The Hubble Space Telescope Advanced Camera for Surveys Quicklook Project

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

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

Abstract—The Hubble Space Telescope (HST) Advanced Camera for Surveys (ACS) instrument has been acquiring thousands of astronomical images each year since its installation in 2002 and subsequent restoration in 2009. The ACS Quicklook Project (`acsql`) provides a means for users to discover and interact with these data via a database-driven web application. This is accomplished via several `acsql` 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, allowing users to build custom datasets based on obseravational parameters, (3) A `Python`/`Flask`-based web application, which allows users to view "Quicklook" JPEG images of any publicly-available ACS data along with its metadata, and (4) a `Python` code library, which provides a platform on which users can build automated instrument calibration and monitoring routines. The `acsql` project may be extended to support the forthcoming James Webb Space Telescope (JWST) mission, which is scheduled to launch in 2018.

1 Introduction

The Advanced Camera for Surveys (ACS) is a third-generation imaging instrument on board the Hubble Space Telescope (HST), installed in 2002 during Servicing Mission 3B. 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), desingned for far-ultraviolet imaging and spectroscopy. ACS expererienced an electronics failure in 2007 that affected the WFC and HRC detectors, until 2009 when astronauts succesfully restored the WFC detector during Servicing Mission 4; the HRC still remains unoperational.

Besides these few hiccups, the ACS instrument has been steadily acquiring astronomical images over its 15 on-orbit lifetime. Figure 1 shows an estimates of the number of observations over time for each of the three detectors. To date, there have been nearly 200,000 of observations total. Further information about the ACS instrument including its history, configuration, performance, and scientific capability can be found in the ACS Instrument Handbook (Avila et al., 2017).

ACS data, along with all other data from the other HST instruments past and present (e.g. The Wide Field Camera 3 (WFC3), The Cosmic Origins Spectrography (COS), etc.), are primarily stored and publicly-available in the Barbara A. Mikulski Archive for Space Telescopes (MAST)[1] (Barbara, 2017). Through MAST, users can request and retreive data for any publicly-available dataset via `ftp`, `sftp`, or DVD by mail[2]. The ACS data, like most all other astronomical data, are stored in the Flexible Image Transport System (FITS) filetype (FITS, 2008). This filetype has several unique characteristics, as will be discuessed in section 1.1.

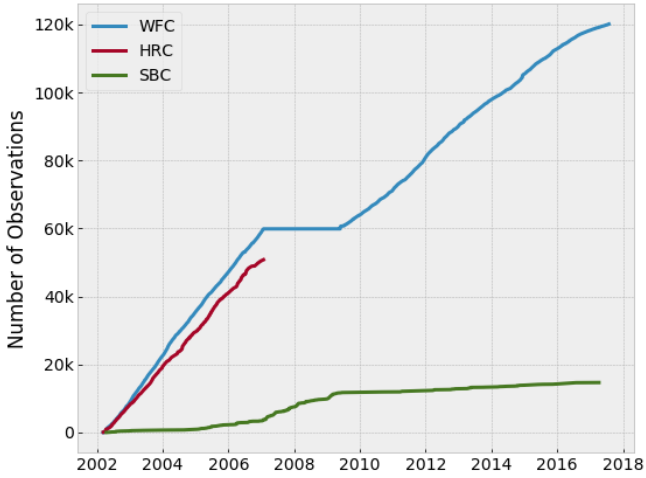

Fig. 1: The number of observations over time for each of the three detectors on ACS.

The ACS Quicklook Project is a `python`-based application for discovering, viewing, and querying all publicly-

Manuscript received Month DD, YYYY

1. named after the U.S. Senator from Maryland who has been a pivitol 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.

available ACS data. It consists of several subsystems: (1) A filesystem that stores ACS instrument data files and "Quicklook" JPEGs in an organized Network File System (NFS), (2) A `MySQL` database that stores image metadata of each observation, (3) A `python/Flask`-based web application for interacting with the filesystem and database, and (4) A `python` code library (named `acsql`) that contains code for connecting to the database, ingesting new data, logging production code execution, and building/maintaining the database and web application. Each of these subsystems are explained in further detail in the Methodology section of this paper.

