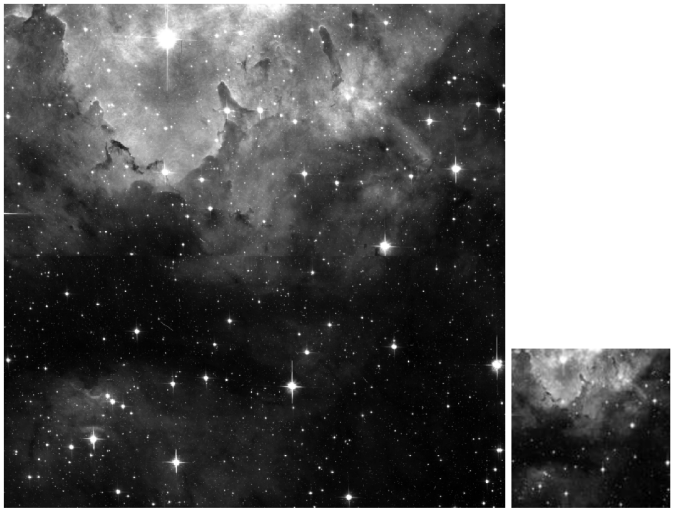


Fig. 9: *An example of a JPEG image (left) and its corresponding thumbnail image using example dataset* `jcs718koq`.

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

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

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

characters of the `rootname`. This design was chosen as a means to simplify the design of the web application; users often wish to view data based on the 4/5-digit `PROPOSID` and less often on the details of the `rootname`. An example of this structure is shown in Figure 10. Note that the

thumbnail/ filesystem only contains thumbnails created from FLT filetypes, since the thumbnails are only intended for navigation and quick-viewing.

3.5 Database: Relational Schema

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

To accomplish this, we implemented the relational schema shown in Figure 11. The acsql database contains 111 tables in total: one master table, which contains basic information about each rootname that is important for maintaining the database, one datasets table which indicate which filetypes are available for a particular rootname, and 109 'header' tables which stores the header key/value pairs, one for each detector/filetype/extension combination (e.g. wfc_raw_0). Each of these tables are described in detail below.

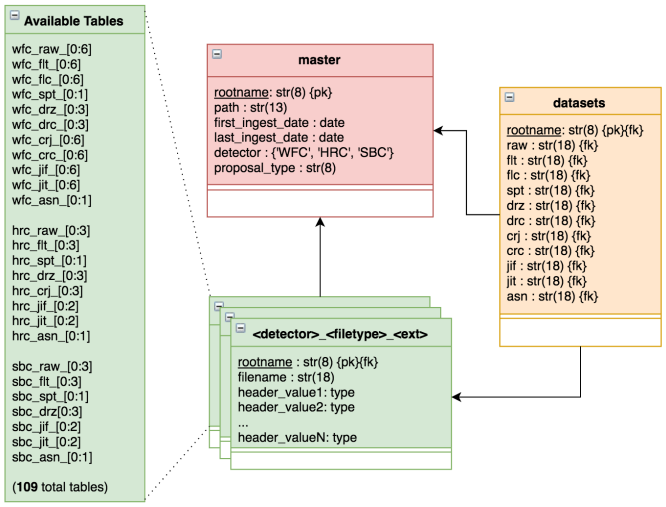


Fig. 11: *The relational database schema for the acsql database.*

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

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

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

3.6 Database: MySQL + SQLAlchemy

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

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

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

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

```
def load_connection(connection_string):
    """Return ``session``, ``base``, and ``engine`` objects for
    connecting to the ``acsql`` 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 ``acsql`` database. The
        connection string should take the form:
        ``dialect+driver://username:password@host:port/database``

    Returns
    -------
    session : sesson 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, used to build a connection to the* `acsql` *database.*

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 commiting inserts or updates (i.e. `session.commit()`).

```
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`*.*

The `master` and `datasets` tables were implemented via explicit class definitions in `database_interface`, and are shown in Figures 13 and 14, respectively. Each table column is defined using the `sqlalchemy.Column` object, which is a class that can be initialized with the column datatype (e.g. `String`, `Float`, `Integer`, etc.) as

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

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

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

well as parameters that set `SQL`-like constraints on the column values. These include, but are not limited to, primary keys (e.g. the `primary_key=True` parameter in the `master.rootname` column), foreign key constrains (e.g. the `ForeignKey` constraint in the `datasets.rootname` column), uniqueness constraints (e.g. the `unique=True` parameters in the `master.path` column), and `NULL` constraints (e.g. the `nullable=False` parameter in the `master.first_ingest_date` column). `SQLAlchemy` determines the name of the table via the `__tablename__` attribute, and determines the name of the columns by the name of the variable used to initialize each `Column` object.

```
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.*

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

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

Furthermore, the 109 text files used to define the header table columns are also generated in an automated fashion via the `acsql.database.make_tabledefs.py` module. This module uses a small set of example ACS FITS files to scrape their header contents, determine all of the header keywords and their datatypes, and write the results to a text file. Similarly, the `acsql.database.update_tabledefs.py` module adds any new header keywords to the text files by comparing the header contents of a given FITS file to the existing column definition text files[4].

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

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

```python
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.*

3.7 Data ingestion software: Algorithm

Another critical component of the `acsql` application is the modules that are used to ingest data into the `acsql` database and to create the 'Quicklook' JPEGs and thumbnails. The ingestion algorithm is encapsulated within a collection of modules in the `acsql` package, namely the `acsql.ingest.ingest` and `acsql.scripts.ingest_production.py` modules (further discussed in section 3.10). Collectively, we refer to this collection of modules as the "ingestion software". We describe the overall algorithm of the ingestion software below.

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

1. ***Identify newly available public ACS data in the filesystem:*** This is accomplished by comparing the list of `rootnames` in the filesystem with the list of `rootnames` in the `master` table of the database. Any `rootnames` that exist in the filesystem but not in the database are considered new `rootnames` to be ingested.

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

```
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.*

is being ingested (i.e. inside of the loop).

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

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

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

```
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.*

```
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 Version: 1.2.1
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`).*

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

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

8. ***If the `filetype` is either `raw`, `flt`, or `flc`, then create a Quicklook JPEG image:*** JPEGs are produced only for `raw`, `flt`, and `flc` filetypes, since these are the filetypes that contain two-dimensional image data. The image data are read into multidimensional `numpy` array data types via the `astropy.io.fits` module. The data are then rescaled as to avoid an undesirable image stretch caused by extremely high or low-valued pixels, and saved to a `.jpg` format. The JPEGs are saved to the `jpegs/` portion of the `acsql` filesystem (described in section 3.4).

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

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

3.8 Data ingestion software: `logging`

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

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

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

With the logging settings configured, any call to the `logging` module within the ingestion software results in a log statement. For example, the command `logging.info('Gathering files to ingest')` results in a timestamped log message, e.g. `08/15/2017 11:05:26 INFO: Gathering files to ingest` (as shown in line 10 of Figure 20.)

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

3.9 Web Application

3.9.1 Overview

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

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

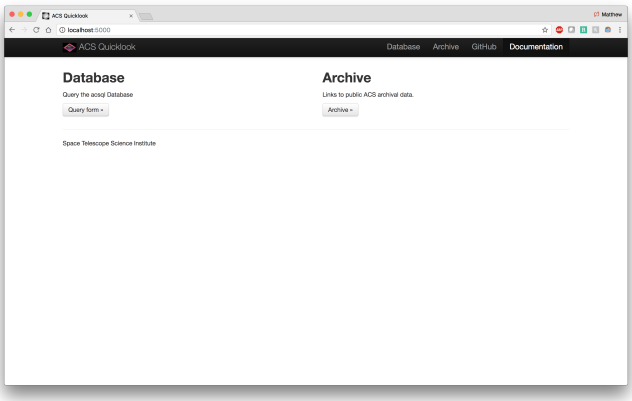


Fig. 21: *The homepage of the `acsql` web application.*

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

In this example, we use the form to find all data taken with the `WFC` detector, are of Observation Type `IMAGING`, and were observed by the Principle Investigator Zolt Levay[5]. Note that there are three output options: (1) an `HTML table`, which returns a webpage showing the

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.

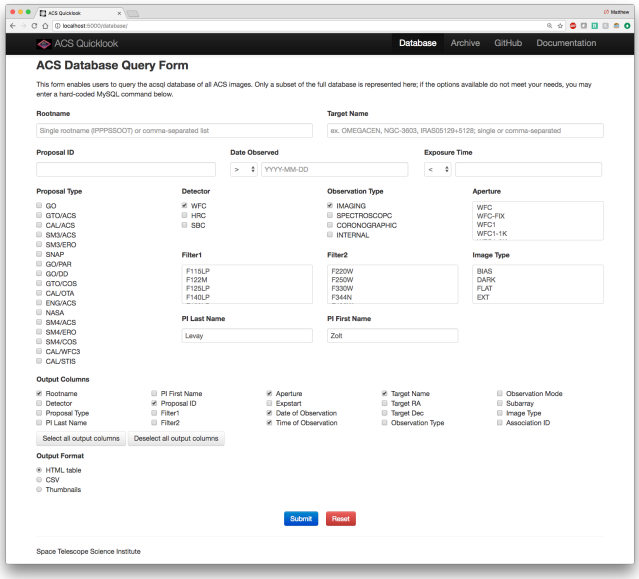


Fig. 22: *A portion of the database query page of the `acsql` web application.*

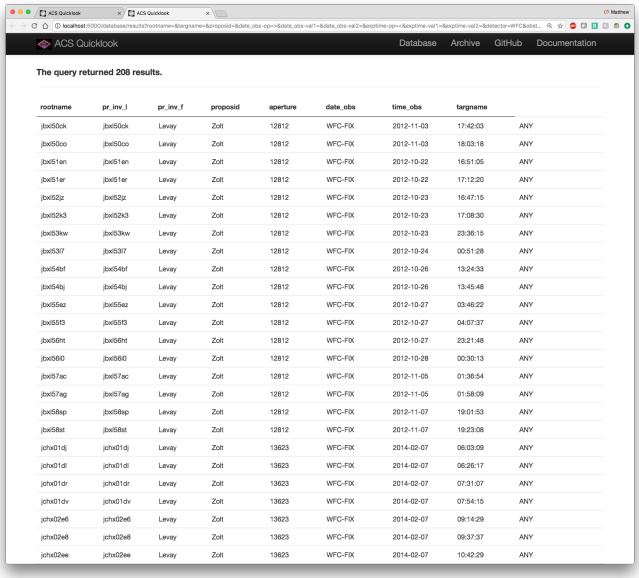
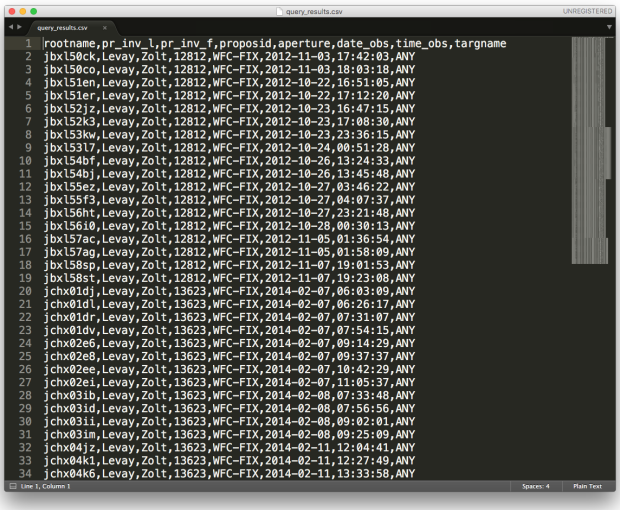


Fig. 23: *The results of the database query shown in Figure 22 in the form of an HTML table.*

selected 'Output Columns' (in this case, the Rootname, PI last/first name, Proposal ID, Aperture, Date/Time of observation, and the Target Name; see Figure 23), (2) a CSV file downloaded to the user's machine containing a comma-separated table of the selected output columns (see Figure 24), or (3) a webpage showing the thumbnail images of all of the resulting data (see Figure 25).

Figure 26 shows the archive links page. This page contains links to every proposal that contains publicly available ACS data via the 4-5 digit `PROPOSID`. Clicking one of the links opens a new window to a 'view proposal' webpage. Similar to the database query results shown in Figure 25, the 'view proposal' webpage shows thumbnail images of



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

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

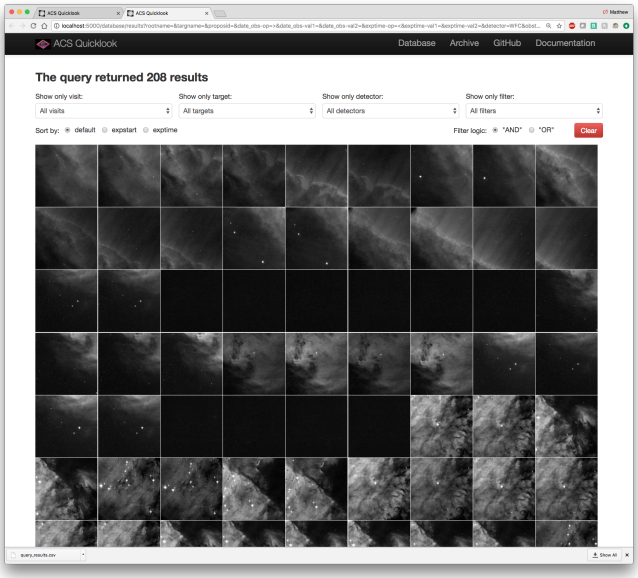


Fig. 25: *The results of the database query shown in Figure 22 in the form of thumbnail images.*

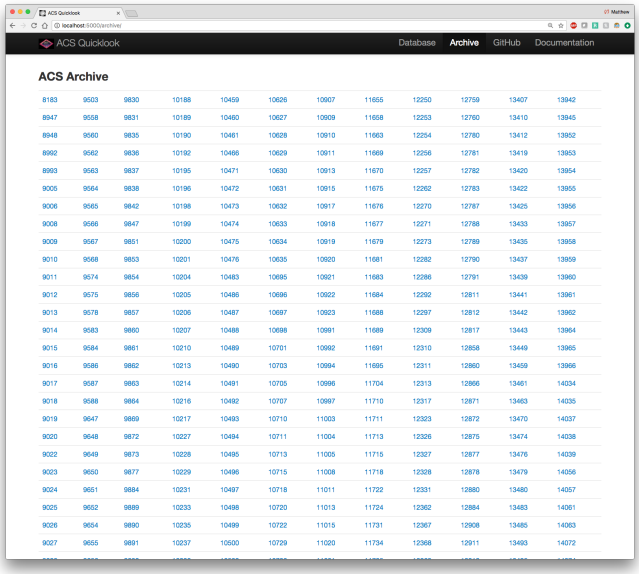


Fig. 26: *The archive links webpage of the* `acsql` *web application.*

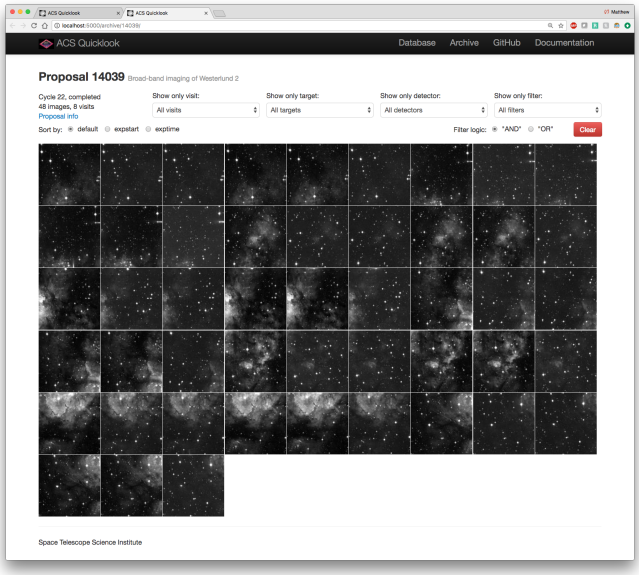


Fig. 27: *An example 'view proposal' webpage, using proposal 14039.*

each ACS dataset available in the given proposal. The 'view proposal' pages contain several buttons and filters near the top of the page; using these will sort and/or filter out the thumbnail images based on the chosen parameters. Figure 27 shows an example of this webpage, using proposal 14039.

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

3.9.2 *Implementation*

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

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

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

- `static/css/*.css`

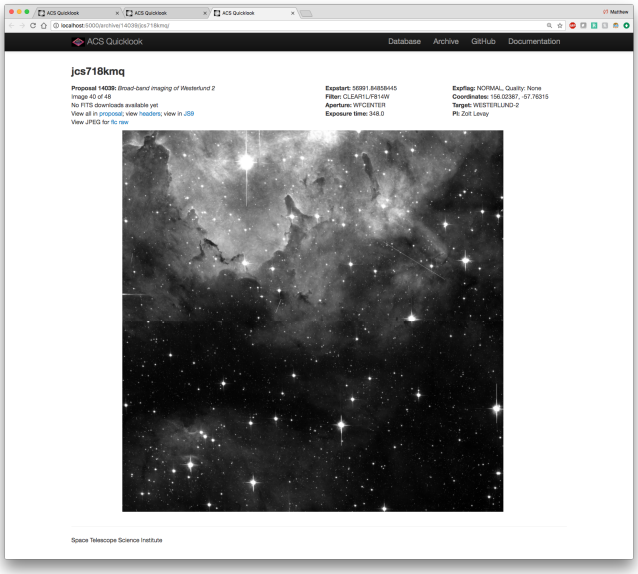


Fig. 28: *An example 'view image' webpage, using dataset* `jcs718kmq`.

- `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; this module must be executed in order to serve the web application (either on a dedicated web server, or locally on a user's machine and accessed through the user's `localhost`). This module uses the `Flask` web framework to receive incoming `HTML` requests. It is structured such that each `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 call functions from the `data_containers` module to gather the data that is eventually passed on to the `HTML` template. This design choice was made in order to separate the data from the functionality, often regarded as a best practice in software engineering.

Similarly, the `form_options.py` module is a container module for storing the possible form options (e.g. apertures, filters, output column, proposal types, etc.) for the database query form page. These data could have been defined within the `acsql_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 `acsql` database, respectively. The `query_form` module contains several class objects for supplying specific types of form fields (e.g. check box fields, text fields, multiple select fields) as well as functions to validate form entries. Several of the form objects are extensions of components (via subclassing) provided by the `wtforms` library, which is a form validation and rendering library for Python web development (wtforms)[27]. The query_lib module contains functions that parse, build, validate, and return `SQLAlchemy query` objects from the results of a submitted database query form. These queries are then executed on the `acsql` database and results are used to render the database query result HTML, CSV, and/or thumbnail webpages.

Like many web applications, we also store static files such as `HTML` templates, `CSS` templates, and static images in separate `static/` and `templates/` directories. The `static/css/` directory contains the `CSS` templates used in the `HTML` templates. Similarly, the `static/js/` directory contains several JavaScript libraries, also used in the `HTML` templates. Lastly, the `templates/` directory contains the `HTML` templates used to render the various webpages of the web application. There is one template for each type of webpage, as well as a template that houses `HTML` that is common to each webpage (e.g. the menu bar), named `base.html`. Aside from some minor tweaks, we use templates from `Bootstrap`, which is a front-end component library and open source toolkit for developing `HTML`, `CSS`, and `JS`[28].

3.10 `acsql` Package

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

```
acsql/
    LICENSE
    README.md
    MANIFEST.in
    setup.py
    paper/
        ...
    presentation/
        ...
    docs/
        Makefile
        requirements.rst
        source/
          conf.py
          database.rst
          index.rst
          ingest.rst
          scripts.rst
          utils.rst
          website.rst
    acsql/
        __init__.py
        database/
            __init__.py
            database_interface.py
            make_tabledefs.py
            queries.py
            reset_database.py
            table_definitions/
                *.txt
            update_tabledefs.py
        ingest/
```

```
        __init__.py
        ingest.py
        make_file_dict.py
        make_jpeg.py
        make_thumbnail.py
    scripts/
        __init__.py
        ingest_production.py
    utils/
        __init__.py
        config.yaml
        utils.py
    website/
        __init__.py
        acsql_webapp.py
        data_containers.py
        form_options.py
        query_form.py
        query_lib.py
        static/
            css/
                *.css
            img/
                jpegs
                thumbnails
            js/
                *.js
        templates/
            *.html
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`scripts/`: The `scripts` subpackage, containing Python modules for ingesting multiple files from the `acsql`

filesystem. This also serves as storage place for possible future instrument calibration and monitoring routines (see section 3.7).

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

`utils/`: The `utils` subpackage, containing Python modules that are useful for general `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 `acsql` web application (see section 3.9).

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

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

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

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

`static/`: A subdirectory containing static materials used by the `acsql` 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 `acsql` web application, one for each page (see section 3.9).

4 Results

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

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

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

4.1 Filesystem Deliverable

As described in sections 3.3 and 3.4, the `acsql` filesystem is comprised of two parts: (1) A filesystem that stores the archive of publicly-available ACS data (i.e. FITS files), and (2) a filesystem of Quicklook JPEGs and thumbnails.

For the former, we utilized an already-existing filesystem of ACS data internal to STScI known as the "MAST public cache"; this filesystem is organized in a similar way to that shown in Figure 7[6]. 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 publically available to download via the MAST archive (i.e. https://archive.stsci.edu/). The location of the filesystem is determined by the `filesystem` parameter in the user-supplied `config.yaml` file (see section 3.10).

Currently, the filesystem consists of ∼1,030,000 total files. Figure 29 shows how this breaks down by individual filetype. We see that `spt`, `raw`, and `flt` make up the majority of the files, which is not surprising considering that these filetypes exist for nearly every ACS observing mode. On the other hand, we see a small amount of `crj` and `crc` files, which is also unsurprising considering that these filetypes are only triggered for specific observing modes.

As for item (2), the filesystem of JPEGs and thumbnails was constructed during the ingestion of all publicly-available ACS data (via `ingest_production.py`, see section 3.9). The location of the JPEG/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 filesystem[7].

4.2 Database Deliverable

As described in sections 3.5 and 3.6, the `acsql` database is a `MySQL` database that stores the header information of every public ACS dataset. Currently, this database is

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

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

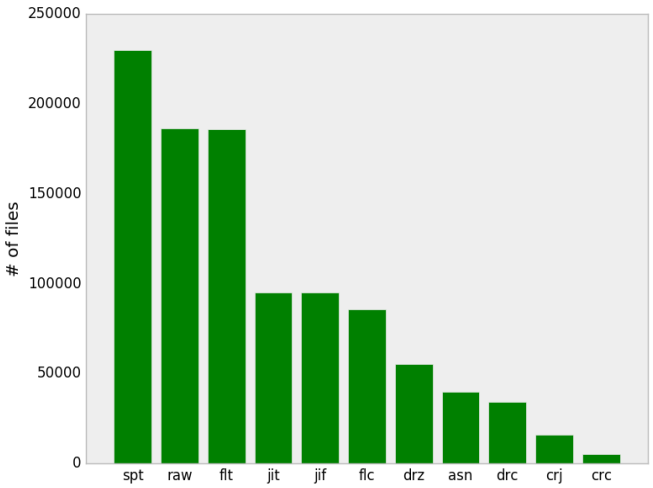


Fig. 29: *The number of files in the* `acsql` *filesystem by filetype.*

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

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

4.3 Web Application Deliverable

As described in section 3.9, the `acsql` web application consists of a series of Python modules and static files. Currently, the web application is not hosted on a dedicated web server (neither externally nor internally to STScI[8]), however, users can operate the application through the user's `localhost`. Detailed installation and operation instructions for the web application are provided in the official documentation (see section 3.2). In short, users must (1) ensure that they have built (or have access to) the `acsql` filesystem and database, (2) have installed the Python package dependencies, and (3) have filled out the `config.yaml` file (see section 3.10).

4.4 Software Package Deliverable

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

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

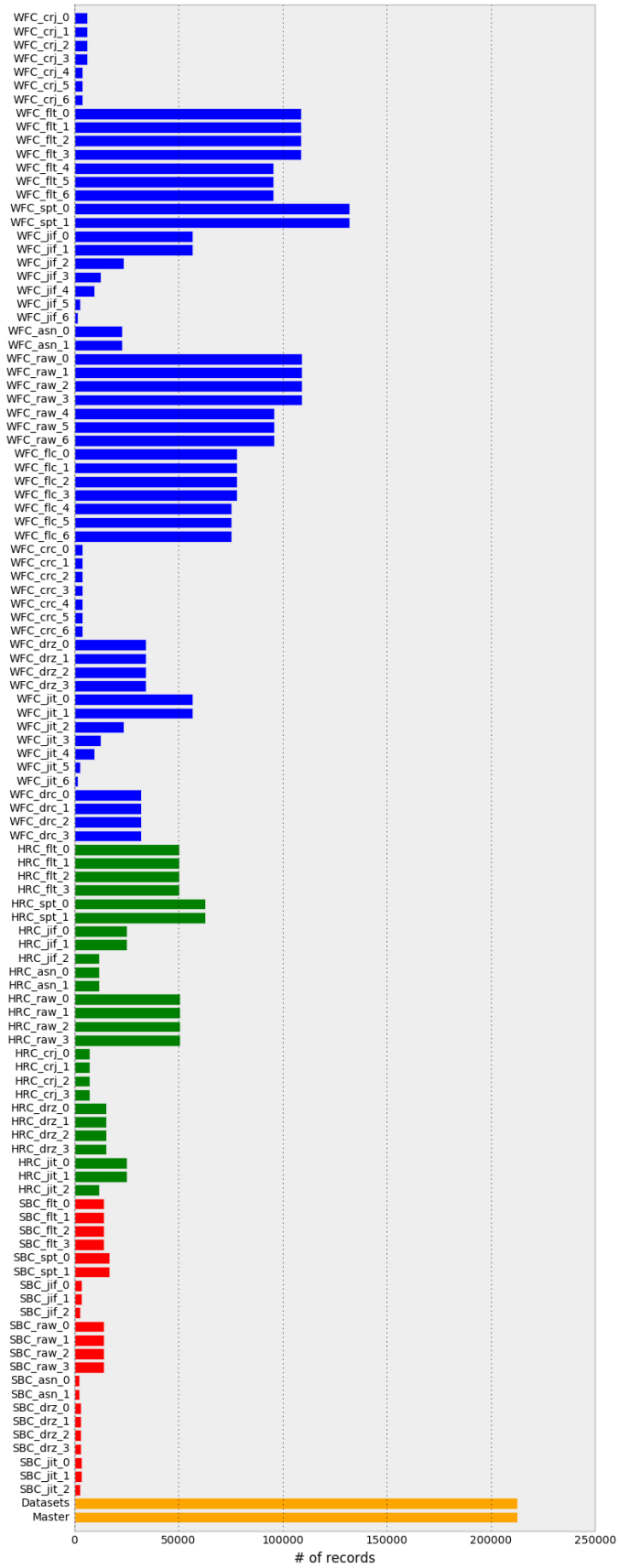


Fig. 30: *The number of records in each of the* `acsql` *database tables.*

4.5 Software Documentation Deliverable

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

5 Discussion

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

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

5.1 Possible enhancements to the `acsql` application

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

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

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

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

4. ***Expanding the database query form:*** The database query form webpage on the `acsql` web application allows user to construct and execute a query to the `acsql` database using some of the most commonly used observational parameters (e.g. target name, filter, principle investigator name, etc.). However, these parameters are just a small subset of the total collection of parameters available in the `acsql` database. With the database containing all header keywords, it is possible to extend this query form use some more obscure (but useful) observational parameters. Examples of these may include telescope telemetry parameters (e.g. instrument temperature, telescope pointing, telescope latitude/longitude, etc.), engineering parameters (e.g. detector gain, detector readnoise, etc.), and photometric parameters (e.g. detector sensitivity, photometric zeropoints, etc.). We expect that the database query webpage in particular will endure the most feature development as the usefulness of certain parameters become evident from the user base.

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

5.2 Possible extensions to other instruments or missions

This type of database-driven, web-based application for interacting with instrument data is currently only implemented for the WFC3 (`wfc3ql`) and ACS (`acsql` instruments of HST. However, the continued success of `wfc3ql` and potential success of `acsql` creates a strong case for implementing such a system for other instruments and missions. As this project has been fully developed at STScI,

which is home of the operations for the Hubble Space Telescope and the forthcoming James Webb Space Telescope (JWST), it is reasonable to consider extending this application to other imaging instruments on HST (e.g. The Space Telescope Imaging Spectrography; STIS) and the future imaging instruments on JWST.

6 Conclusion

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

Acknowledgments

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

References

[1] *SM3B*, National Aeronautics and Space Administration, Goddard Space Flight Center, [Online; accessed 2017-09-16], available at https://asd.gsfc.nasa.gov/archive/hubble/missions/sm3b.html.

[2] *SM4*, National Aeronautics and Space Administration, Goddard Space Flight Center, [Online; accessed 2017-09-16], available at https://asd.gsfc.nasa.gov/archive/hubble/missions/sm4.html.

[3] Avila, R., et al., 2017, *ACS Instrument Handbook*, Version 16.0 (Baltimore: STScI)

[4] *The Barbara A. Mikulski Archive for Space Telescopes*, [Online; accessed 2017-07-30], available at https://archive.stsci.edu/.

[5] *Definition of the Flexible Image Transport System (FITS): The FITS Standard*, 2008, International Astronomical Union FITS Working Group, available at https://fits.gsfc.nasa.gov/standard30/fits_standard30aa.pdf.

[6] *STScI: Space Telescope Science Institute*, [Online; accessed 2017-09-16], available at www.stsci.edu.

[7] Smith, E., et al., 2011, *Introduction to the HST Data Handbooks*, Version 8.0 (Baltimore: STScI)

[8] Lucas, R. A., et al., 2016, *ACS Data Handbook*, Version 8.0 (Baltimore: STScI)

[9] Robitaille, T.P., et al., 2013, *Astropy: A community Python package for astronomy*, Astronomy & Astrophysics, 558, A33.

[10] *The MAST Portal*, [Online; accessed 2017-09-15], available at https://mast.stsci.edu/.

[11] Bourque, M., et al., 2016, *The Hubble Space Telescope Wide Field Camera 3 Quicklook Project*, Astronomical Data Analysis Software & Systems Conference Proceedings, ADASS XXVI, ASP-CS, submitted

[12] Bourque, M., et al., 2016, *The HST/WFC3 Quicklook Project: A User Interface to Hubble Space Telescope Wide Field Camera 3 Data*, Proceedings IAU Symposium No. 325, 2016, Astroinformatics.

[13] Gosmeyer, C.M., and The Quicklook Team, 2017, *WFC3 Anomalies Flagged by the Quicklook Team*, WFC3 Instrument Science Report, 2017-22, available at http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2017-22.pdf

[14] *git*, [Online; accessed 2017-08-05], available at https://git-scm.com.

[15] *GitHub*, [Online; accessed 2017-08-05], available at https:/github.com.

[16] van Rossum, G., 2001, *PEP 8 – Style Guide for Python Code*, Python Developer's Guide, available at https://www.python.org/dev/peps/pep-0008/.

[17] Goodger, D., 2001, *PEP 257 – Docstring Conventions*, Python Developer's Guide, available at https://www.python.org/dev/peps/pep-0257/.

[18] *A Guide to NumPy/SciPy Documentation*, [Online; accessed 2017-08-05], available at https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt/

[19] van der Walt, S., Colbert, C., and Varoquaux, G., 2011, *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering, 13, 22-30.

[20] Brandi, G., et al., 2007, *Sphinx: Python Documentation Generator*, available at http://www.sphinx-doc.org/en/stable/.

[21] Loper, E., 2004, *Epydoc: Generating API Documentation in Python*, Proceedings of the Second Annual Python Conference, available at http://epydoc.sourceforge.net/.

[22] *Read the Docs*, [Online; accessed 2017-08-05], available at https://readthedocs.org.

[23] *MySQL 5.6 Reference Manual*, Oracle, [Online; accessed 2017-08-13], available at https://dev.mysql.com/doc/refman/5.6/en/.

[24] Bayer, M., 2006, *SQLAlchemy: The database toolkit for Python*, [Online; accessed 2017-02-21], available at http://www.sqlalchemy.org/.

[25] Lemburg, M., 2017, *PEP 249 – Python Database API Specification v2.0*, Python Developer's Guide, available at https://www.python.org/dev/peps/pep-0249/.

[26] Ronacher, A., et al., 2010, [Online; accessed 2017-09-04], available at http://flask.pocoo.org/.

[27] *wtforms*, [Online; accessed 2017-09-04], available at https://github.com/wtforms/wtforms/.

[28] *Bootstrap*, [Online; accessed 2017-09-04], available at http://getbootstrap.com/.

[29] Bourque, M. and Baggett, S., 2013, *WFC3/UVIS Bowtie Monitor*, WFC3 Instrument Science Report, 2013-09, available at http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2013-09.pdf

[30] Bourque, M. and Baggett, S., 2016, *WFC3/UVIS Dark Calibration: Monitoring Results and Improvements to Dark Reference FIles*, WFC3 Instrument Science Report, 2016-08, available at http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2016-08.pdf

[31] Bushouse, H., 2005, *CSM Particulate Investigation*, WFC3 Instrument Science Report, 2005-26, available at http://www.stsci.edu/hst/wfc3/documents/ISRs/WFC3-2005-26.pdf

APPENDIX A
ACSQL CODE

Table of Contents:

LICENSE

```
Copyright (c) 2017, AURA
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name of acsql nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

README.md

````
# acsql
The Advanced Camera for Surveys Quicklook Project

Official Documentation: http://acsql.readthedocs.io/

## Installation

#### Dependencies

The `acsql` package requires the following dependencies:

- `python 3.+`
- `astropy`
- `flask`
- `numpy`
- `numpydoc`
- `pymysql`
- `requests`
- `sqlalchemy`
- `wtforms`
- `wtforms_components`
- `stak` (http://github.com/spacetelescope/stak)

#### Installing the `acsql` Package

The `acsql` application can be installed by cloning this repository and running the `setup.py` script:

```
git clone https://github.com/spacetelescope/acsql.git
cd acsql
python setup.py install
```

Users must fill in the `config.yaml` configuration file, located at `acsql/utils/config.yaml`:

```yaml
connection_string : 'mysql+pymysql://username:password@host:12345/database'
filesystem : ''
log_dir : ''
jpeg_dir : ''
thumbnail_dir : ''
ncores : 1
```

The `connection_string` item should contain the users credentials to the `acsql` database.  Please ask [
    @bourque](http://github.com/bourque) to set up an account.

````

````
The 'filesystem' item should point to the directory that holds ACS FITS files in a directory structure of
'<proposal_id>/<rootname>/<rootname_<filetype>.fits', for example:

```
filesystem : '/Users/bourque/filesystem/'
```

where '/Users/bourque/filesystem/' contains subdirectories such as:

```
/users/bourque/filesystem/jbm1/
/users/bourque/filesystem/jbm1/jbm110u2q/
/users/bourque/filesystem/jbm1/jbm110u0q/
/users/bourque/filesystem/jbm1/jbm110tyq/
/users/bourque/filesystem/jbm1/jbm110txq/
/users/bourque/filesystem/jbm1/jbm110010/
/users/bourque/filesystem/jbm1/jbm105y7q/
/users/bourque/filesystem/jbm1/jbm105y5q/
/users/bourque/filesystem/jbm1/jbm105y3q/
/users/bourque/filesystem/jbm1/jbm105y2q/
/users/bourque/filesystem/jbm1/jbm105010/
```

and each one of these subdirectories contains the data files, for example:

```
>>> ls /users/bourque/filesystem/jbm1/jbm110u2q/

jbm110u2q_raw.jpg
jbm110u2q_flt.jpg
jbm110u2q_trl.fits
jbm110u2q_flt_hlet.fits
jbm110u2q_spt.fits
jbm110u2q_flt.fits
jbm110u2q_flc.fits
jbm110u2q_raw.fits
```

In production, this 'filesystem' item points to the directory of the MAST Online Cache.

The 'log_dir' item should point to a directory in which log files that describe code execution will be
    written.

The 'jpeg_dir' item should point to a directory in which JPEGs will be written.

The 'thumbnail_dir' item should point to a directory in which smaller Thumbnail images will be written.

The 'ncores' item is set to the number of processors that should be used when performing data ingestion.

#### Running the 'acsql' web application locally:

Once the 'acsql' package is installed, the 'acsql' web application can be run locally:

```
python acsql/website/acsql_webapp.py
```

The just go to 'localhost:5000' in a browser.
````

MANIFEST.in

```
include LICENSE
include acsql/utils/config.yaml
include acsql/database/table_definitions/hrc_asn_0.txt
include acsql/database/table_definitions/hrc_asn_1.txt
include acsql/database/table_definitions/hrc_crj_0.txt
include acsql/database/table_definitions/hrc_crj_1.txt
include acsql/database/table_definitions/hrc_crj_2.txt
include acsql/database/table_definitions/hrc_crj_3.txt
include acsql/database/table_definitions/hrc_drz_0.txt
include acsql/database/table_definitions/hrc_drz_1.txt
include acsql/database/table_definitions/hrc_drz_2.txt
include acsql/database/table_definitions/hrc_drz_3.txt
include acsql/database/table_definitions/hrc_flt_0.txt
include acsql/database/table_definitions/hrc_flt_1.txt
include acsql/database/table_definitions/hrc_flt_2.txt
include acsql/database/table_definitions/hrc_flt_3.txt
include acsql/database/table_definitions/hrc_jif_0.txt
```

```
include acsql/database/table_definitions/hrc_jif_1.txt
include acsql/database/table_definitions/hrc_jif_2.txt
include acsql/database/table_definitions/hrc_jit_0.txt
include acsql/database/table_definitions/hrc_jit_1.txt
include acsql/database/table_definitions/hrc_jit_2.txt
include acsql/database/table_definitions/hrc_raw_0.txt
include acsql/database/table_definitions/hrc_raw_1.txt
include acsql/database/table_definitions/hrc_raw_2.txt
include acsql/database/table_definitions/hrc_raw_3.txt
include acsql/database/table_definitions/hrc_spt_0.txt
include acsql/database/table_definitions/hrc_spt_1.txt
include acsql/database/table_definitions/sbc_asn_0.txt
include acsql/database/table_definitions/sbc_asn_1.txt
include acsql/database/table_definitions/sbc_drz_0.txt
include acsql/database/table_definitions/sbc_drz_1.txt
include acsql/database/table_definitions/sbc_drz_2.txt
include acsql/database/table_definitions/sbc_drz_3.txt
include acsql/database/table_definitions/sbc_flt_0.txt
include acsql/database/table_definitions/sbc_flt_1.txt
include acsql/database/table_definitions/sbc_flt_2.txt
include acsql/database/table_definitions/sbc_flt_3.txt
include acsql/database/table_definitions/sbc_jif_0.txt
include acsql/database/table_definitions/sbc_jif_1.txt
include acsql/database/table_definitions/sbc_jif_2.txt
include acsql/database/table_definitions/sbc_jit_0.txt
include acsql/database/table_definitions/sbc_jit_1.txt
include acsql/database/table_definitions/sbc_jit_2.txt
include acsql/database/table_definitions/sbc_raw_0.txt
include acsql/database/table_definitions/sbc_raw_1.txt
include acsql/database/table_definitions/sbc_raw_2.txt
include acsql/database/table_definitions/sbc_raw_3.txt
include acsql/database/table_definitions/sbc_spt_0.txt
include acsql/database/table_definitions/sbc_spt_1.txt
include acsql/database/table_definitions/wfc_asn_0.txt
include acsql/database/table_definitions/wfc_asn_1.txt
include acsql/database/table_definitions/wfc_crc_0.txt
include acsql/database/table_definitions/wfc_crc_1.txt
include acsql/database/table_definitions/wfc_crc_2.txt
include acsql/database/table_definitions/wfc_crc_3.txt
include acsql/database/table_definitions/wfc_crc_4.txt
include acsql/database/table_definitions/wfc_crc_5.txt
include acsql/database/table_definitions/wfc_crc_6.txt
include acsql/database/table_definitions/wfc_crj_0.txt
include acsql/database/table_definitions/wfc_crj_1.txt
include acsql/database/table_definitions/wfc_crj_2.txt
include acsql/database/table_definitions/wfc_crj_3.txt
include acsql/database/table_definitions/wfc_crj_4.txt
include acsql/database/table_definitions/wfc_crj_5.txt
include acsql/database/table_definitions/wfc_crj_6.txt
include acsql/database/table_definitions/wfc_drc_0.txt
include acsql/database/table_definitions/wfc_drc_1.txt
include acsql/database/table_definitions/wfc_drc_2.txt
include acsql/database/table_definitions/wfc_drc_3.txt
include acsql/database/table_definitions/wfc_drz_0.txt
include acsql/database/table_definitions/wfc_drz_1.txt
include acsql/database/table_definitions/wfc_drz_2.txt
include acsql/database/table_definitions/wfc_drz_3.txt
include acsql/database/table_definitions/wfc_flc_0.txt
include acsql/database/table_definitions/wfc_flc_1.txt
include acsql/database/table_definitions/wfc_flc_2.txt
include acsql/database/table_definitions/wfc_flc_3.txt
include acsql/database/table_definitions/wfc_flc_4.txt
include acsql/database/table_definitions/wfc_flc_5.txt
include acsql/database/table_definitions/wfc_flc_6.txt
include acsql/database/table_definitions/wfc_flt_0.txt
include acsql/database/table_definitions/wfc_flt_1.txt
include acsql/database/table_definitions/wfc_flt_2.txt
include acsql/database/table_definitions/wfc_flt_3.txt
include acsql/database/table_definitions/wfc_flt_4.txt
include acsql/database/table_definitions/wfc_flt_5.txt
include acsql/database/table_definitions/wfc_flt_6.txt
include acsql/database/table_definitions/wfc_jif_0.txt
include acsql/database/table_definitions/wfc_jif_1.txt
include acsql/database/table_definitions/wfc_jif_2.txt
include acsql/database/table_definitions/wfc_jif_3.txt
include acsql/database/table_definitions/wfc_jif_4.txt
include acsql/database/table_definitions/wfc_jif_5.txt
```