This paper aims to outline and detail the ACS Quicklook project as part of the Towson University Computer Science Masters Program Graduate Project. The remaining subsections in this chapter discuss the motivation and use cases for this application, as well as details on the underlying data structure on top of which this project was built. Chapter 2 discusses related work to this project and how the ACS Quicklook project differs from existing similar applications. Chapter 3 details the implementations of each of the ACS Quicklook subsystems. Chapter 4 outlines the results of the project, namely the project deliverables. Lastly, chapters 5 and 6 conclude the paper with a discussion of possible extensions and modifications to the application.

It should be noted 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).

1.1 Data Structure

The design of the ACS Quicklook application, especially the database, is heavily dependant on the underlying data structure of ACS FITS files. As such, it is important for the reader to understand this data structure and thus the next four sections are dedicated to giving an overview on the subject.

1.1.1 Filenames

Each ACS data file is named in a consistent fashion:

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

where each `<rootname>` consists of nine unique alphanumeric characters, and `<filetype>` is one of several three-character filetype options (discussed in proceeding section 1.1.4). For example, one ACS observation has the rootname `j6mf16lhq_raw.fits` (Principle Investigator Gary Bernstein, observation date `2016-09-22`). Each character in the 9-character `rootname` has meaning, and is discussed in section 5.2 of the Introduction to the HST Data Handbooks (Smith et al., 2011). The `.fits` extension at the end of the filename signifies that the file is of FITS format.

Note about rootname caveat.

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

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

1.1.2 FITS file structure

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

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

Tables 1-3 describe the different extensions of ACS FITS files for each of the three ACS detectors. Note that there are two sets of `SCI/ERR/DQ` extensions for WFC since WFC is comprised of two separate CCD chips.