```
include acsql/database/table_definitions/wfc_jif_6.txt
include acsql/database/table_definitions/wfc_jit_0.txt
include acsql/database/table_definitions/wfc_jit_1.txt
include acsql/database/table_definitions/wfc_jit_2.txt
include acsql/database/table_definitions/wfc_jit_3.txt
include acsql/database/table_definitions/wfc_jit_4.txt
include acsql/database/table_definitions/wfc_jit_5.txt
include acsql/database/table_definitions/wfc_jit_6.txt
include acsql/database/table_definitions/wfc_raw_0.txt
include acsql/database/table_definitions/wfc_raw_1.txt
include acsql/database/table_definitions/wfc_raw_2.txt
include acsql/database/table_definitions/wfc_raw_3.txt
include acsql/database/table_definitions/wfc_raw_4.txt
include acsql/database/table_definitions/wfc_raw_5.txt
include acsql/database/table_definitions/wfc_raw_6.txt
include acsql/database/table_definitions/wfc_spt_0.txt
include acsql/database/table_definitions/wfc_spt_1.txt
```

setup.py

```
from setuptools import setup
from setuptools import find_packages

setup(
    name = 'acsql',
    description = 'The Advanced Camera for Surveys Quicklook Project',
    url = 'https://github.com/spacetelescope/acsql.git',
    author = 'Matthew Bourque, Sara Ogaz, Meredith Durbin, Alex Viana',
    author_email = 'bourque@stsci.edu, ogaz@stsci.edu, mdurbin@uw.edu, alexcostaviana@gmail.com',
    keywords = ['astronomy'],
    classifiers = ['Programming Language :: Python'],
    packages = find_packages(),
    install_requires = [],
    version = 0.0,
    include_package_data=True
    )
```

docs.Makefile

```
# Minimal makefile for Sphinx documentation
#

# You can set these variables from the command line.
SPHINXOPTS    =
SPHINXBUILD   = sphinx-build
SPHINXPROJ    = acsql
SOURCEDIR     = source
BUILDDIR      = build

# Put it first so that "make" without argument is like "make help".
help:
  @$(SPHINXBUILD) -M help "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)

.PHONY: help Makefile

# Catch-all target: route all unknown targets to Sphinx using the new
# "make mode" option.  $(O) is meant as a shortcut for $(SPHINXOPTS).
%: Makefile
  @$(SPHINXBUILD) -M $@ "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)
```

docs/requirements.txt

```
astropy
flask
numpy
numpydoc
pymysql
pyyaml
sqlalchemy
wtforms
wtforms_components
-e git://github.com/spacetelescope/stak.git#egg=stak
```

docs/source/conf.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
# acsql documentation build configuration file, created by
# sphinx-quickstart on Wed Feb 22 16:11:29 2017.
#
# This file is execfile()d with the current directory set to its
# containing dir.
#
# Note that not all possible configuration values are present in this
# autogenerated file.
#
# All configuration values have a default; values that are commented out
# serve to show the default.

# If extensions (or modules to document with autodoc) are in another directory,
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown here.
#
import os
import sys
sys.path.insert(0, os.path.abspath('../../acsql'))


# -- General configuration ------------------------------------------------

# If your documentation needs a minimal Sphinx version, state it here.
#
# needs_sphinx = '1.0'

# Add any Sphinx extension module names here, as strings. They can be
# extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
# ones.
extensions = ['sphinx.ext.autodoc', 'numpydoc']

# Add any paths that contain templates here, relative to this directory.
# templates_path = ['_templates']

# The suffix(es) of source filenames.
# You can specify multiple suffix as a list of string:
#
# source_suffix = ['.rst', '.md']
source_suffix = '.rst'

# The master toctree document.
master_doc = 'index'

# General information about the project.
project = 'acsql'
copyright = '2017, Matthew Bourque'
author = 'Matthew Bourque'

# The version info for the project you're documenting, acts as replacement for
# |version| and |release|, also used in various other places throughout the
# built documents.
#
# The short X.Y version.
version = '0.0'
# The full version, including alpha/beta/rc tags.
release = '0.0'

# The language for content autogenerated by Sphinx. Refer to documentation
# for a list of supported languages.
#
# This is also used if you do content translation via gettext catalogs.
# Usually you set "language" from the command line for these cases.
language = None

# List of patterns, relative to source directory, that match files and
# directories to ignore when looking for source files.
# This patterns also effect to html_static_path and html_extra_path
exclude_patterns = []

# The name of the Pygments (syntax highlighting) style to use.
pygments_style = 'sphinx'

```

```
# If true, `todo` and `todoList` produce output, else they produce nothing.
todo_include_todos = False


# -- Options for HTML output ----------------------------------------------

# The theme to use for HTML and HTML Help pages.  See the documentation for
# a list of builtin themes.
#
html_theme = 'default'

# Theme options are theme-specific and customize the look and feel of a theme
# further.  For a list of options available for each theme, see the
# documentation.
#
# html_theme_options = {}

# Add any paths that contain custom static files (such as style sheets) here,
# relative to this directory. They are copied after the builtin static files,
# so a file named "default.css" will overwrite the builtin "default.css".
# html_static_path = ['_static']


# -- Options for HTMLHelp output ------------------------------------------

# Output file base name for HTML help builder.
htmlhelp_basename = 'acsqldoc'


# -- Options for LaTeX output ---------------------------------------------

latex_elements = {
    # The paper size ('letterpaper' or 'a4paper').
    #
    # 'papersize': 'letterpaper',

    # The font size ('10pt', '11pt' or '12pt').
    #
    # 'pointsize': '10pt',

    # Additional stuff for the LaTeX preamble.
    #
    # 'preamble': '',

    # Latex figure (float) alignment
    #
    # 'figure_align': 'htbp',
}

# Grouping the document tree into LaTeX files. List of tuples
# (source start file, target name, title,
#  author, documentclass [howto, manual, or own class]).
latex_documents = [
    (master_doc, 'acsql.tex', 'acsql Documentation',
     'Matthew Bourque', 'manual'),
]


# -- Options for manual page output ---------------------------------------

# One entry per manual page. List of tuples
# (source start file, name, description, authors, manual section).
man_pages = [
    (master_doc, 'acsql', 'acsql Documentation',
     [author], 1)
]


# -- Options for Texinfo output -------------------------------------------

# Grouping the document tree into Texinfo files. List of tuples
# (source start file, target name, title, author,
#  dir menu entry, description, category)
texinfo_documents = [
    (master_doc, 'acsql', 'acsql Documentation',
     author, 'acsql', 'One line description of project.',
     'Miscellaneous'),]
```

docs/source/database.rst

```
========
Database
========

The ``database`` subpackage contains various modules for constructing and interacting with the ``acsql``
    Database.


database_interface
------------------
.. automodule:: database.database_interface
    :members: define_columns, get_special_column, load_connection, orm_factory
    :undoc-members:
    :show-inheritance:

make_tabledefs
--------------
.. automodule:: database.make_tabledefs
    :members:
    :undoc-members:
    :show-inheritance:

queries
-------
.. automodule:: database.queries
    :members:
    :undoc-members:
    :show-inheritance:

reset_database
--------------
.. automodule:: database.reset_database
    :members:
    :undoc-members:
    :show-inheritance:

update_tabledefs
----------------
.. automodule:: database.update_tabledefs
    :members:
    :undoc-members:
    :show-inheritance:
```

docs/source/index.rst

```
.. acsql documentation master file, created by
   sphinx-quickstart on Wed Feb 22 16:11:29 2017.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

The Hubble Space Telescope (HST) Advanced Camera for Surveys (ACS) Quicklook Project (ascql)
==========================================================================================

The ``acsql`` project is a database and web application that serves Hubble Space Telescope (HST)
Advanced Camera for Surveys (ACS) data for easy access and quick-viewing.


.. toctree::
   :maxdepth: 2

   database.rst
   ingest.rst
   utils.rst
   website.rst


Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

docs/source/ingest.rst

```
======
Ingest
======

The ``ingest`` subpackage provides various modules to support the ingestion of files into the ``acsql`` database
and filesystem, such as creation of JPEGs & thumbnails and inserting or updating records in the database.

ingest
------
.. automodule:: ingest.ingest
    :members:
    :undoc-members:
    :show-inheritance:

make_file_dict
--------------
.. automodule:: ingest.make_file_dict
    :members:
    :undoc-members:
    :show-inheritance:

make_jpeg
---------
.. automodule:: ingest.make_jpeg
    :members:
    :undoc-members:
    :show-inheritance:

make_thumbnail
--------------
.. automodule:: ingest.make_thumbnail
    :members:
    :undoc-members:
    :show-inheritance:
```

docs/source/scripts.rst

```
=======
Scripts
=======

ingest_production
-----------------
.. automodule:: scripts.ingest_production.py
    :members:
    :undoc-members:
    :show-inheritance:
```

docs/source/utils.rst

```
=====
Utils
=====

The ``utils`` subpackage provides various various utility functions and objects that aid the other
subpackages, such as configuration settings, logging functions, and database convenience functions.

utils
-----
.. automodule:: utils.utils
    :members:
    :undoc-members:
    :show-inheritance:
```

docs/source/website.rst

```
=======
Website
=======

The ``website`` subpackage provides various modules to support the ``acsql`` web application.

acsql_webapp
------------
.. automodule:: website.acsql_webapp
    :members:
```

```
    :undoc-members:
    :show-inheritance:

data_containers
---------------
.. automodule:: website.data_containers
    :members:
    :undoc-members:
    :show-inheritance:

form_options
------------
.. automodule:: website.form_options
    :members:
    :undoc-members:
    :show-inheritance:

query_form
----------
.. automodule:: website.query_form
    :members:
    :undoc-members:
    :show-inheritance:

query_lib
---------
.. automodule:: website.query_lib
    :members:
    :undoc-members:
    :show-inheritance:
```

acsql/database/database_interface.py

```
"""This module provides ORMs for the ``acsql`` database, as well as
``engine`` and ``session`` objects for connecting to the database.

The ``load_connection()`` function within this module allows the user
to connect to the ``acsql`` database via the ``session``, ``base``,
and ``engine`` objects (described below).  The classes within serve as
ORMs (Object-relational mappings) that define the individual tables of
the relational database.

The ``engine`` object serves as the low-level database API and perhaps
most importantly contains dialects which allows the ``sqlalchemy``
module to communicate with the database.

The ``base`` object serves as a base class for class definitions.  It
produces ``Table`` objects and constructs ORMs.

The ``session`` object manages operations on ORM-mapped objects, as
construced by the base.  These operations include querying, for
example.

Authors
-------
    Matthew Bourque

Use
---
    This module is intended to be imported from various ``acsql``
    modules and scripts.  The importable objects from this module are
    as follows:
    ::

        from acsql.database.database_interface import base
        from acsql.database.database_interface import engine
        from acsql.database.database_interface import session
        from acsql.database.database_interface import Master
        from acsql.database.database_interface import Datasets
        from acsql.database.database_interface import <header_table>

Dependencies
------------
    External library dependencies include:

    - ``acsql``
    - ``pymysql``
    - ``sqlalchemy``
```

```
"""

import os

from sqlalchemy import Boolean
from sqlalchemy import Column
from sqlalchemy import create_engine
from sqlalchemy import Date
from sqlalchemy import DateTime
from sqlalchemy import Index
from sqlalchemy import Enum
from sqlalchemy import ForeignKey
from sqlalchemy import ForeignKeyConstraint
from sqlalchemy import Integer
from sqlalchemy.orm import sessionmaker
from sqlalchemy import String
from sqlalchemy import Time
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.types import Float

from acsql.utils.utils import SETTINGS


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


def get_special_column(keyword):
```

```
    """Treat specific keywords separately.

    Parameters
    ----------
    keyword : str
        The header keyword.

    Returns
    -------
    Column : obj
        A SQLAlchemy Column object for the given ``keyword``.
    """

    if keyword in ['RULEFILE', 'PROPTTL1', 'TARDESCR', 'QUALCOM2']:
        return Column(String(500))
    elif keyword in ['FWERROR', 'FW2ERROR']:
        return Column(String(100))
    else:
        return Column(String(50))


def load_connection(connection_string):
    """Return ``session``, ``base``, and ``engine`` objects for
    connecting to the ``acsql`` 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 ``acsql`` database. The
        connection string should take the form:
        ``dialect+driver://username:password@host:port/database``

    Returns
    -------
    session : sesson 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


session, base, engine = load_connection(SETTINGS['connection_string'])


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)


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)


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

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

    # foreign_keys = []
    # for filetype in FILE_EXTS:
    #     for ext in [0, 1]:
    #         foreign_keys.append(ForeignKeyConstraint([filetype],
    #             ['wfc_{}_{}.filename'.format(filetype, ext)]))
    # foreign_keys = tuple(foreign_keys)
    # __table_args__ = foreign_keys


# WFC tables
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')

WFC_spt_0 = orm_factory('WFC_spt_0')
WFC_spt_1 = orm_factory('WFC_spt_1')

WFC_drz_0 = orm_factory('WFC_drz_0')
WFC_drz_1 = orm_factory('WFC_drz_1')
WFC_drz_2 = orm_factory('WFC_drz_2')
WFC_drz_3 = orm_factory('WFC_drz_3')

WFC_drc_0 = orm_factory('WFC_drc_0')
WFC_drc_1 = orm_factory('WFC_drc_1')
WFC_drc_2 = orm_factory('WFC_drc_2')
WFC_drc_3 = orm_factory('WFC_drc_3')

WFC_crj_0 = orm_factory('WFC_crj_0')
WFC_crj_1 = orm_factory('WFC_crj_1')
WFC_crj_2 = orm_factory('WFC_crj_2')
WFC_crj_3 = orm_factory('WFC_crj_3')
WFC_crj_4 = orm_factory('WFC_crj_4')
WFC_crj_5 = orm_factory('WFC_crj_5')
WFC_crj_6 = orm_factory('WFC_crj_6')

WFC_crc_0 = orm_factory('WFC_crc_0')
WFC_crc_1 = orm_factory('WFC_crc_1')
WFC_crc_2 = orm_factory('WFC_crc_2')
WFC_crc_3 = orm_factory('WFC_crc_3')
WFC_crc_4 = orm_factory('WFC_crc_4')
WFC_crc_5 = orm_factory('WFC_crc_5')
WFC_crc_6 = orm_factory('WFC_crc_6')

WFC_jif_0 = orm_factory('WFC_jif_0')
WFC_jif_1 = orm_factory('WFC_jif_1')
WFC_jif_2 = orm_factory('WFC_jif_2')
WFC_jif_3 = orm_factory('WFC_jif_3')
WFC_jif_4 = orm_factory('WFC_jif_4')
WFC_jif_5 = orm_factory('WFC_jif_5')
WFC_jif_6 = orm_factory('WFC_jif_6')

WFC_jit_0 = orm_factory('WFC_jit_0')
WFC_jit_1 = orm_factory('WFC_jit_1')
WFC_jit_2 = orm_factory('WFC_jit_2')
WFC_jit_3 = orm_factory('WFC_jit_3')
WFC_jit_4 = orm_factory('WFC_jit_4')
WFC_jit_5 = orm_factory('WFC_jit_5')
WFC_jit_6 = orm_factory('WFC_jit_6')

WFC_asn_0 = orm_factory('WFC_asn_0')
WFC_asn_1 = orm_factory('WFC_asn_1')


# HRC tables
HRC_raw_0 = orm_factory('HRC_raw_0')
HRC_raw_1 = orm_factory('HRC_raw_1')
HRC_raw_2 = orm_factory('HRC_raw_2')
HRC_raw_3 = orm_factory('HRC_raw_3')

HRC_flt_0 = orm_factory('HRC_flt_0')
HRC_flt_1 = orm_factory('HRC_flt_1')
HRC_flt_2 = orm_factory('HRC_flt_2')
HRC_flt_3 = orm_factory('HRC_flt_3')

HRC_spt_0 = orm_factory('HRC_spt_0')
HRC_spt_1 = orm_factory('HRC_spt_1')

HRC_drz_0 = orm_factory('HRC_drz_0')
HRC_drz_1 = orm_factory('HRC_drz_1')
HRC_drz_2 = orm_factory('HRC_drz_2')
HRC_drz_3 = orm_factory('HRC_drz_3')

HRC_crj_0 = orm_factory('HRC_crj_0')
HRC_crj_1 = orm_factory('HRC_crj_1')
HRC_crj_2 = orm_factory('HRC_crj_2')
HRC_crj_3 = orm_factory('HRC_crj_3')

HRC_jif_0 = orm_factory('HRC_jif_0')
```

```
HRC_jif_1 = orm_factory('HRC_jif_1')
HRC_jif_2 = orm_factory('HRC_jif_2')

HRC_jit_0 = orm_factory('HRC_jit_0')
HRC_jit_1 = orm_factory('HRC_jit_1')
HRC_jit_2 = orm_factory('HRC_jit_2')

HRC_asn_0 = orm_factory('HRC_asn_0')
HRC_asn_1 = orm_factory('HRC_asn_1')


# SBC tables
SBC_raw_0 = orm_factory('SBC_raw_0')
SBC_raw_1 = orm_factory('SBC_raw_1')
SBC_raw_2 = orm_factory('SBC_raw_2')
SBC_raw_3 = orm_factory('SBC_raw_3')

SBC_flt_0 = orm_factory('SBC_flt_0')
SBC_flt_1 = orm_factory('SBC_flt_1')
SBC_flt_2 = orm_factory('SBC_flt_2')
SBC_flt_3 = orm_factory('SBC_flt_3')

SBC_spt_0 = orm_factory('SBC_spt_0')
SBC_spt_1 = orm_factory('SBC_spt_1')

SBC_drz_0 = orm_factory('SBC_drz_0')
SBC_drz_1 = orm_factory('SBC_drz_1')
SBC_drz_2 = orm_factory('SBC_drz_2')
SBC_drz_3 = orm_factory('SBC_drz_3')

SBC_jif_0 = orm_factory('SBC_jif_0')
SBC_jif_1 = orm_factory('SBC_jif_1')
SBC_jif_2 = orm_factory('SBC_jif_2')

SBC_jit_0 = orm_factory('SBC_jit_0')
SBC_jit_1 = orm_factory('SBC_jit_1')
SBC_jit_2 = orm_factory('SBC_jit_2')

SBC_asn_0 = orm_factory('SBC_asn_0')
SBC_asn_1 = orm_factory('SBC_asn_1')


if __name__ == '__main__':

    # Give user a second chance
    prompt = ('About to reset all table(s) for database instance {}. Do you '
        'wish to proceed? (y/n)\n'.format(SETTINGS['connection_string']))

    response = input(prompt)

    if response.lower() == 'y':
        print('Resetting table(s)')
        base.metadata.drop_all()
        base.metadata.create_all()
```

acsql/database/make_tabledefs.py

```
#! /usr/bin/env/ python

"""Creates static text files that hold header keywords and keyword
data types for each ACS filetype (and each extension).  Each text file
corresponds to a header table in the ``acsql`` database.

Authors
-------
    - Sara Ogaz
    - Matthew Bourque

Use
---
    This module is intended to be run via the command line as such:
    ::

        python make_tabledefs.py

Dependencies
------------
    External library dependencies include:
```

```
22
23      - ``acsql``
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 handeled 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

```
#! /usr/bin/env python

"""Contains various functions to perform useful queries of the
``acsql`` database.

The available queries are:

    1. ``all_filenames``
    2. ``filters_for_rootname(rootname)``
    3. ``filter_distribution()``
    4. ``rootnames_for_target(targname)``
    5. ``filenames_for_calibration(calibration_keyword, value)``
    6. ``goodmean_for_dataset(dataset)``
    7. ``rootnames_with_postflash()``
    8. ``non_asn_rootnames()``
    9. ``filenames_in_date_rage()``

See each function's docstrings for further details.

Each function returns the ``sqlalchemy.query`` object for the query
performed so that the user may perform the query themselves and
perform additional operations with the query and/or its results.

Authors
-------
    - Sara Ogaz
    - Matthew Bourque

Use
---
    This script is intended to be imported as such:
    ::

        from acsql.database import queries

    ``queries`` can then be used to perform individual queries, e.g.:
    ::

        query = queries.filter_distribution()

    Each function will print the results to the screen, but the user
    may also perform the query and handle the results themselves, e.g.:
    ::

        results = query.all()

Dependencies
------------
    External library dependencies include:

    - ``acsql``
    - ``sqlalchemy``
"""

from sqlalchemy import and_
from sqlalchemy import exists
from sqlalchemy import func

from acsql.database.database_interface import session
from acsql.database.database_interface import Master
from acsql.database.database_interface import Datasets
from acsql.database.database_interface import WFC_asn_0
from acsql.database.database_interface import WFC_raw_0
from acsql.database.database_interface import WFC_flt_1
from acsql.database.database_interface import WFC_flt_4


def all_filenames(dataset):
    """Queries for all filenames that exist for the given dataset.

    Parameters
    ----------
    dataset : str
        Any portion of (or entire) rootname (e.g. 'jd2615qi', or
        'jd2615').

```

```
    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(Datasets)\
        .filter(Datasets.rootname.like('{}%'.format(dataset)))
    query_results = query.all()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        results_dict = result.__dict__
        del results_dict['_sa_instance_state']
        print(results_dict)

    return query


def filters_for_rootname(rootname):
    """Queries for the FILTER1/FILTER2 combination for the geven
    observation.

    Parameters
    ----------
    rootname : str
        The rootname to query by.

    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(WFC_raw_0.filter1, WFC_raw_0.filter2)\
        .filter(WFC_raw_0.rootname == rootname)
    query_results = query.one()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))
    print('{}: {}'.format(rootname, query_results))

    return query


def filter_distribution():
    """Queries for the FILTER1/FILTER2 combination for the given
    observation.

    Parameters
    ----------
    rootname : str
        The rootname to query by.

    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(WFC_raw_0.filter1, WFC_raw_0.filter2,
        func.count(WFC_raw_0.filter1))\
            .group_by(WFC_raw_0.filter1, WFC_raw_0.filter2)
    query_results = query.all()
    db_count = session.query(WFC_raw_0).count()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        perc_used = round((result[2] / db_count) * 100., 2)
        print('\t{}/{}: {}%'.format(result[0], result[1], perc_used))

    return query

```

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

```
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(Master.rootname, WFC_flt_1.goodmean,
        WFC_flt_4.goodmean)\
            .join(WFC_flt_1)\
            .join(WFC_flt_4)\
            .filter(Master.rootname.like('{}%'.format(dataset)))
    query_results = query.all()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        print(result)

    return query


def rootnames_with_postflash():
    """Queries for rootnames and FLASHDURs for non-DARK observations
    that have a FLASHDUR > 0.

    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(Master.rootname, WFC_raw_0.flashdur)\
        .join(WFC_raw_0)\
        .filter(WFC_raw_0.flashdur > 0)\
        .filter(WFC_raw_0.targname != 'DARK')

    query_results = query.all()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        print(result)

    return query


def non_asn_rootnames():
    """Queries for rootnames that are not part of an association.

    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(Master.rootname)\
        .filter(~exists().where(and_(Master.rootname == WFC_asn_0.rootname)))
    query_results = query.all()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        print(result[0])

    return query


def filenames_in_date_range(begin_date, end_date):
    """Queries for filenames for observations that occur between the
    ``begin_date`` and ``end_date``.

    Parameters
    ----------
    begin_date : str
        The start of the date range (in the format YYYY-MM-DD).
    end_date : str
```

```
        The end of the date range (in the format YYYY-MM-DD).

    Returns
    -------
    query : obj
        The query object that contains attributes and methods for
        performing the query.
    """

    query = session.query(WFC_raw_0.filename)\
        .filter(WFC_raw_0.date_obs >= begin_date)\
        .filter(WFC_raw_0.date_obs <= end_date)
    query_results = query.all()

    print('\nQuery performed:\n\n{}\n'.format(str(query)))

    for result in query_results:
        print(result[0])

    return query
```

acsql/database/reset_database.py

```
#! /usr/bin/env python

"""Reset all tables in the ``acsql`` database.

Authors
-------
    Matthew Bourque, 2017

Use
---
    This script is intended to be used in the command line:
    ::

        python reset_database.py

Dependencies
------------
    External library dependencies include:

    - ``acsql``
"""

from acsql.database.database_interface import base
from acsql.utils.utils import SETTINGS


if __name__ == '__main__':

    prompt = ('About to reset all tables for database instance {}. Do you '
              'wish to proceed? (y/n)\n'.format(SETTINGS['connection_string']))
    response = input(prompt)

    if response.lower() == 'y':
        print('Resetting database.')
        base.metadata.drop_all()
        base.metadata.create_all()
```

acsql/database/update_tabledefs.py

```
#! /usr/bin/env python

"""Updates the ``table_definitions`` text files that store header
keyword/data type pairs (see ``make_tabledefs.py`` module documentation
for further details).

A given ``logfile`` is used to scrape for logging ``WARNINGS`` that
warn of missing ``acsql`` database columns (i.e. header keywords that
exist in the file header but does not exist as a database column in the
appropriate header table).  The new header keywords are appended to the
appropriate ``table_definitions`` text file based on the ``detector``,
``filetype`` and ``extension`` (e.g. ``WFC_raw_0.txt``).

Additionally, the ``ALTER TABLE`` commands needed to add the
corresponding columns are printed to the standard output.  ``acsql``
users can then use these commands within ``MySQL`` to add the
appropriate columns.
```

```
18
19  Authors
20  -------
21      Matthew Bourque
22
23  Use
24  ---
25      This module is intended to be used when an ``acsql`` user sees fit
26      to add any new header keywords to the database.  The module can
27      be used from the command line as such:
28      ::
29
30          python update_tabledefs.py <logfile>
31
32      Required arguments:
33      ::
34
35          logfile: The path to an ``acsql.ingest.ingest.py`` log to be
36              used to determine new header keywords.
37
38  Dependencies
39  ------------
40      External library dependencies include:
41
42      - ``acsql``
43      - ``numpy``
44      - ``stak`` (``https://github.com/spacetelescope/stak``)
45
46  Notes
47  -----
48      The ``stak.Hselect`` dependency still depends on Python 2 at the
49      time of this writing.
50  """
51
52  import argparse
53  import os
54
55  import numpy as np
56  #from stak import Hselect
57
58  from acsql.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
```

```
    ----------
    logfile : str
        The path to an ``acsql.ingest.ingest.py`` log to be used to
        determine new header keywords.
    """

    # Read in the logfile
    with open(logfile) as f:
        data = f.readlines()

    # Grab out the appropriate WARNING lines
    warnings = [item.strip() for item in data if 'not in' in item]
    warnings = [item.split(':') for item in warnings]

    # Determine metadata for each warning
    rootnames = [item[-2].strip() for item in warnings]
    keywords = [item[-1].strip().split(' not in')[0] for item in warnings]
    tables = [item[-1].strip().split('not in ')[-1] for item in warnings]
    filetypes = [item.split('_')[1] for item in tables]
    paths = [os.path.join(SETTINGS['filesystem'], rootname[0:4], rootname,
             '{}_{}.fits'.format(rootname, filetype)) for rootname, filetype
             in zip(rootnames, filetypes)]

    # Take a set of the warnings to avoid duplications
    files_to_test = ['{},{},{}'.format(path, keyword, table) for path,
                     keyword, table in zip(paths, keywords, tables)]
    keywords_to_add = set([(keyword, table) for keyword, table in zip(keywords,
                                                                      tables)])

    # For each keyword to add, use test file to determine data type
    command_list = []
    for keyword, table in keywords_to_add:
        test_file = [f for f in files_to_test if '{},{}'
                     .format(keyword, table) in f][0]
        test_file = test_file.split(',')[0]

        # Use hselect to determine data type
        try:
            hsel = Hselect(test_file, keyword, extension=(int(table[-1]),))
            dtype = hsel.table[keyword].dtype
        except:
            print('Cannot determine datatype for {}. Defaulting to String'
                  .format(keyword))
            dtype = np.dtype('S80')

        if dtype in [np.dtype('S68'), np.dtype('S80')]:
            col_type = 'String'
            db_type = 'VARCHAR(50)'
        elif dtype in [np.int64]:
            col_type = 'Integer'
            db_type = 'INTEGER'
        elif dtype in [bool]:
            col_type = 'Bool'
            db_type = 'BOOLEAN'
        elif dtype in [np.float64]:
            col_type = 'Float'
            db_type = 'FLOAT'
        else:
            print('Could not find type match: {}:{}'.format(keyword, dtype))

        # Update the appropriate table definitions file
        tabledefs_file = 'table_definitions/{}.txt'.format(table.lower())
        with open(tabledefs_file, 'r') as f:
            data = f.readlines()
        existing_keywords = [item.strip().split(',')[0] for item in data]
        with open(tabledefs_file, 'a') as f:
            if keyword not in existing_keywords:
                f.write('{}, {}\n'.format(keyword, col_type))
        print('Updated {} with {}'.format(tabledefs_file, keyword))

        # Add ALTER TABLE command to list of commands
        command = 'ALTER TABLE {} ADD {} {};'.format(table.lower(),
                                                     keyword.lower(), db_type)
        command_list.append(command)

    # Print out the ALTER TABLE commands
    print('\n\nALTER TABLE commands to execute for database:\n')
```

```
    for command in command_list:
        print(command)

if __name__ == '__main__':

    args = parse_args()
    update_tabledefs(args.logfile)
```

acsql/ingest/ingest.py

```
"""Ingests a given rootname (and its associated files) into the
``ascql`` database.  The tables that are updated are the ``master``
table, the ``datasets`` table, and any appropriate header tables
(e.g. ``wfc_raw_0``) based on the available filetypes and header
extensions.

Authors
-------
    - Matthew Bourque
    - Sara Ogaz

Use
---
    This module is intended to be imported from and used by the
    ``ingest_production`` script as such:
    ::

        from acsql.ingest.ingest import ingest
        ingest(rootname)

Dependencies
------------
    External library dependencies include:

    - ``acsql``
    - ``astropy``
    - ``sqlalchemy``
"""

from datetime import date
import glob
import logging
import os
import urllib.request

from astropy.io import fits
from astropy.io.fits.verify import VerifyError
from sqlalchemy import Table
from sqlalchemy.exc import IntegrityError

from acsql.database.database_interface import Datasets
from acsql.database.database_interface import load_connection
from acsql.ingest.make_file_dict import get_metadata_from_test_files
from acsql.ingest.make_file_dict import make_file_dict
from acsql.ingest.make_jpeg import make_jpeg
from acsql.ingest.make_thumbnail import make_thumbnail
from acsql.utils.utils import insert_or_update
from acsql.utils.utils import SETTINGS
from acsql.utils.utils import TABLE_DEFS
from acsql.utils.utils import VALID_FILETYPES
from acsql.utils.utils import VALID_PROPOSAL_TYPES


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``).
    """

    if not proposid:
        proposal_type = None
    else:
        try:
            url = 'http://www.stsci.edu/cgi-bin/get-proposal-info?id='
            url += '{}&submit=Go&observatory=HST'.format(proposid)
            webpage = urllib.request.urlopen(url)
            proposal_type = webpage.readlines()[11].split(b'prop_type">')[-1]
            proposal_type = proposal_type.split(b'</a>')[0].decode()
        except:
            logging.warning('Cannot determine proposal type for {}'\
                .format(proposid))
            proposal_type = None

    # Check for bad proposal types
    if proposal_type not in VALID_PROPOSAL_TYPES:
        logging.warning('Cannot determine proposal type for {}'\
            .format(proposid))
        proposal_type = None

    return proposal_type


def update_datasets_table(file_dict):
    """Insert/update an entry for the file in the ``datasets`` table.

    Parameters
    ----------
    file_dict : dict
        A dictionary containing various data useful for the ingestion
        process.
    """

    session, base, engine = load_connection(SETTINGS['connection_string'])

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

    # If there are no results, then perform an insert
    if not query_count:

        data_dict = {}
        data_dict['rootname'] = file_dict['rootname']
        data_dict[file_dict['filetype']] = file_dict['basename']

        tab = Table('datasets', base.metadata, autoload=True)
        insert_obj = tab.insert()
        try:
            insert_obj.execute(data_dict)
        except IntegrityError as e:
            logging.warning('{}: Unable to insert {} into datasets table: {}'\
                .format(file_dict['full_rootname'], file_dict['basename'], e))

    # If there are results, add the filename to the existing entry
    else:
        data_dict = query.one().__dict__
        del data_dict['_sa_instance_state']
        data_dict[file_dict['filetype']] = file_dict['basename']
        try:
            query.update(data_dict)
        except IntegrityError as e:
            logging.warning('{}: Unable to update {} in datasets table: {}'\
                .format(file_dict['full_rootname'], file_dict['basename'], e))

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

    logging.info('{}: Updated datasets table for {}.'\
        .format(file_dict['rootname'], file_dict['filetype']))

```

```
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 {}'\
200                         .format(file_dict['full_rootname'], key, table))
201                     continue
202
203                 input_dict[key.lower()] = value
204
205             insert_or_update(table, input_dict)
206             logging.info('{}: Updated {} table.'.format(file_dict['rootname'],
207                                                         table))
208
209         except VerifyError as e:
210             logging.warning('\tUnable to insert {} into {}: {}'.format(
211                 file_dict['rootname'], table, e))
212
213
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'),
236                  'proposal_type': proposal_type}
237     insert_or_update('Master', data_dict)
238     logging.info('{}: Updated master table.'.format(rootname))
239
240
241 def ingest(rootname_path, filetype='all'):
242     """The main function of the ingest module.  Ingest a given rootname
243     (and its associated files) into the various tables of the ``acsql``
244     database.
245
246     If for some reason the file is unable to be ingested, a warning is
247     logged.
248
249     Parameters
250     ----------
251     rootname_path : str
252         The path to the rootname directory in the MAST cache.
253     """
254
255     rootname = os.path.basename(rootname_path)[:-1]
256     logging.info('{}: Begin ingestion'.format(rootname))
257
258     # Update the master table for the rootname
259     update_master_table(rootname_path)
260
261     if filetype == 'all':
262         search = '*.fits'
263     else:
264         search = '*{}.fits'.format(filetype)
265     file_paths = glob.glob(os.path.join(rootname_path, search))
266
267     for filename in file_paths:
268         filetype = os.path.basename(filename).split('.')[0][10:]
269         if filetype in VALID_FILETYPES:
270
271             # Make dictionary that holds all the information you would ever
272             # want about the file
273             file_dict = make_file_dict(filename)
274
275             # Update header tables
276             if 'file_exts' in file_dict:
277                 for ext in file_dict['file_exts']:
278                     update_header_table(file_dict, ext)
279
280                 # Update datasets table
281                 update_datasets_table(file_dict)
282
283                 # Make JPEGs and Thumbnails
284                 if file_dict['filetype'] in ['raw', 'flt', 'flc']:
285                     make_jpeg(file_dict)
286                 if file_dict['filetype'] == 'flt':
287                     make_thumbnail(file_dict)
288
289     logging.info('{}: End ingestion'.format(rootname))
```

acsql/ingest/make_file_dict.py

```
"""Create a dictionary containing useful information for the ingestion
process.

The ``file_dict`` contains various information that can be used by
``ingest.py`` (e.g. filesystem paths, observational metadata) and can
be used as a data container that can be easily passed around to various
functions.

Authors
-------
    Matthew Bourque

Use
---
    This module and its functionars are intended to be imported and
    used by ``acsql.ingest.ingest.py`` as such:
    ::

        from ascql.ingest.make_file_dict import get_detector
        from ascql.ingest.make_file_dict import get_metadata_from_test_files
        from ascql.ingest.make_file_dict import get_proposid
        from acsql.ingest.make_file_dict import make_file_dict

        make_file_dict(filename)
        get_detector(filename)
        get_metadata_from_test_files(rootname_path, keyword)
        get_proposid(filename)

Dependencies
------------
    External library dependencies include:

    - ``astropy``
"""

import glob
import logging
import os

from astropy.io import fits

from acsql.utils import utils
from acsql.utils.utils import SETTINGS


def get_detector(filename):
    """Return the ``detector`` associated with the given ``filename``,
    if possible.

    Parameters
    ----------
    filename : str
        The path to the file to attempt to get the ``detector`` header
        keyword from.

    Returns
    -------
    detector : str
        The detector (e.g. ``WFC``)
    """

    if 'jit' in filename:
        detector = fits.getval(filename, 'config', 0)
        if detector == 'S/C':  # FGS observation
            detector = None
        else:
            detector = detector.lower().split('/')[1]
    else:
        detector = fits.getval(filename, 'detector', 0).lower()

    return detector


def get_metadata_from_test_files(rootname_path, keyword):
    """Return the value of the given ``keyword`` and ``rootname_path``.

```

```
    The given ``rootname_path`` is checked for various filetypes that
    are beleived to have the ``keyword`` that is sought, in order
    of most likeliness: ``raw``, ``flt``, ``spt``, ``drz``, and
    ``jit``.  If a candidate file is found, it is used to determine
    the value of the ``keyword`` in the primary header.  If no
    candidate file exists, or the ``keyword`` value cannot be
    determined from the primary header, a ``value`` of ``None`` is
    returned, essentially ending the ingestion process for the given
    rootname.

    Parameters
    ----------
    rootname_path : str
        The path to the rootname in the MAST cache.
    keyword : str
        The header keyword to determine the value of (e.g.
        ``detector``)

    Returns
    -------
    value : str or None
        The header keyword value.
    """

    raw = glob.glob(os.path.join(rootname_path, '*raw.fits'))
    flt = glob.glob(os.path.join(rootname_path, '*flt.fits'))
    spt = glob.glob(os.path.join(rootname_path, '*spt.fits'))
    drz = glob.glob(os.path.join(rootname_path, '*drz.fits'))
    jit = glob.glob(os.path.join(rootname_path, '*jit.fits'))

    for test_files in [raw, flt, spt, drz, jit]:
        try:
            test_file = test_files[0]
            if keyword == 'detector':
                value = get_detector(test_file)
            elif keyword == 'proposid':
                value = get_proposid(test_file)
            break
        except (IndexError, KeyError):
            value = None

    if not value:
        logging.warning('Cannot determine {} for {}'\
            .format(keyword, rootname_path))

    return value


def get_proposid(filename):
    """Return the proposal ID from the primary header of the given
    ``filename``.

    Parameters
    ----------
    filename : str
        The path to the file to get the ``proposid`` form.

    Returns
    -------
    proposid : int
        The proposal ID (e.g. ``12345``).
    """

    proposid = str(fits.getval(filename, 'proposid', 0))

    return proposid


def make_file_dict(filename):
    """Create a dictionary that holds information that is useful for
    the ingestion process.  This dictionary can then be passed around
    the various functions of the module.

    Parameters
    ----------
    filename : str
        The path to the file.
```

```
    Returns
    -------
    file_dict : dict
        A dictionary containing various data useful for the ingestion
        process.
    """

    file_dict = {}

    # Filename related keywords
    file_dict['filename'] = os.path.abspath(filename)
    file_dict['dirname'] = os.path.dirname(filename)
    file_dict['basename'] = os.path.basename(filename)
    file_dict['rootname'] = file_dict['basename'].split('_')[0][:-1]
    file_dict['full_rootname'] = file_dict['basename'].split('_')[0]
    file_dict['filetype'] = file_dict['basename'].split('.fits')[0].split('_')[-1]
    file_dict['proposid'] = file_dict['basename'][0:4]
    file_dict['proposid_int'] = get_metadata_from_test_files(file_dict['dirname'], 'proposid')

    # Metadata kewords
    file_dict['detector'] = get_metadata_from_test_files(file_dict['dirname'], 'detector')
    if file_dict['detector']:
        file_dict['file_exts'] = getattr(utils, '{}_FILE_EXTS'.format(file_dict['detector'].upper()))[
    file_dict['filetype']]

    # JPEG related kewords
    if file_dict['filetype'] in ['raw', 'flt', 'flc']:
        file_dict['jpg_filename'] = file_dict['basename'].replace('.fits', '.jpg')
        file_dict['jpg_dst'] = os.path.join(SETTINGS['jpeg_dir'], file_dict['proposid_int'], file_dict['
    jpg_filename'])
        file_dict['thumbnail_filename'] = file_dict['basename'].replace('.fits', '.thumb')
        file_dict['thumbnail_dst'] = os.path.join(SETTINGS['thumbnail_dir'], file_dict['proposid_int'],
    file_dict['thumbnail_filename'])
    else:
        file_dict['jpg_filename'] = None
        file_dict['jpg_dst'] = None
        file_dict['thumbnail_filename'] = None
        file_dict['thumbnail_dst'] = None

    return file_dict
```

acsql/ingest/make_jpeg.py

```
"""Create a "Quicklook" JPEG for the given observation.

A JPEG is created for every ``raw``, ``flt``, and ``flc`` file and is
placed into the ``acsql`` filesystem of JPEGs.  The JPEGs are then
used by the ``acsql`` web application to easily view ACS observaitons.

Authors
-------
    Matthew Bourque

Use
---
    This module is inteneded to be imported and used by
    ``acsql.ingest.ingest.py`` as such:
    ::

        from acsql.ingest.make_jpeg import make_jpeg
        make_jpeg(file_dict)

Dependencies
------------
    External library dependencies include:

    - ``astropy``
    - ``numpy``
    - ``PIL``
"""

import logging
import os

from astropy.io import fits
import numpy as np
from PIL import Image
```

```

def make_jpeg(file_dict):
    """Creates a JPEG for the given file.

    Parameters
    ----------
    file_dict : dict
        A dictionary containing various data useful for the ingestion
        process.
    """

    logging.info('{}: Creating JPEG'.format(file_dict['rootname']))

    hdulist = fits.open(file_dict['filename'], mode='readonly')
    data = hdulist[1].data

    # If the image is full-frame WFC, add on the other extension
    if len(hdulist) > 4 and hdulist[0].header['detector'] == 'WFC':
        if hdulist[4].header['EXTNAME'] == 'SCI':
            data2 = hdulist[4].data
            height = data.shape[0] + data2.shape[0]
            width = data.shape[1]
            new_array = np.zeros((height, width))
            new_array[0:int(height/2), :] = data
            new_array[int(height/2):height, :] = data2
            data = new_array

    # Clip the top and bottom 1% of pixels.
    top = np.percentile(data, 99)
    data[data > top] = top
    bottom = np.percentile(data, 1)
    data[data < bottom] = bottom

    # Scale the data.
    data = data - data.min()
    data = (data / data.max()) * 255.
    data = np.flipud(data)
    data = np.uint8(data)

    # Create parent JPEG directory if necessary
    jpg_dir = os.path.dirname(file_dict['jpg_dst'])
    if not os.path.exists(jpg_dir):
        os.makedirs(jpg_dir)
        logging.info('{}: Created directory {}'\
            .format(file_dict['rootname'], jpg_dir))

    # Write the image to a JPEG
    image = Image.fromarray(data)
    image.save(file_dict['jpg_dst'])

    # Close the hdulist
    hdulist.close()
```

acsql/ingest/make_thumbnail.py

```
"""Create a "Quicklook" Thumbail for the given observation.

A Thumbail image is created from a given JPEG file (see module
documentation for ``make_jpeg.py`` for further details). A
thumbnail is a JPEG image reduced to 128 x 128 pixel size.  The
thumbails are used by the ``acsql`` web application for quickly viewing
many JPEGs.

Authors
-------
    Matthew Bourque

Use
---
    This module is inteneded to be imported and used by
    ``acsql.ingest.ingest.py`` as such:
    ::

        from acsql.ingest.make_thumbnail import make_thumbnail
        make_thumbnail(file_dict)

Dependencies
```

```
------------
    External library dependencies include:

    - ``PIL``
"""

import logging
import os
import shutil

from PIL import Image


def make_thumbnail(file_dict):
    """Creates a 128 x 128 pixel 'thumbnail' JPEG for the given file.

    Parameters
    ----------
    file_dict : dict
        A dictionary containing various data useful for the ingestion
        process.
    """

    logging.info('{}: Creating Thumbnail'.format(file_dict['rootname']))

    # Create parent Thumbnail directory if necessary
    thumb_dir = os.path.dirname(file_dict['thumbnail_dst'])
    if not os.path.exists(thumb_dir):
        try:
            os.makedirs(thumb_dir)
            logging.info('{}: Created directory {}'\
                .format(file_dict['rootname'], thumb_dir))
        except FileExistsError:
            pass

    # Make a copy of the JPEG in the thumbnail directory
    shutil.copyfile(file_dict['jpg_dst'], file_dict['thumbnail_dst'])

    # Open the copied JPEG and reduce its size
    image = Image.open(file_dict['thumbnail_dst'])
    image.thumbnail((128, 128), Image.ANTIALIAS)
    image.save(file_dict['thumbnail_dst'], 'JPEG')
```

acsql/scripts/ingest_production.py

```
#! /usr/bin/env python

"""Performs ingestion of HST/ACS data into the ``acsql`` database and
filesystem.

This script is a wapper around ``acsql.ingest.ingest.py`` to ingest
multiple rootnames into the system.  The user may supply a list of
individual rootnames to ingest, or (by default) ingest whichever
rootnames exist in the MAST cache but yet to exist in the ``acsql``
database.

See ``acsql.ingest.ingest.py`` module docstrings for further
information on the ingestion process.

Authors
-------
    Matthew Bourque

Use
---
    This script is inteneded to be executed from the command line as
    such:
    ::

        python ingest_production.py [-i|--ingest_filelist]
            ['-f|--filetype']

    Parameters:
    (Optional) [-i|--ingest_filelist] - A text file containing
        individual rootnames to be ingested.  If not supplied, this
        module will determine which rootnames are to be ingested by
        comparing the MAST cache against what already exists in the
        ``acsql`` database.
```

```
    (Optional) [-f|--filetype] - The type of file to ingest.  May be
        an indivual filetype (e.g. ``flt``) or ``all`` to ingest all
        filetypes.  ``all`` is the default value.
"""

import argparse
import glob
import logging
import multiprocessing
import os

from astropy.io import fits

from acsql.database.database_interface import Master, session
from acsql.ingest.ingest import ingest
from acsql.utils.utils import SETTINGS, setup_logging, VALID_FILETYPES


def get_rootnames_to_ingest():
    """Return a list of paths to rootnames in the filesystem that need
    to be ingested (i.e. do not already exist in the ``acsql``
    database).

    Returns
    -------
    rootnames_to_ingest : list
        A list of full paths to rootnames that exist in the filesystem
        but not in the ``acsql`` database.
    """

    logging.info('Gathering files to ingest')

    # Query the database to determine which rootnames already exist
    results = session.query(Master.rootname).all()
    db_rootnames = set([item[0] for item in results])

    # Gather list of rootnames that exist in the filesystem
    fsys_paths = glob.glob(os.path.join(SETTINGS['filesystem'], 'j*', '*'))
    fsys_rootnames = set([os.path.basename(item)[:-1] for item in fsys_paths])

    # Determine new rootnames to ingest
    new_rootnames = fsys_rootnames - db_rootnames

    # Re-retreive the full paths
    rootnames_to_ingest = [item for item in fsys_paths if
                           os.path.basename(item)[:-1] in new_rootnames]

    logging.info('{} rootnames in database'.format(len(db_rootnames)))
    logging.info('{} rootnames in filesystem'.format(len(fsys_rootnames)))
    logging.info('{} rootnames to ingest'.format(len(rootnames_to_ingest)))

    return rootnames_to_ingest


def ingest_production(filetype, ingest_filelist):
    """Perform ingestion on the given filelist of rootnames (or if not
    provided, any new rootnames that exist in the MAST filesystem but
    not in the ``acsql`` database) for the given ``filetype`` (or all
    filetypes if ``filetype`` == 'all').

    Parameters
    ----------
    filetype : str
        The filetype to ingest (e.g. ``flt``, or ``all``)
    ingest_filelist : str or None
        The path to a file that contains rootnames to ingest.  If
        ``None``, then the acsql database and MAST filesystem are
        used to determine new rootnames to ingest.
    """

    if ingest_filelist:
        with open(ingest_filelist) as f:
            rootnames = f.readlines()
        rootnames = [rootname.strip().lower() for rootname in rootnames]
        rootnames = [os.path.join(SETTINGS['filesystem'], rootname[0:4], rootname) for rootname in
    rootnames]
    else:
```

```
        rootnames = get_rootnames_to_ingest()

    pool = multiprocessing.Pool(processes=SETTINGS['ncores'])
    filetypes = [filetype for item in rootnames]
    mp_args = [(rootname, filetype) for rootname, filetype in zip(rootnames, filetypes)]
    pool.starmap(ingest, mp_args)

    logging.info('Process Complete.')


def parse_args():
    """Parse command line arguments. Returns ``args`` object

    Returns
    -------
    args : obj
        An argparse object containing all of the arguments
    """

    VALID_FILETYPES.extend(['all'])

    # Create help strings
    filetype_help = 'The filetypes to ingest.  Can be one of the following: '
    filetype_help += '{}.  If "all", then all '.format(VALID_FILETYPES)
    filetype_help += 'availble filetypes for each rootname will be ingested. '
    filetype_help += 'If a specific filetype is given, then only that '
    filetype_help += 'filetype will be ingested. "all" is the default option.'
    ingest_filelist_help = 'A file containing a list of rootnames to ingest. '
    ingest_filelist_help += 'If not provided, then the acsql database is used '
    ingest_filelist_help += 'to determine which files get ingested.'

    # Add arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('-f --filetype',
                        dest='filetype',
                        action='store',
                        required=False,
                        default='all',
                        help=filetype_help)
    parser.add_argument('-i --ingest_filelist',
                        dest='ingest_filelist',
                        action='store',
                        required=False,
                        default=None,
                        help=ingest_filelist_help)

    # Parse args
    args = parser.parse_args()

    # Test the args
    test_args(args)

    return args


def test_args(args):
    """Test the command line arguments to ensure that they are valid.

    Parameters
    ----------
    args : obj
        An argparse objects containing all of the arguments.

    Raises
    ------
    AssertionError
        If any of the argument conditions fail.
    """

    # Ensure the filetype is valid
    VALID_FILETYPES.extend(['all'])
    assert args.filetype in VALID_FILETYPES,\
        '{} is not a valid filetype'.format(args.filetype)

    # Ensure that the ingest_filelist exists
    if args.ingest_filelist:
        assert os.path.exists(args.ingest_filelist),\
```

```
            '{} does not exist.'.format(args.ingest_filelist)


if __name__ == '__main__':

    module = os.path.basename(__file__).strip('.py')
    setup_logging(module)

    args = parse_args()
    ingest_production(args.filetype, args.ingest_filelist)
```

acsql/utils/utils.py

```
"""This module contains several functions that are useful to various
modules within the ``acsql`` package.  See individual function
docstrings for further information.

Authors
-------
    Matthew Bourque

Use
---

    The functions within this module are intened to be imported by
    various acsql modules and scripts, as such:
    ::

        from acsql.utils.utils import insert_or_update
        from acsql.utils.utils import SETTINGS
        from acsql.utils.utils import setup_logging

    There also exists static importable data:
    ::

        from acsql.utils.utils import FILE_EXTS
        from acsql.utils.utils import TABLE_DEFS

Dependencies
------------
    External library dependencies include:

    - ``acsql``
    - ``astropy``
    - ``numpy``
    - ``sqlalchemy``
"""

import datetime
import getpass
import glob
import logging
import os
import socket
import sys
import yaml

import astropy
import numpy
import sqlalchemy
from sqlalchemy import Table
from sqlalchemy.exc import DataError
from sqlalchemy.exc import IntegrityError
from sqlalchemy.exc import InternalError

import acsql

__config__ = os.path.realpath(os.path.join(os.getcwd(),
                                           os.path.dirname(__file__)))

# Define possible detector/filetype/extension combinations
WFC_FILE_EXTS = {'jif': [0, 1, 2, 3, 4, 5, 6],
                 'jit': [0, 1, 2, 3, 4, 5, 6],
                 'flt': [0, 1, 2, 3, 4, 5, 6],
                 'flc': [0, 1, 2, 3, 4, 5, 6],
                 'drz': [0, 1, 2, 3],
                 'drc': [0, 1, 2, 3],
                 'raw': [0, 1, 2, 3, 4, 5, 6],
```

```
                  'crj': [0, 1, 2, 3, 4, 5, 6],
                  'crc': [0, 1, 2, 3, 4, 5, 6],
                  'spt': [0, 1],
                  'asn': [0, 1]}

SBC_FILE_EXTS = {'jif': [0, 1, 2],
                 'jit': [0, 1, 2],
                 'flt': [0, 1, 2, 3],
                 'drz': [0, 1, 2, 3],
                 'raw': [0, 1, 2, 3],
                 'spt': [0, 1],
                 'asn': [0, 1]}

HRC_FILE_EXTS = {'jif': [0, 1, 2],
                 'jit': [0, 1, 2],
                 'flt': [0, 1, 2, 3],
                 'drz': [0, 1, 2, 3],
                 'raw': [0, 1, 2, 3],
                 'crj': [0, 1, 2, 3],
                 'spt': [0, 1],
                 'asn': [0, 1]}

# Define ingestable filetypes
VALID_FILETYPES = ['jif', 'jit', 'flt', 'flc', 'drz', 'drc', 'raw', 'crj',
                   'crc', 'spt', 'asn']

# Define value proposal types
VALID_PROPOSAL_TYPES = ['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']


def get_settings():
    """Returns the settings that are located in the acsql config file.

    Returns
    -------
    settings : dict
        A dictionary with setting key/value pairs.
    """

    with open(os.path.join(__config__, 'config.yaml'), 'r') as f:
        settings = yaml.load(f)

    return settings


SETTINGS = get_settings()


def setup_logging(module):
    """Configures a log file that logs the execution of the given
    module.  Log files are written to the log_dir that is set in the
    config.yaml configuration file.  The filename of the log file is
    <module>_<timestamp>.log.

    Parameters
    ----------
    module : str
        The name of the module to log.
    """

    SETTINGS = get_settings()

    # Configure logging
    timestamp = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M')
    filename = '{0}_{1}.log'.format(module, timestamp)
    logfile = os.path.join(SETTINGS['log_dir'], filename)
    logging.basicConfig(
        filename=logfile,
        format='%(asctime)s %(levelname)s: %(message)s',
        datefmt='%m/%d/%Y %H:%M:%S',
        level=logging.INFO)

    # Log environment information
    logging.info('User: {0}'.format(getpass.getuser()))
```

```
    logging.info('System: {0}'.format(socket.gethostname()))
    logging.info('Python Version: {0}'.format(sys.version.replace('\n', '')))
    logging.info('Python Path: {0}'.format(sys.executable))
    logging.info('Numpy Version: {0}'.format(numpy.__version__))
    logging.info('Numpy Path: {0}'.format(numpy.__path__[0]))
    logging.info('Astropy Version: {0}'.format(astropy.__version__))
    logging.info('Astropy Path: {0}'.format(astropy.__path__[0]))
    logging.info('SQLAlchemy Version: {0}'.format(sqlalchemy.__version__))
    logging.info('SQLAlchemy Path: {0}'.format(sqlalchemy.__path__[0]))


def get_table_defs():
    """Return a dictionary containing the columns for each database
    table, as taken from the table_definition text files.

    Returns
    -------
    table_defs : dict
        A dictionary whose keys are detector/file_type/extension
        configurations (e.g. 'wfc_flt_0') and whose values are lists
        of column names for the corresponding table.
    """

    # Get table definition files
    table_def_directory = os.path.realpath(os.path.join(os.getcwd(),
                                           os.path.dirname(__file__)))
    table_def_directory = table_def_directory.replace('utils', 'database/table_definitions/')
    table_def_files = glob.glob(os.path.join(table_def_directory, '*.txt'))

    table_defs = {}

    for table_def_file in table_def_files:

        configuration = os.path.basename(table_def_file).split('.txt')[0]
        with open(table_def_file, 'r') as f:
            contents = f.readlines()
        contents = [item.strip() for item in contents]
        columns = [item.split(',')[0] for item in contents]
        table_defs[configuration] = columns

    return table_defs

TABLE_DEFS = get_table_defs()


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()
```

acsql/website/acsql_webapp.py

```
#! /usr/bin/env python

"""This module serves as the ``acsql`` web application, which allows
users to view image data and interact with the ``acsql`` database.

The module is build using the ``flask`` python web framework.  See
accompanying ``data_containers``, ``query_form``, and ``form_choices``
modules for further information.

Authors
-------

    - Matthew Bourque
    - Meredith Durbin

Use
---

    This module is intended to be executed on a web server, but it can
    also be run locally:
    ::

        python acsql_webapp.py
        <go to localhost:5000 in a browser>

Dependencies
------------

    - ``acsql``
    - ``flask``
    - ``numpy``
"""

from collections import OrderedDict
import glob
import os

from flask import Flask, render_template, request, Response
import numpy as np

from acsql.utils.utils import SETTINGS
from acsql.website.data_containers import get_view_image_dict
from acsql.website.data_containers import get_view_proposal_dict
from acsql.website.data_containers import get_view_query_results_dict
from acsql.website.query_form import get_query_form
from acsql.website.query_lib import generate_csv
from acsql.website.query_lib import get_query_results

app = Flask(__name__)


@app.route('/archive/')
def archive():
    """Returns webpage containing links to all ACS archive proposals.

    Returns
    -------
    template : obj
        The ``archive.html`` template.
    """

    # Get list of all proposal numbers
    proposal_list = glob.glob(os.path.join(SETTINGS['jpeg_dir'], '*'))
    proposal_list = sorted([int(os.path.basename(item)) for item in proposal_list])

    # rearrange list so that it appears in multiple columns
```

```
    ncols = 12
    if len(proposal_list) % ncols != 0:
            proposal_list.extend(['']  (ncols - (len(proposal_list) % ncols)))
    proposal_array = np.asarray(proposal_list).reshape(ncols, int(len(proposal_list) / ncols)).T

    return render_template('archive.html', proposal_array=proposal_array)


@app.route('/database/')
@app.route('/database/results')
def database():
    """Returns webpage containing a query form for querying the
    ``acsql`` database.

    Returns
    -------
    template : obj
        The ``database.html`` webpage.
    """

    query_form = get_query_form(request.args)

    if request.query_string:
        if query_form.validate():
            query_form_dict = request.args.to_dict(flat=False)
            query_results_dict = get_query_results(query_form_dict)

            results = query_results_dict['query_results']
            num_results = query_results_dict['num_results']
            output_format = query_results_dict['output_format']
            output_columns = query_results_dict['output_columns']

            # If something went wrong with the query
            if num_results is None:
                template = render_template(
                    'database_error.html',
                    form=query_form,
                    msg=num_results)

            # If the query returned no results
            elif num_results == 0:
                results = True
                template = render_template(
                    'database_table.html',
                    results=results,
                    num_results=num_results)

            # If the query returned results
            else:

                # For HTML table output format
                if output_format == ['table']:
                    template = render_template(
                        'database_table.html',
                        results=results,
                        num_results=num_results,
                        output_columns=output_columns)

                # For CSV output format
                elif output_format == ['csv']:
                    template = Response(generate_csv(output_columns, results), mimetype='text/csv')
                    template.headers['Content-Disposition'] = 'attachment; filename=query_results.csv'

                # For Thumbnail output format
                elif output_format == ['thumbnails']:
                    thumbnail_dict = get_view_query_results_dict(query_results_dict)
                    template = render_template('view_query_results.html', thumbnail_dict=thumbnail_dict)

                else:
                    template = render_template('database_error.html', form=query_form)

            return template

        # Form was not validated
        else:
            return render_template('database_error.html', form=query_form)

```

```
    else:
        return render_template('database.html', form=query_form)


def handle_500(trace):
    """Handle 500 error.

    Parameters
    ----------
    trace : str
        The traceback of the error.

    Returns
    -------
    template : obj
        The ``500.html`` template.
    """

    trace_html = trace.replace('\n', '<br>')

    return render_template('500.html', trace_html=trace_html)


@app.route('/')
def main():
    """Generates the ``acsql`` website homepage.

    Returns
    -------
    template : obj
        The ``index.html`` template.
    """

    return render_template('index.html')


@app.errorhandler(404)
def page_not_found(error):
    """Redirects any nonexistent URL to 404 page.

    Parameters
    ----------
    error : obj
        The ``error`` thrown.

    Returns
    -------
    template : obj
        The ``404.html`` template.
    """

    return render_template('404.html'), 404


# @app.route('/archive/<proposal>/<filename>/<fits_type/header/')
# def view_header(proposal, filename, fits_type):
#     """
#     """

#     header_dict = get_view_header_dict()
#     return render_template('header.html', header_dict=header_dict)


@app.route('/archive/<proposal>/<filename>/')
@app.route('/archive/<proposal>/<filename>/<fits_type>/')
def view_image(proposal, filename, fits_type='flt'):
    """Returns webpage for viewing a single JPEG image, along with some
    useful metadata and links to additional information/downloads.

    If an invalid ``fits_type`` is supplied, a 404 page is returned.

    Parameters
    ----------
    proposal : str
        The proposal ID (e.g. ``'12345'``).
    filename : str
        The 9-character IPPPSSOOT rootname (e.g. ``jcye04zsq``.)
```

```
    fits_type : str
        The JPEG FITS type to view. Can either be ``raw``, ``flt``, or
        ``flc``.

    Returns
    -------
    template : obj
        The ``view_image.html`` template.
    """

    if fits_type in ['raw', 'flt', 'flc']:
        image_dict = get_view_image_dict(proposal, filename, fits_type)
        return render_template('view_image.html', image_dict=image_dict)
    else:
        return render_template('404.html'), 404


@app.route('/archive/<proposal>/')
def view_proposal(proposal):
    """Returns webpage for viewing all thumbnails for a given
    ``proposal``, along with some metadata and links to additional
    information.

    If an invalid proposal is supplied, a 404 page is returned.

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

    Returns
    -------
    template : obj
        The ``view_proposal.html`` template.
    """

    proposal_list = glob.glob(os.path.join(SETTINGS['jpeg_dir'], '*'))
    proposal_list = [item.split('/')[-1] for item in proposal_list]

    if proposal in proposal_list:
        proposal_dict = get_view_proposal_dict(proposal)
        return render_template('view_proposal.html', proposal_dict=proposal_dict)
    else:
        return render_template('404.html'), 404


if __name__ == '__main__':

    app.run()
```

acsql/website/data_containers.py

```
"""Various functions for creating and returning various data to be used
by the ``acsql`` web application.

This module contains functions to obtains image and proposal metadata
for use by the ``acsql`` web application.  See the ``acsql_webapp``
module for further information about the web application.

Authors
-------

    - Matthew Bourque
    - Meredith Durbin

Use
---

    This module is intended to be imported and used by ``acsql_webapp``
    as such:
    ::

        from acsql.website.data_containers import get_view_image_dict
        from acsql.website.data_containers import get_view_proposal_dict

        image_dict = get_view_image_dict(proposal, filename, fits_type)
        proposal_dict = get_view_proposal_dict(proposal)

```

```
Dependencies
------------

    - ``acsql``
"""


import glob
import html
import os
import requests

from acsql.database import database_interface
from acsql.database.database_interface import Master
from acsql.utils.utils import SETTINGS


def _get_image_lists(data_dict, fits_type):
    """Add a list of JPEG and Thumbnail paths to the ``data_dict``
    dictionary.

    Parameters
    ----------
    data_dict : dict
        A dictionary containing data used to render a webpage.
    fits_type : str
        The FITS type.  Can either be ``raw``, ``flt``, or ``flc``.

    Returns
    -------
    data_dict : dict
        A dictionary containing data used to render a webpage.
    """

    jpeg_proposal_path = os.path.join('static/img/jpegs/', data_dict['proposal_id'])
    thumb_proposal_path = os.path.join('static/img/thumbnails/', data_dict['proposal_id'])

    data_dict['jpegs'] = sorted(glob.glob(os.path.join(jpeg_proposal_path, '*flt.jpg')))
    data_dict['thumbs'] = sorted(glob.glob(os.path.join(thumb_proposal_path, '*flt.thumb')))

    return data_dict


def _get_metadata_from_database(data_dict):
    """Add observation metadata (e.g. ``aperture``, ``exptime``, etc.)
    to the ``data_dict`` by querying the ``acsql`` database.

    Parameters
    ----------
    data_dict : dict
        A dictionary containing data used to render a webpage.

    Returns
    -------
    data_dict : dict
        A dictionary containing data used to render a webpage.
    """

    session = getattr(database_interface, 'session')

    results = []
    for rootname in data_dict['rootnames']:

        detector = session.query(Master.detector)\
            .filter(Master.rootname == rootname).one()[0]

        table = getattr(database_interface, '{}_raw_0'.format(detector))
        result = session.query(
            table.aperture, table.exptime, table.filter1, table.filter2,
            table.targname, table.date_obs, table.time_obs, table.expstart,
            table.expflag, table.quality, table.ra_targ, table.dec_targ,
            table.pr_inv_f, table.pr_inv_l).filter(table.rootname == rootname).one()
        result = [item for item in result]
        result.append(detector)
        results.append(result)

    session.close()
```

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

```
        status_string = req.content.decode()
        data_dict['proposal_title'] = html.unescape(status_string.\
            split('<b>Title:</b> ')[1].\
            split('<br>')[0])
        data_dict['cycle'] = html.unescape(status_string.\
            split('<b>Cycle:</b> ')[1].\
            split('<br>')[0])
        data_dict['schedule'] = html.unescape(status_string.\
            split('/proposal-help-HST.html#')[2].\
            split('">')[0])

    else:
        print('Request failed: {}'.format(data_dict['status_page']))
        data_dict['proposal_title'] = 'proposal title unavailable'
        data_dict['cycle'] = None
        data_dict['schedule'] = None

    return data_dict


def _initialize_data_dict(proposal, fits_type='flt'):
    """Create and return a dictionary containing commonly used data
    amongst ``/archive/<proposal>/`` and
    ``/archive/<proposal>/<filename>`` webpages.

    Parameters
    ----------
    proposal : str
        The proposal number (e.g. ``12345``).
    fits_type : str
        The FITS type.  Can be ``raw``, ``flt``, or ``flc``.

    Returns
    -------
    data_dict : dict
        A dictionary containing data used to render a webpage.
    """

    data_dict = {}
    data_dict['proposal_id'] = proposal
    data_dict = _get_image_lists(data_dict, fits_type)
    data_dict['rootnames'] = [os.path.basename(item).split('_')[0][:-1] for item in data_dict['jpegs']]
    data_dict['filenames'] = [os.path.basename(item).split('_')[0] for item in data_dict['jpegs']]
    data_dict['num_images'] = len(data_dict['jpegs'])
    data_dict = _get_proposal_status(data_dict)

    return data_dict


# def get_view_header_dict(filename, fits_type='flt'):
#     """
#     """

#     header_dict = {}
#     header_dict['filename'] = filename
#     header_dict['fits_type'] = fits_type.upper

#     return header_dict

def get_view_image_dict(proposal, filename, fits_type='flt'):
    """Return a dictionary containing data used for the
    ``/archive/<proposal>/<filename>/`` webpage.

    Parameters
    ----------
    proposal : str
        The proposal number (e.g. ``12345``).
    filename : str
        The 9-character IPPPSSOOT rootname (e.g. ``jcye04zsq``.)
    fits_type : str
        The JPEG FITS type to view. Can either be ``raw``, ``flt``, or
        ``flc``.

    Returns
    -------
    image_dict : dict
```

```
        A dictionary containing data used for the
        ``/archive/<proposal>/<filename>`` webpage.
    """

    image_dict = _initialize_data_dict(proposal, fits_type)
    image_dict['fits_type'] = fits_type.upper()
    image_dict['filename'] = filename
    image_dict['rootname'] = filename[:-1]
    image_dict = _get_metadata_from_database(image_dict)
    image_dict['index'] = image_dict['filenames'].index(image_dict['filename'])
    image_dict['page'] = image_dict['index'] + 1
    image_dict['expstart'] = image_dict['expstarts'][image_dict['index']]
    image_dict['filter1'] = image_dict['filter1s'][image_dict['index']]
    image_dict['filter2'] = image_dict['filter2s'][image_dict['index']]
    image_dict['aperture'] = image_dict['apertures'][image_dict['index']]
    image_dict['exptime'] = image_dict['exptimes'][image_dict['index']]
    image_dict['expflag'] = image_dict['expflags'][image_dict['index']]
    image_dict['quality'] = image_dict['qualitys'][image_dict['index']]
    image_dict['ra'] = image_dict['ras'][image_dict['index']]
    image_dict['dec'] = image_dict['decs'][image_dict['index']]
    image_dict['targname'] = image_dict['targnames'][image_dict['index']]
    image_dict['pi_first_name'] = image_dict['pi_firsts'][image_dict['index']]
    image_dict['pi_last_name'] = image_dict['pi_lasts'][image_dict['index']]
    image_dict['view_url'] = 'archive/{}/{}/{}'.format(image_dict['proposal_id'], image_dict['filename'],
    fits_type)
    image_dict['fits_links'] = {}
    image_dict['first'] = image_dict['index'] == 0
    image_dict['last'] = image_dict['index'] == image_dict['num_images'] - 1

    # Determine path to JPEG
    jpeg_path = '/static/img/jpegs/{}/{}_{}.jpg'.format(image_dict['proposal_id'], image_dict['filename'],
    fits_type)
    jpeg_path_abs = os.path.join(SETTINGS['jpeg_dir'], image_dict['proposal_id'], '{}_{}.jpg'.format(
    image_dict['filename'], fits_type))
    if os.path.exists(jpeg_path_abs):
        image_dict['image'] = jpeg_path
    else:
        image_dict['image'] = None

    # Determine next and previous images, if possible
    if not image_dict['last']:
        image_dict['next'] = {'proposal': image_dict['proposal_id'], 'filename': image_dict['filenames'][
    image_dict['index'] + 1], 'fits_type': fits_type}
    if not image_dict['first']:
        image_dict['prev'] = {'proposal': image_dict['proposal_id'], 'filename': image_dict['filenames'][
    image_dict['index'] - 1], 'fits_type': fits_type}

    # Determine other available JPEGs for given observation
    jpeg_types = glob.glob(jpeg_path_abs.replace('{}.jpg'.format(fits_type), '*.jpg'))
    jpeg_types = [os.path.basename(item).split('_')[-1].split('.')[0] for item in jpeg_types]
    jpeg_types = [item for item in jpeg_types if item.upper() != image_dict['fits_type']]
    image_dict['available_jpegs'] = {}
    for jpeg_type in jpeg_types:
        image_dict['available_jpegs'][jpeg_type] = image_dict['view_url'].replace(fits_type, jpeg_type)

    # For downloading the files
    # image_dict['proposal_name'] = image_dict['filename'][0:4]
    # image_dict['fits_links']['FLT'] = os.path.join(
    #                                     SETTINGS['filesystem'],
    #                                     image_dict['proposal_name'],
    #                                     image_dict['filename'],
    #                                     '{}_flt.fits'.format(image_dict['filename']))

    return image_dict


def get_view_proposal_dict(proposal):
    """Return a dictionary containing data used for the
    ``/archive/<proposal>/`` webpage.

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

    Returns
    -------
```

```
    proposal_dict : dict
        A dictionary containing data used for the
        ``/archive/<proposal>/`` webpage.
    """

    proposal_dict = _initialize_data_dict(proposal)
    proposal_dict['visits'] = [os.path.basename(item).split('_')[0][4:6].upper() for item in proposal_dict[
    'jpegs']]
    proposal_dict['num_visits'] = len(set(proposal_dict['visits']))
    proposal_dict = _get_metadata_from_database(proposal_dict)
    proposal_dict = _get_buttons_dict(proposal_dict)
    proposal_dict['viewlinks'] = ['/archive/{}/{}/'.format(proposal_dict['proposal_id'], filename) for
    filename in proposal_dict['filenames']]

    return proposal_dict


def get_view_query_results_dict(query_results_dict):
    """Return a dictionary containing data used for the
    ``/database/results/`` webpage.

    Parameters
    ----------
    query_results_dict : dict
        A dictionary containing the results of the query performed
        through the ``/database/`` webpage, along with some additional
        metadata.

    Returns
    -------
    thumbnail_dict : dict
        A dictionary containing data used for the ``/database/results/``
        webpage.
    """

    query_results = query_results_dict['query_results']

    thumbnail_dict = {}
    thumbnail_dict['num_images'] = query_results_dict['num_results']
    thumbnail_dict['rootnames'] = [item[3] for item in query_results]
    thumbnail_dict['filenames'] = [item[4].split('_')[0] for item in query_results]
    thumbnail_dict['detectors'] = [item[5] for item in query_results]
    thumbnail_dict['expstarts'] = [item[6] for item in query_results]
    thumbnail_dict['filter1s'] = [item[7] for item in query_results]
    thumbnail_dict['filter2s'] = [item[8] for item in query_results]
    thumbnail_dict['exptimes'] = [item[9] for item in query_results]
    thumbnail_dict['targnames'] = [item[10] for item in query_results]
    thumbnail_dict['proposal_ids'] = [item[11] for item in query_results]
    thumbnail_dict['visits'] = [item[4:6] for item in thumbnail_dict['rootnames']]
    thumbnail_dict = _get_buttons_dict(thumbnail_dict)
    thumbnail_dict['thumbs'] = ['static/img/thumbnails/{}/{}_flt.thumb'.format(proposid, filename)
        for proposid, filename in zip(thumbnail_dict['proposal_ids'], thumbnail_dict['filenames'])]
    thumbnail_dict['viewlinks'] = ['/archive/{}/{}/'.format(proposid, filename)
        for proposid, filename in zip(thumbnail_dict['proposal_ids'], thumbnail_dict['filenames'])]

    return thumbnail_dict
```

acsql/website/form_options.py

```
"""Provides a dictionary containing ``acsql`` database query form data
for use by the ``acsql`` web application.

Authors
-------

    - Matthew Bourque
    - Meredith Durbin

Use
---

    The dictionary contained in this module is inteded to be imported
    as such:
    ::

        from acsql.website.form_options import FORM_OPTIONS
"""

```

```
APERTURES = ['WFC', 'WFC-FIX', 'WFC1', 'WFC1-1K', 'WFC1-2K', 'WFC1-512',
    'WFC1-CTE', 'WFC1-FIX', 'WFC1-IRAMP', 'WFC1-IRAMPQ', 'WFC1-MRAMP',
    'WFC1-MRAMPQ', 'WFC1-POL0UV', 'WFC1-POL0V', 'WFC1-POL120UV', 'WFC1-POL120V',
    'WFC1-POL60UV', 'WFC1-POL60V', 'WFC1A-1K', 'WFC1A-2K', 'WFC1A-512', 'WFC1B-1K',
    'WFC1B-2K', 'WFC1B-512', 'WFC2', 'WFC2-2K', 'WFC2-FIX', 'WFC2-MRAMP',
    'WFC2-ORAMP', 'WFC2-ORAMPQ', 'WFC2C-1K', 'WFC2C-2K', 'WFC2C-512', 'WFC2D-1K',
    'WFC2D-2K', 'WFC2D-512', 'WFCENTER', 'HRC', 'HRC-512', 'HRC-ACQ', 'HRC-CORON1.8',
    'HRC-CORON3.0', 'HRC-FIX', 'HRC-OCCULT0.8', 'HRC-SUB1.8', 'SBC', 'SBC-FIX']
DETECTORS = ['WFC','HRC','SBC']
FILTER1S = ['F115LP', 'F122M', 'F125LP', 'F140LP', 'F150LP', 'F165LP',
    'F475W', 'F502N', 'F550M', 'F555W', 'F606W', 'F625W', 'F658N', 'F775W',
    'F850LP', 'F892N', 'G800L', 'POL0UV', 'POL60UV', 'POL120UV' 'PR110L',
    'PR130L', 'CLEAR1L', 'CLEAR1S', 'BLOCK1', 'BLOCK3', 'BLOCK4', 'NotCmded']
FILTER2S = ['F220W', 'F250W', 'F330W', 'F344N', 'F435W', 'F660N', 'F814W',
    'FR388N', 'FR423N', 'FR462N', 'FR459M', 'FR505N', 'FR551N', 'FR601N',
    'FR647M', 'FR656N', 'FR716N', 'FR782N', 'FR853N', 'FR914M', 'FR931N',
    'FR1016N', 'POL0V', 'POL60V', 'POL120V', 'PR200L', 'CLEAR2L', 'CLEAR2S',
    'NotCmded']
IMAGETYPS = ['BIAS', 'DARK', 'FLAT', 'EXT']
OBSTYPES = ['IMAGING', 'SPECTROSCOPC', 'CORONOGRAPHIC', 'INTERNAL']
OUTPUT_COLUMNS = [('rootname','Rootname'), ('detector','Detector'),
    ('proposal_type','Proposal Type'), ('pr_inv_l','PI Last Name'),
    ('pr_inv_f','PI First Name'), ('proposid','Proposal ID'), ('filter1','Filter1'),
    ('filter2','Filter2'), ('aperture','Aperture'), ('expstart','Expstart'),
    ('date_obs','Date of Observation'), ('time_obs','Time of Observation'),
    ('targname','Target Name'), ('ra_targ','Target RA'), ('dec_targ','Target Dec'),
    ('obstype','Observation Type'), ('obsmode','Observation Mode'),
    ('subarray','Subarray'), ('imagetyp', 'Image Type'), ('asn_id','Association ID')]
OUTPUT_FORMAT = [('table','HTML table'), ('csv','CSV'), ('thumbnails','Thumbnails')]
PROPOSAL_TYPES = ['GO', 'GTO/ACS', 'CAL/ACS', 'SM3/ACS', 'SM3/ERO', 'SNAP',
    'GO/PAR', 'GO/DD', 'GTO/COS', 'CAL/OTA', 'ENG/ACS', 'NASA', 'SM4/ACS',
    'SM4/ERO', 'SM4/COS', 'CAL/WFC3', 'CAL/STIS']


FORM_OPTIONS = {}
FORM_OPTIONS['aperture'] = [(aperture, aperture) for aperture in APERTURES]
FORM_OPTIONS['detector'] = [(detector, detector) for detector in DETECTORS]
FORM_OPTIONS['filter1'] = [(filter1, filter1) for filter1 in FILTER1S]
FORM_OPTIONS['filter2'] = [(filter2, filter2) for filter2 in FILTER2S]
FORM_OPTIONS['imagetyp'] = [(imagetyp, imagetyp) for imagetyp in IMAGETYPS]
FORM_OPTIONS['obstype'] = [(obstype, obstype) for obstype in OBSTYPES]
FORM_OPTIONS['output_columns'] = OUTPUT_COLUMNS
FORM_OPTIONS['output_format'] = OUTPUT_FORMAT
FORM_OPTIONS['proposal_type'] = [(prop_type, prop_type) for prop_type in PROPOSAL_TYPES]
```

acsql/website/query_form.py

```
"""Contains class objects for building a query form for querying the
``acsql`` database through the ``acsql`` web application.

Many of the class objects are subclasses or extensions from components
provided by the ``wtforms`` library.  Hard coded data such as form
options are imported from the ``form_options`` module.

Authors
-------

    - Matthew Bourque
    - Meredith Durbin

Use
---

    This module is inteded to be imported and used by the
    ``acsql_webapp`` module as such:
    ::

        from acsql.website.query_form import get_query_form
        query_form = get_query_form()

Dependencies
------------

    - ``acsql``
    - ``wtforms``
    - ``wtforms_components``
"""

```

```
from wtforms import DateField
from wtforms import DecimalField
from wtforms import Form
from wtforms import FormField
from wtforms import RadioField
from wtforms import SelectField
from wtforms import SelectMultipleField
from wtforms import TextField
from wtforms import validators
from wtforms import widgets
from wtforms_components.fields import IntegerField

from acsql.website.form_options import FORM_OPTIONS


operator_form = SelectField('Operator',
                [validators.Optional()],
                choices=[('=', '='), ('<', '<'), ('>','>'), ('between','between')],
                default=('=', '='))


class CheckboxField(SelectMultipleField):
    """Like a ``SelectField``, except displays a list of checkbox
    buttons.

    Parameters
    ----------
    SelectMultipleField : obj
        The ``SelectMultipleField`` object from ``wtforms``
    """

    widget = widgets.ListWidget(prefix_label=False)
    option_widget = widgets.CheckboxInput()


class DateForm(Form):
    """Creates a ``DateForm`` object that allows for date input in a
    form field.

    Parameters
    ----------
    Form : obj
        The ``Form`` object from ``wtforms``.
    """

    op = operator_form
    val1 = DateField('Date Observed',
         [validators.Optional()],
         description='YYYY-MM-DD',
         format='%Y-%m-%d')
    val2 = DateField('dateobs2',
         [validators.Optional()],
         description='YYYY-MM-DD',
         format='%Y-%m-%d')


class ExptimeForm(Form):
    """Creates a ``ExptimeForm`` object that allows for ``exptime``
    input in a form field.

    Parameters
    ----------
    Form : obj
        The ``Form`` object from ``wtforms``.
    """
    op = operator_form
    val1 = DecimalField('Exposure Time', [validators.Optional()])
    val2 = DecimalField('exptime2', [validators.Optional()])


class MultiCheckboxField(SelectMultipleField):
    """A multiple-select, except displays a list of checkboxes.

    Parameters
    ----------
    SelectMultipleField : obj
        The ``SelectMultipleField`` object from ``wtforms``
```

```
    """

    widget = widgets.ListWidget(prefix_label=False)
    option_widget = widgets.CheckboxInput()


def is_field_value(form, fieldname, value, negate=False):
    """Helper function to check if the given field in the given form is
    of a specified value.

    Parameters
    ----------
    form: obj
        The form to test on
    fieldname : str
        The fieldname to test value against. If not found an Exception
        is raised.
    value : str
        Value to test for.
    negate : boolean
        True/False to invert the result.
    """

    field = form._fields.get(fieldname)
    if field is None:
        raise Exception('Invalid field "%s"' % fieldname)
    test = value == field.data
    test = not test if negate else test

    return test


class RequiredIf(validators.Required):
    """Custom validator to enforce requires only if another field
    matches a specified value. the ``negate`` allows for inverting
    the result.

    Parameters
    ----------
    validators.Required : obj
        The ``validators.Required`` object from ``wtforms``.
    """

    def __init__(self, other_fieldname, value, negate, *args, **kwargs):
        self.other_fieldname = other_fieldname
        self.negate = negate
        self.value = value
        super(RequiredIf, self).__init__(*args, **kwargs)

    def __call__(self, form, field):
        if is_field_value(form, self.other_fieldname, self.value, self.negate):
            super(RequiredIf, self).__call__(form, field)


class QueryForm(Form):
    """Form for querying the ``acsql`` database.

    Parameters
    ----------
    Form : obj
        The ``Form`` object from ``wtforms``
    """

    rootname = TextField('Rootname',
               [validators.Optional()],
               description='Single rootname (IPPPSSOOT) or comma-separated list span6')
    targname = TextField('Target Name',
               [validators.Optional()],
               description='ex. OMEGACEN, NGC-3603, IRAS05129+5128; single or comma-separated span6')
    proposid = IntegerField('Proposal ID',
               [validators.Optional(),
               validators.NumberRange(min=8183, max=19999,
               message='Please enter a valid proposal ID')],
               description='span4')
    date_obs = FormField(DateForm,
               'Date Observed',
               description='span4')
```

```
    exptime = FormField(ExptimeForm,
              'Exposure Time',
              description='span4')
    proposal_type = CheckboxField('Proposal Type',
                    [validators.Optional()],
                    description='span3',
                    choices=FORM_OPTIONS['proposal_type'])
    detector = CheckboxField('Detector',
               [validators.Optional()],
               description='span3',
               choices=FORM_OPTIONS['detector'])
    obstype = CheckboxField('Observation Type',
              [validators.Optional()],
              description='span3',
              choices=FORM_OPTIONS['obstype'])
    aperture = SelectMultipleField('Aperture',
               [validators.Optional()],
               choices=FORM_OPTIONS['aperture'],
               description='span3')
    filter1 = SelectMultipleField('Filter1',
              [validators.Optional()],
              description='span3',
              choices=FORM_OPTIONS['filter1'])
    filter2 = SelectMultipleField('Filter2',
              [validators.Optional()],
              description='span3',
              choices=FORM_OPTIONS['filter2'])
    imagetyp = SelectMultipleField('Image Type',
               [validators.Optional()],
               choices=FORM_OPTIONS['imagetyp'],
               description='span3')
    pr_inv_l = TextField('PI Last Name',
               [validators.Optional()],
               description='Single or comma-separated span3')
    pr_inv_f = TextField('PI First Name',
               [validators.Optional()],
               description='Single or comma-separated span3')
    output_columns = MultiCheckboxField('Output Columns',
                     [RequiredIf('output_format', 'thumbnails', True,
                     message='Please select at least one output column.')],
                     choices=FORM_OPTIONS['output_columns'])
    output_format = RadioField('Output Format',
                    [validators.Required(message='Please select an output format.')],
                    choices=FORM_OPTIONS['output_format'],
                    default='thumbnails', description='span3')


def get_query_form(Form):
    """Return the ``QueryForm`` object that contains query form
    components

    Parameters
    ----------
    Form : obj
        A request form.

    Returns
    -------
    query_form : obj
        The ``QueryForm`` object, which contains the various components
        to build the ACS Database query form.
    """

    query_form = QueryForm(Form)

    return query_form
```

acsql/website/query_lib.py

```
"""Contains various functions to support the ``/database/`` webpage of
the ``acsql`` web application.

Functions include those that parse, build, validate, and return
``SQLAlchemy`` ``query`` objects in order to perform a database query
through the web application.

Authors
-------

    - Matthew Bourque
    - Meredith Durbin

Use
---

    This module is intended to be imported and used by the
    ``acsql_webapp`` module as such:
    ::

        from query_lib import generate_csv
        from query_lib import get_query_results

        generate_csv(output_columns, results)
        get_query_results(query_form_dict)

Dependencies
------------

    - ``acsql``
    - ``sqlalchemy``
"""

from sqlalchemy import create_engine
from sqlalchemy import literal_column
from sqlalchemy import or_
from sqlalchemy.orm import sessionmaker

from acsql.database.database_interface import Master
from acsql.database.database_interface import WFC_raw_0
from acsql.database.database_interface import HRC_raw_0
from acsql.database.database_interface import SBC_raw_0
from acsql.utils.utils import SETTINGS


def _apply_query_filter(table, key, values, query):
    """Apply a filter to the given ``query`` based on the ``table``,
    ``key``, and ``value``.

    Parameters
    ----------
    table : obj
        The ``SQLAlchemy`` table object associated with the table to
        apply the filter to.
    key : str
        The keyword for the column to filter on.
    values : obj
        The requested values of the ``key``.
    query : obj
        The ``SQLAlchemy`` ``query`` objecgt to perform the filtering
        on.

    Returns
    -------
    query : obj
        The ``SQLAlchemy`` ``query`` object with filter applied.
    """

    # Fields that accpet comma-separated lists and wildcards
    csv_keys = ['rootname', 'targname', 'pr_inv_l', 'pr_inv_f']

    # Fields that allow operators
    operator_keys = ['date_obs', 'exptime']

    # Parse the key/value pairs for comma-separated values
    if key in csv_keys:
```

```
        parsed_value = values[0].replace(' ', '').split(',')
        parsed_value = [item.replace('*', '%') for item in parsed_value]
        conditions = [getattr(table, key).like(val) for val in parsed_value]
        query = query.filter(or_(*conditions))

    # Parse the key/value pairs for operator keys
    elif key in operator_keys:
        if values['op'] == 'between':
            if len(values) == 3:
                if float(values['val1'].replace('-', '')) < float(values['val2'].replace('-', '')):
                    query = query.filter(getattr(table, key).between(values['val1'], values['val2']))
                elif float(values['val1'].replace('-', '')) > float(values['val2'].replace('-', '')):
                    query = query.filter(getattr(table, key).between(values['val2'], values['val1']))
                elif float(values['val1'].replace('-', '')) == float(values['val2'].replace('-', '')):
                    raise ValueError('Values submitted for field "{}" must be different.'.format(key))
                else:
                    raise ValueError('Invalid values submitted for field "{}".'.format(key))
            else:
                query = query.filter(getattr(table, key).op('=')(values['val1']))
        else:
            query = query.filter(getattr(table, key).op(values['op'])(values['val1']))

    # Else the filtering is straightforward
    query = query.filter(getattr(table, key).in_(values))

    return query


def _build_queries(output_columns):
    """Builds queries of appropriate tables and columns

    Parameters
    ----------
    output_columns : list
        List of columns desired for query output

    Returns
    -------
    wfc_query : obj
        Query object for requested collumns in WFC database tables.
    hrc_query : obj
        Query object for requested collumns in HRC database tables.
    sbc_query : obj
        Query object for requested collumns in SBC database tables.
    """

    # Determine which columns belong to which tables
    master_cols = [getattr(Master, col) for col in output_columns if hasattr(Master, col)]
    wfc_cols = [getattr(WFC_raw_0, col) for col in output_columns if hasattr(WFC_raw_0, col)]
    hrc_cols = [getattr(HRC_raw_0, col) for col in output_columns if hasattr(HRC_raw_0, col)]
    sbc_cols = [getattr(SBC_raw_0, col) for col in output_columns if hasattr(SBC_raw_0, col)]

    # Determine which columns are unique to a specific table
    wfc_only = [col for col in output_columns if hasattr(WFC_raw_0, col)
                and not hasattr(Master, col)
                and not hasattr(HRC_raw_0, col)
                and not hasattr(SBC_raw_0, col)]
    hrc_only = [col for col in output_columns if hasattr(HRC_raw_0, col)
                and not hasattr(Master, col)
                and not hasattr(WFC_raw_0, col)
                and not hasattr(SBC_raw_0, col)]
    sbc_only = [col for col in output_columns if hasattr(SBC_raw_0, col)
                and not hasattr(Master, col)
                and not hasattr(WFC_raw_0, col)
                and not hasattr(HRC_raw_0, col)]

    # Combine columns amongst tables
    master_wfc = master_cols + wfc_cols + [literal_column('"--"').label(col) for col in hrc_only] + \
                 [literal_column('"--"').label(col) for col in sbc_only]
    master_hrc = master_cols + hrc_cols + [literal_column('"--"').label(col) for col in wfc_only] + \
                 [literal_column('"--"').label(col) for col in sbc_only]
    master_sbc = master_cols + sbc_cols + [literal_column('"--"').label(col) for col in wfc_only] + \
                 [literal_column('"--"').label(col) for col in hrc_only]

    if len(master_cols) == 0:

        # For WFC queries
```

```
        if len(wfc_cols) == 0:
            wfc_query = False
        else:
            session = _get_session()
            wfc_query = session.query(*master_wfc)
            session.close()

        # For HRC queries
        if len(hrc_cols) == 0:
            hrc_query = False
        else:
            session = _get_session()
            hrc_query = session.query(*master_hrc)
            session.close()

        # For SBC queries
        if len(sbc_cols) == 0:
            sbc_query = False
        else:
            session = _get_session()
            sbc_query = session.query(*master_sbc)
            session.close()

    else:
        session = _get_session()
        wfc_query = session.query(*master_wfc).join(WFC_raw_0)
        hrc_query = session.query(*master_hrc).join(HRC_raw_0)
        sbc_query = session.query(*master_sbc).join(SBC_raw_0)

    return wfc_query, hrc_query, sbc_query


def _convert_query_form_dict(query_form_dict):
    """Converts raw output from ``form.to_dict()`` to a format that is
    more useable for ``acsql`` database queries.

    Parameters
    ----------
    query_form_dict : dict
        The dictionary returned by ``form.to_dict()``.

    Returns
    -------
    query_form_dict : dict
        A new dictionary with blank entried removed and operator key
        entries reformatted.
    """

    # Keys that allow operators (e.g. greater than)
    operator_keys = ['date_obs', 'exptime']

    # Remove blank entries from form data
    query_form_dict = {key: value for key, value in list(query_form_dict.items()) if value != ['']}

    # Combine data returned from fields with operator dropdowns
    for operator_key in operator_keys:
        operator_dict = {}
        for key, value in list(query_form_dict.items()):
            if key == operator_key + '-op':
                operator_dict['op'] = value[0]
            elif key == operator_key + '-val1':
                operator_dict['val1'] = value[0]
            elif key == operator_key + '-val2':
                operator_dict['val2'] = value[0]
        if len(operator_dict) > 1:
            query_form_dict[operator_key] = operator_dict

    return query_form_dict


def generate_csv(output_columns, results):
    """Create a CSV file of the database query ouput.

    Parameters
    ----------
    output_columns : list
        A list of columns desired for the output file.
```

```
    results : list
        A list of results from the database query
    """

    header = ','.join(output_columns) + '\n'
    yield header

    for result in results:
        if len(result) == 1:
            yield str(result[0]) + '\n'
        else:
            yield ','.join(map(str, result)) + '\n'


def get_query_results(query_form_dict):
    """Returns a dictionary with the results of the requested query
    along with some additional metadata.  Calls on several internal
    functions to build and perform the query in order to abstract
    out its complexity.

    Parameters
    ----------
    query_form_dict : dict
        A dictionary containing information about the requested query,
        such as the ``output_format``, ``output_columns`` and the
        requested values.

    Returns
    -------
    query_results_dict : dict
        A dictionary containing the query results as well as some
        metadata such as ``output_format`` and number of results.
    """

    # Determine output format
    output_format = query_form_dict.pop('output_format')
    if output_format == ['thumbnails']:
        output_columns = ['rootname', 'filename', 'detector',
                          'proposal_type', 'expstart', 'filter1',
                          'filter2', 'exptime', 'targname', 'proposid']
        if 'output_columns' in query_form_dict:
            query_form_dict.pop('output_format', None)
    else:
        output_columns = query_form_dict.pop('output_columns')

    # Remove blank entries from form data
    query_form_dict = _convert_query_form_dict(query_form_dict)

    # Build the query
    wfc_query, hrc_query, sbc_query = _build_queries(output_columns)

    # Perform filtering on the query
    for key, value in list(query_form_dict.items()):
        if hasattr(Master, key):
            if wfc_query:
                wfc_query = _apply_query_filter(Master, key, value, wfc_query)
            if hrc_query:
                hrc_query = _apply_query_filter(Master, key, value, hrc_query)
            if sbc_query:
                sbc_query = _apply_query_filter(Master, key, value, sbc_query)
        if wfc_query and hasattr(WFC_raw_0, key) and not hasattr(Master, key):
            wfc_query = _apply_query_filter(WFC_raw_0, key, value, wfc_query)
        if hrc_query and hasattr(HRC_raw_0, key) and not hasattr(Master, key):
            hrc_query = _apply_query_filter(HRC_raw_0, key, value, hrc_query)
        if sbc_query and hasattr(SBC_raw_0, key) and not hasattr(Master, key):
            sbc_query = _apply_query_filter(SBC_raw_0, key, value, sbc_query)

    # Combine the results
    query = _merge_query(wfc_query, hrc_query, sbc_query)

    # Perform the query
    if query:
        query_results = query.all()
        num_results = len(query_results)
    else:
        query_results = False
        num_results = 0
```

```
308
309     # Put results in a dictinoary
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 conenction to the
321     ``acsql`` database.
322
323     Returns
324     -------
325     session : obj
326         A ``SQLAlchemy`` ``session`` object which serves as a
327         connection to the ``acsql`` 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:
```

```
        query = sbc_query

    else:
        query = None

    return query
```

acsql/website/templates/404.html

```
{% extends "base.html" %}
{% block content %}

<h3>Oops! The page you're looking for doesn't seem to exist.</h3>

<center><img src="/../static/img/404.gif" alt="Technical Difficulties"></center>

{% endblock %}
```

acsql/website/templates/500.html

```
{% extends "base.html" %}
{% block content %}

<h3>Oops! Something went wrong.</h3>

<pre>{{trace_html|safe}}</pre>

{% endblock %}
```

acsql/website/templates/_formhelpers.html

```
{% macro render_field(field) %}
{% if field.type == 'MultiCheckboxField' %}
<div class="span12">
  <h5>{{ field.label.text }}</h5>
  {{ field(id=field.name, **kwargs)|safe }}
</div>
{% if field.label.text.endswith('s') %}
<div class="span12">
      <button type="button" id="check_all_{{field.name}}" class="btn">Select all {{field.label.text.lower()}}</button>
      <button type="button" id="uncheck_all_{{field.name}}" class="btn">Deselect all {{field.label.text.lower()}}</button>
{% else %}
<div class="span12">
      <button type="button" id="check_all_{{field.name}}" class="btn">Select all {{field.label.text.lower()}}s</button>
      <button type="button" id="uncheck_all_{{field.name}}" class="btn">Deselect all {{field.label.text.lower()}}s</button>
{% endif %}

{% elif field.type == 'SelectMultipleField' %}
<div class="{{ field.description[-5:] }} formfield">
  <h5>{{ field.label.text }}</h5>
  {{ field(style="width:100%;", placeholder=field.description[:-6], **kwargs)|safe }}

{% else %}
<div class="{{ field.description[-5:] }} formfield">
  <h5>{{ field.label.text }}</h5>
  {{ field(style="width:calc(100% - 1em);width:-moz-calc(100% - 1em);
   width:-webkit-calc(100% - 1em);width:-o-calc(100% - 1em);width:-ms-calc(100% - 1em);",
   placeholder=field.description[:-6], **kwargs)|safe }}
{% endif %}
</div>
{% endmacro %}

{% macro render_formfield(field) %}
<div class="{{ field.description[-5:] }} formfield">
  <h5>{{ field.label.text }}</h5>
  {% for subfield in field %}
    {% if subfield.name.endswith('op') %}
      {{ subfield(style="width:4em;display:inline;", **kwargs)|safe }} 
    {% elif subfield.name.endswith('val1') %}
      {{ subfield(style="width:calc(100% - 5.75em);display:inline;", placeholder=subfield.description, **kwargs)|safe }}
    {% elif subfield.name.endswith('val2') %}
      {{ subfield(style="width:calc(100% - 5.75em);display:inline;float:right;margin-right:3px;display:none;", placeholder=subfield.description, **kwargs)|safe }}
    {% endif %}
```

```
  {% endfor %}
</div>
{% endmacro %}
```

acsql/website/templates/archive.html

```
{% extends "base.html" %}
{% block content %}

<div class="row">
    <div class="span12">
        <h3>ACS Archive</h3>
        <table class="table">
            {% for row in proposal_array %}
            <tr>
                {% for proposal in row %}
                <td><a href="/archive/{{proposal}}/" target="_blank">{{proposal}}</a></td>
                {% endfor %}
            </tr>
            {% endfor %}
        </table>
    </div>
</div>

{% endblock %}
```

acsql/website/templates/base.html

```
<!DOCTYPE html>
<html class="no-js">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <title>ACS Quicklook</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width">
        <link rel="stylesheet" href="/static/css/bootstrap.min.css">
        <link rel="icon" type="image/png" href="/static/img/favicon.png">

        <style>
        body {
            padding-top: 60px;
            padding-bottom: 40px;
        }
        @media (min-width:980px) and (max-width:1199px) {
            .no-overflow {
                width:11.5vw;
                overflow:hidden;
                white-space:nowrap;
                text-overflow:ellipsis;
            }
        }
        </style>

        <link rel="stylesheet" href="/static/css/bootstrap-responsive.min.css">

        <script src="/../static/js/modernizr-2.6.2.min.js"></script>
        <script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js"></script>
        <script>window.jQuery || document.write('<script src="/static/js/jquery-1.10.1.min.js"><\/script>')
    </script>

    </head>
    <body>
        <div class="navbar navbar-inverse navbar-fixed-top">
            <div class="navbar-inner">
                <div class="container">
                    <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </a>
                    <a class="brand pull-left no-overflow" href="/"><img src="/static/img/acs-logo.png"
    width="32" height="32">  ACS Quicklook</a>
                    <div class="nav-collapse collapse">
                        <ul class="nav pull-right">
                            <li><a href="/database/"><font size="4">Database</font></a></li>
                            <li><a href="/archive/"><font size="4">Archive</font></a></li>
                            <li><a href="https://github.com/spacetelescope/acsql" target="_blank"><font
    size="4">GitHub</font></a></li>
```

```
                            <li><a href="http://acsql.readthedocs.io" target="_blank"><font size="4">Documentation</font></a></li>
                        </ul>
                    </div>
                </div>
            </div>

            <div class="container">

                {% block content %}
                {% endblock %}

                <hr>

                <footer>
                    <p>Space Telescope Science Institute</p>
                </footer>

            </div>

            <script src="/static/js/bootstrap.min.js"></script>
            <script src="/static/js/plugins.js"></script>
            <script src="/static/js/main.js"></script>
        </body>
</html>
```

acsql/website/templates/database.html

```
{% extends "base.html" %}
{% block content %}

<link rel="stylesheet" href="/../static/css/database.css">

<div class="row">
    <div class="span12">
        <h3>ACS Database Query Form</h3>
        <p>This form enables users to query the acsql database of all ACS images.
           Only a subset of the full database is represented here; if the options available do not meet your needs,
           you may enter a hard-coded MySQL command below.
        </p>
    </div>

    {% from "_formhelpers.html" import render_field, render_formfield %}

    <form id="queryform" method=get action='results' target="_blank">

        {% for field in form %}
            {% if field.type == 'FormField' %}
                {{ render_formfield(field) }}
            {% else %}
                {{ render_field(field) }}
            {% endif %}
        {% endfor %}

        <div class="span12" style="margin-top:2em;">
            <center>
                <button type="submit" value="Submit" class="btn btn-lg btn-primary">Submit</button>   
                <button type="reset" value="Reset" class="btn btn-lg btn-danger">Reset</button>
            </center>
        </div>
    </form>
</div>

{% for field in form %}
    {% if field.type == 'MultiCheckboxField' %}
        <script>
            $('#check_all_{{field.name}}').click(function() {
                $('input[name={{field.name}}]').prop('checked', true);
            });
            $('#uncheck_all_{{field.name}}').click(function() {
                $('input[name={{field.name}}]').prop('checked', false);
            });
        </script>
    {% elif field.type == 'FormField' %}
        <script>
```

```
            $('#{{field.name}}-op').change(function(){
                if ($(this).val() == 'between') {
                    $('#{{field.name}}-val2').show();
                } else {
                    $('#{{field.name}}-val2').hide();
                }
            });
        </script>
    {% endif %}
{% endfor %}
{% endblock %}
```

acsql/website/templates/database_error.html

```
{% extends "base.html" %}
{% block content %}

<div class="row">
    <div class="span12">
        <h4>There was a problem with your request. Please try again.</h4>
    </div>
    <div class="span12">

        {% if msg %}
            {{ msg }}
        {% else %}
        <ul>
            {% for field, errors in form.errors.items() %}
                <li>
                {% if form[field].type == 'FormField' %}
                        {{ form[field].label.text }}:
                        {% for err in errors.values() %}
                            {{ ', '.join(err) }}
                        {% endfor %}
                {% else %}
                        {{ form[field].label.text }}: {{ ', '.join(errors) }}
                {% endif %}
                </li>
            {% endfor %}
        {% endif %}

    </div>

</div>

{% endblock %}
```

acsql/website/templates/database_table.html

```
{% extends "base.html" %}
{% block content %}

<link rel="stylesheet" href="//cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css">
<link rel="stylesheet" href="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.
    bootstrap.css">
<link rel="stylesheet" href="/../static/css/database.css">
<link rel="stylesheet" href="/../static/css/loader.css">

<style type="text/css">
div#content { display: none; }
div#loading { text-align:center; z-index: 1000; }
</style>

<div class="row">

        <!-- Print how may results there are -->
        {% if results %}
        <div class="span12">
            {% if num_results == 0 %}
                <h4>The query returned no results.</h4>
            {% elif num_results == 1 %}
                <h4>The query returned 1 result.</h4>
            {% else %}
                <h4>The query returned {{ num_results }} results.</h4>
            {% endif %}

            <!-- If there are results to show -->
            {% if num_results > 0 %}

```

```
                <!-- Loading animation -->
                <div id="loading">
                    <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
                        <div class="cssload-cssload-loader-line-wrap-wrap">
                            <div class="cssload-loader-line-wrap"></div>
                        </div>
                        <div class="cssload-cssload-loader-line-wrap-wrap">
                            <div class="cssload-loader-line-wrap"></div>
                        </div>
                        <div class="cssload-cssload-loader-line-wrap-wrap">
                            <div class="cssload-loader-line-wrap"></div>
                        </div>
                        <div class="cssload-cssload-loader-line-wrap-wrap">
                            <div class="cssload-loader-line-wrap"></div>
                        </div>
                        <div class="cssload-cssload-loader-line-wrap-wrap">
                            <div class="cssload-loader-line-wrap"></div>
                        </div>
                    </div>
                    <br>
                    Loading table...
                </div>

                <!-- HTML table -->
                <div id="content">
                    <hr>
                    <table class="table hover" id="results">
                        <thead>
                            {% for col in output_columns %}
                                <th>{{ col }}</th>
                            {% endfor %}
                        </thead>
                        <tbody>
                        {% for line in results %}
                          <tr>
                            {% for value in line %}
                              <td>{{ value }}</td>
                            {% endfor %}
                          </tr>
                        {% endfor %}
                        </tbody>
                    </table>
                </div>
            {% endif %}

        </div>
        {% endif %}

</div>

<script src="//cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js"></script>
<script src="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.bootstrap.js"></script
    >

<script>
$(document).ready(function(){
    $('#results').DataTable( {
        "lengthMenu": [[10, 25, 50, 100, 500, 1000, -1], [10, 25, 50, 100, 500, 1000, "All"]],
        stateSave: true,
        "language": {
            "lengthMenu": "Display _MENU_ results per page",
            "zeroRecords": "Nothing found",
            "info": "Showing page _PAGE_ of _PAGES_",
            "infoEmpty": "No results available",
            "infoFiltered": "(filtered from _MAX_ total results)"
        }
    } );
});

function preloader(){
    document.getElementById("loading").style.display = "none";
    document.getElementById("content").style.display = "block";
}
window.onload = preloader;

</script>

```

```
{% endblock %}
```

acsql/website/templates/header.html

```
{% extends "base.html" %}
{% block content %}

<div class="row">
    <div class="span12">
        <h3>{{header_dict.filename}} {{header_dict.fits_type}} header</h3>
        {% if header_dict.header %}
            {% for ext, head in header_dict.header.items() if ext.startswith('Extension') %}
                <h4>{{ ext }}</h4>
                <p>{{ head|safe }}</p>
            {% endfor %}
        {% else %}
            <p>There is no header to display.</p>
        {% endif %}
    </div>
</div>

{% endblock %}
```

acsql/website/templates/index.html

```
{% extends "base.html" %}
{% block content %}

<link rel="stylesheet" href="//cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css">
<link rel="stylesheet" href="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.
    bootstrap.css">

<div class="row">
    <div class="span6">
        <h2>Database</h2>
        <p>Query the acsql Database</p>
        <p><a class="btn" href="/database/">Query form &raquo;</a></p>
    </div>
    <div class="span6">
        <h2>Archive</h2>
        <p>Links to public ACS archival data.</p>
        <p><a class="btn" href="/archive/">Archive &raquo;</a> </p>
    </div>
</div>

<script src="//cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js"></script>
<script src="//cdn.datatables.net/plug-ins/1.10.7/integration/bootstrap/2/dataTables.bootstrap.js"></script
    >

<script>
$(document).ready(function(){
    $('#scriptstatus').DataTable( {
        "paging":   false,
        "bFilter": false,
        stateSave: true
    } );
});
</script>
{% endblock%}
```

acsql/website/templates/view_image.html

```
{% extends "base.html" %}
{% block content %}

<!-- Image info -->
<div class="row">

    <!-- Filename title -->
    <div class="span12">
        <h3>{{image_dict.filename}}</h3>
    </div>

    <!-- Image metadata -->
    <div class="span6">
        <p>
        <b>Proposal {{image_dict.proposal_id}}</a>:</b> <i> {{image_dict.proposal_title}}</i><br>
        Image {{image_dict.page}} of {{image_dict.num_images}}<br>

```

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

        <!-- Additional information links -->
        <br>
        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>
        {% if image_dict.available_jpegs %}
            View JPEG for
            {% for key, link in image_dict.available_jpegs.items() %}
                <a href="/archive/{{image_dict.proposal_id}}/{{image_dict.filename}}/{{key}}"> {{key}}</a>
            {% endfor %}
        {% endif %}
        </p>
    </div>
    <div class="span3">
        <p>
        <b>Expstart:</b> {{image_dict.expstart}}<br>
        <b>Filter:</b> {{image_dict.filter1}}/{{image_dict.filter2}}<br>
        <b>Aperture:</b> {{image_dict.aperture}}<br>
        <b>Exposure time:</b> {{image_dict.exptime}}<br>
        </p>
    </div>
    <div class="span3">
        <p>
        <b>Expflag:</b> {{image_dict.expflag}}, Quality: {{image_dict.quality}}<br>
        <b>Coordinates:</b> {{image_dict.ra}}, {{image_dict.dec}}<br>
        <b>Target:</b> {{image_dict.targname}}<br>
        <b>PI:</b> {{image_dict.pi_first_name}} {{image_dict.pi_last_name}}<br>
        </p>
    </div>
</div>

<!-- Other available JPEGs -->


<!-- Image -->
<div class="row">
    <div class="fleximage">
        {% if image_dict.image %}
            <div id="wrapper" style="width:100%; text-align:center">
                <img class="img-responsive" src={{image_dict.image}} alt={{image_dict.image}} align:center
    height="75%" width="75%">
            </div>
        {% else %}
            <br><br><br>
            <center><strong><p>There is no JPEG to display for FITS type {{image_dict.fits_type}}</p></
    strong></center>
        {% endif %}
    </div>
</div>

<div style="height:1em;"></div>

<!-- Enable left and right keystrokes -->
<script type="text/javascript">
    {% if not image_dict.last %}
    $("body").keydown(function(e) {
      if(e.keyCode == 39) { // right
        window.location = "{{ url_for('view_image',**image_dict.next) }}";
      }
    });
    {% endif %}
    {% if not image_dict.first %}
    $("body").keydown(function(e) {
      if(e.keyCode == 37) { // left
        window.location = "{{ url_for('view_image',**image_dict.prev) }}";
      }
    });
    {% endif %}
```

```
</script>

{% endblock %}
```

acsql/website/templates/view_proposal.html

```
{% extends "base.html" %}
{% block content %}

<link rel="stylesheet" href="/../static/css/loader.css">
<link rel="stylesheet" href="/../static/css/view_proposal.css">

<div class="row">
    <div class="span12">

    <!-- Header -->
    {% if proposal_dict.num_images %}
        <h3>Proposal {{proposal_dict.proposal_id}} {% if proposal_dict.proposal_title %}<small>{{proposal_dict.proposal_title}}{% endif %}</small></h3>
        {% if proposal_dict.buttons %}
        <form id="filtering" style="margin-bottom:0;">

            <!-- Filter -->
            <div id="filterOptions" class="container">
                <div class="form-group">
                    {% if proposal_dict.cycle %}Cycle {{proposal_dict.cycle}}, {{proposal_dict.schedule}}<br>{% endif %}
                    {{proposal_dict.num_images}} images{% if proposal_dict.num_visits %}, {{proposal_dict.num_visits}} visits{% endif %}
                    <br>
                    <a href={{proposal_dict.status_page}} target="_blank">Proposal info</a>
                </div>
                {% for key,buttonlist in proposal_dict.buttons.items() %}
                    <div class="form-group">
                        <label for="{{key}}">Show only {{key}}:</label>
                        <select class="form-control" id="{{key}}">
                            <option value="all">All {{key}}s</option>
                            {% for b in buttonlist %}
                                <option value="{{b}}">{{b}}</option>
                            {% endfor %}
                        </select>
                    </div>
                {% endfor %}
            </div>

            <!-- Sort By -->
            <div class="row" id="sortOptions">
                <div class="span6">
                    Sort by:  
                    <label class="radio inline"><input type="radio" name="sort" id="unsort" checked>default</label>
                    <label class="radio inline"><input type="radio" name="sort" id="sort-expstart">expstart</label>
                    <label class="radio inline"><input type="radio" name="sort" id="sort-exptime">exptime</label>
                </div>
                <div class="span5 text-right">
                {% if proposal_dict.buttons|length > 2 %}
                    Filter logic:  
                    <label class="radio inline"><input type="radio" name="logic" value="and" checked>"AND"</label>
                    <label class="radio inline"><input type="radio" name="logic" value="or">"OR"</label>
                {% endif %}
                </div>
                <div class="span1 text-right">
                    <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
                </div>
            </div>
        </form>
        {% endif %}

            <!-- Loading animation -->
            <div id="loading">
                <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
```

```
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                </div>
                <br>
                Loading thumbnails...
            </div>

            <!-- Thumbnails -->
            <div id="onload">
                <div id="thumbnail-array">
                    {% for i in range(proposal_dict.num_images) %}
                        <div class="thumb" page="{{i+1}}" detector="{{proposal_dict.detectors[i]}}" filter1
    ="{{proposal_dict.filter1s[i]}}" filter2="{{proposal_dict.filter2s[i]}}" expstart="{{proposal_dict.
    expstarts[i]}}"  exptime="{{proposal_dict.exptimes[i]}}" targname="{{proposal_dict.targnames[i]}}"
    proposid="{{proposal_dict.proposal_id}}" visit="{{proposal_dict.visits[i]}}" style="background:url
    (/../{{proposal_dict.thumbs[i]}}) no-repeat;">
                            <div class="overlay">
                                <strong>{{proposal_dict.filenames[i]}}</strong><br><font size="0.5">
                                <b>Detector:</b> {{proposal_dict.detectors[i]}}<br>
                                <b>Filter:</b> {{proposal_dict.filter1s[i]}} / {{proposal_dict.filter2s[i
    ]}}<br>
                                <b>Exptime:</b> {{proposal_dict.exptimes[i]}}<br>
                                <b>Target:</b> {{proposal_dict.targnames[i]}}<br></font>
                            </div>
                            <a href="{{proposal_dict.viewlinks[i]}}" target="_blank">
                                <img src="/../{{proposal_dict.thumbs[i]}}", style="opacity:0;">
                            </a>
                        </div>
                    {% endfor %}
                </div>
            </div>
    {% else %}
        <p>
            <h5>There are no images to display.</h5>
        </p>
    {% endif %}
    </div>
</div>

<script type="text/javascript">
    function preloader(){
        document.getElementById("loading").style.display = "none";
        document.getElementById("onload").style.display = "block";
    }
    window.onload = preloader;
</script>

<script src="/../static/js/tinysort.min.js"></script>
<script src="/../static/js/thumbnails.js"></script>

{% endblock %}
```

acsql/website/templates/view_query_results.html

```
{% extends "base.html" %}
{% block content %}

<link rel="stylesheet" href="/../static/css/loader.css">
<link rel="stylesheet" href="/../static/css/view_proposal.css">

<div class="row">
    <div class="span12">

    <!-- Header -->
    {% if thumbnail_dict.num_images %}
        <h3>The query returned {{thumbnail_dict.num_images}} results</h3>
        {% if thumbnail_dict.buttons %}
        <form id="filtering" style="margin-bottom:0;">

```

```
            <!-- Filter -->
            <div id="filterOptions" class="container">
                {% for key,buttonlist in thumbnail_dict.buttons.items() %}
                    <div class="form-group">
                        <label for="{{key}}">Show only {{key}}:</label>
                        <select class="form-control" id="{{key}}">
                            <option value="all">All {{key}}s</option>
                            {% for b in buttonlist %}
                                <option value="{{b}}">{{b}}</option>
                            {% endfor %}
                        </select>
                    </div>
                {% endfor %}
            </div>

            <!-- Sort By -->
            <div class="row" id="sortOptions">
                <div class="span6">
                    Sort by:  
                    <label class="radio inline"><input type="radio" name="sort" id="unsort" checked>default
    </label>
                    <label class="radio inline"><input type="radio" name="sort" id="sort-expstart">expstart
    </label>
                    <label class="radio inline"><input type="radio" name="sort" id="sort-exptime">exptime</
    label>
                </div>
                <div class="span5 text-right">
                {% if thumbnail_dict.buttons|length > 2 %}
                    Filter logic:  
                    <label class="radio inline"><input type="radio" name="logic" value="and" checked>"AND"<
    /label>
                    <label class="radio inline"><input type="radio" name="logic" value="or">"OR"</label>
                {% endif %}
                </div>
                <div class="span1 text-right">
                    <button id="clear" class="btn btn-danger" type="reset" value="Reset">Clear</button>
                </div>
            </div>
        </form>
        {% endif %}

            <!-- Loading animation -->
            <div id="loading">
                <div class="cssload-loader-inner" style="height:100px;margin:0 auto;">
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                    <div class="cssload-cssload-loader-line-wrap-wrap">
                        <div class="cssload-loader-line-wrap"></div>
                    </div>
                </div>
                <br>
                Loading thumbnails...
            </div>

            <!-- Thumbnails -->
            <div id="onload">
                <div id="thumbnail-array">
                    {% for i in range(thumbnail_dict.num_images) %}
                        <div class="thumb" page="{{i+1}}" detector="{{thumbnail_dict.detectors[i]}}"
    filter1="{{thumbnail_dict.filter1s[i]}}" filter2="{{thumbnail_dict.filter2s[i]}}" expstart="{{
    thumbnail_dict.expstarts[i]}}"  exptime="{{thumbnail_dict.exptimes[i]}}" targname="{{thumbnail_dict.
    targnames[i]}}" proposid="{{thumbnail_dict.proposal_ids[i]}}" visit="{{thumbnail_dict.visits[i]}}"
    style="background:url(/../{{thumbnail_dict.thumbs[i]}}) no-repeat;">
                            <div class="overlay">
                                <strong>{{thumbnail_dict.filenames[i]}}</strong><br><font size="0.5">
                                <b>Detector:</b> {{thumbnail_dict.detectors[i]}}<br>
                                <b>Filter:</b> {{thumbnail_dict.filter1s[i]}} / {{thumbnail_dict.filter2s[i
```

```
    ]}}<br>
                                    <b>Exptime:</b> {{thumbnail_dict.exptimes[i]}}<br>
                                    <b>Target:</b> {{thumbnail_dict.targnames[i]}}<br></font>
                                </div>
                                <a href="{{thumbnail_dict.viewlinks[i]}}" target="_blank">
                                    <img src="/../{{thumbnail_dict.thumbs[i]}}", style="opacity:0;">
                                </a>
                            </div>
                    {% endfor %}
                </div>
            </div>
    {% else %}
        <p>
            <h5>There are no images to display.</h5>
        </p>
    {% endif %}
    </div>
</div>

<script type="text/javascript">
    function preloader(){
        document.getElementById("loading").style.display = "none";
        document.getElementById("onload").style.display = "block";
    }
    window.onload = preloader;
</script>

<script src="/../static/js/tinysort.min.js"></script>
<script src="/../static/js/thumbnails.js"></script>

{% endblock %}
```