Over the years, there have been several tools written in various programming languages to read in FITS files and automatically convert their extension data to multi-dimensional array data types and their extension headers to dictionary data types. For this project, the `astropy.fits` `python` library is used extensivly to read and interact with ACS FITS files (Robitaille et al., 2013).

```
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: An example header.

1.1.3 FITS file extension headers

As mentioned in the previous section, each FITS extension contains a "header", which contains key/value pairs of metadata associated with the extension data. Such metadata includes various data that describes the astronomical observation (e.g. target name, exposure time, principle investigator name, etc.), telemetry of ACS or HST in general at the time of observation (e.g. temperature of the ACS instrument, orientation of the telescope pointing, position of the telescope relative to Earth, etc.) or the FITS file itself (e.g. the number of extensions, file creation date, etc.). A subsection of an example header is shown in Figure 2.

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

1.1.4 FITS filetypes for ACS

As discusses in section 1.1.1, each ACS observation may result in several FITS filetypes. Each filetype describes the observation in a different way. The set of available filetypes for a given observation is dependent on the characteristics of the observation, the details of which are beyond the scope of this paper. Also beyond the scope of this paper are the vast details that surround each filetype; each one has a different scientific application that is not important to understanding the ACS Quicklook project. However, to provide at least some context, below we give a brief description of each possible filetype that a given observation may contain:

- **raw** - The raw, uncalibrated data that comes directly from HST
- **flt** - nominally calibrated data
- **flc** - nominally calibrated data plus corrected for Charge Transfer Efficieny (CTE) deficits.
- **drz** - Geometric distortion-corrected data
- **drc** - Geometric distrotion-corrected plus CTE corrected data
- **spt** - Telescope telemetry data
- **jit** - Telescope pointing data
- **jif** - Telescope drifting data
- **crj** - Cosmic ray rejected data
- **crc** - Cosmic ray rejected plus CTE corrected data
- **asn** - Observation association table.

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

1.2 Key Metadata

There are several metadata key/value pairs that are particulary important for the ACS Quicklook application, specifically the web application. For some reference, and context, these metadata are briefly described below. Note that the `rootname` and `proposal_type` are not metadata from extension headers, but rather are metadata that were explicitly added to the database schema.

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

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

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

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

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

EXPSTART - The exposure start time of the observation, in units of Modified Julian Date (e.g. `52473.8`).

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

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

FILTER2 - The selected element from the `ACS` filter wheel # 1 (e.g. `F814W`).

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

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

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

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

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

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

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

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

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

1.3 Motivation

The motivation for the `acsql` system is driven by several limitations of the FITS file structure as well as the current capabilities of MAST from specific user perspectives (inteded users and their use cases are discussed in section 1.2). Some of these limitations are described below, along with the intended way the ACS Quicklook application aims to address them.

1.3.1 Data retreival letency

Currently, users who wish to retreive data from the MAST must submit a retreival request via the MAST online interface. Once the retreival request is processed (usually automatically unless it is a request of a large number of datasets), the data are either transfered to the user directly via `sftp`, transfered to a "staging area" in which the user can log into and copy the data via `ftp` at their leisure, or sent by mail via DVD, depending on which option the user selects. In the case of any one of these options, the time between a download request and the the time in which the user has fully retreived the data may be a significant amount of time. In the fastest scenario of the `sftp` option, a typical request can take minutes to hours to be completed. Furthermore, there are limited options available for programmatically obtaining new data; users who wish to retreive the latest availble data must (1) discover which datasets are of interest via a query to the MAST database, (2) construct a download request, (3) submit the download request, and (4) await the data.

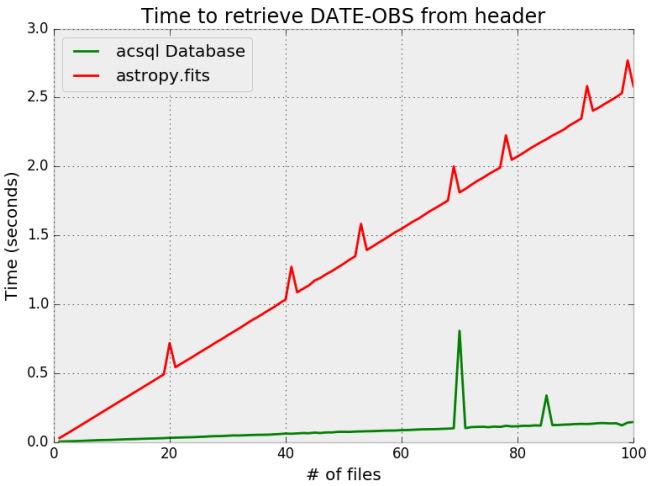

Fig. 3: The amount of time it takes to retreive the `DATE-OBS` header keyword from a varying number of ACS data files via reading the FITS file using the `astropy.io.fits` module (red) and querying the `acsql` database (green)

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

1.4 File I/O

1.5 Data redundancy

1.6 Data discovery

1.7 Use Cases

The intended user of ACS Quicklook are ACS instrument scientists, analysts, or scientific users who wish to perform one or more of the following use cases:

1. Visually inspect an image from the ACS archive 2. Determine which datasets exist in the ACS archive for a given set of observational parameters 3. Determine the observational parameters for a given dataset 4. Programmatically analyze images across custom datasets

2 Related Work

Topics to discuss:

1. The MAST archive 2. The MAST portal 3. The WFC3/Quicklook project 4. Other Astronomy Institutions 5. How ACS/Quicklook is different

3 Methodology and Implementation

In this chapter, we disucss the methods by which we implemented the various subsystems of the ACS Quicklook system. Additionally, we discuss the programming standards and standard workflows that were employed to

promote code quality such as readability, maintainability, extensibility, etc; we believe that this aspect of the project is at least equaly important to the system as its individual components.

3.1 Version control

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

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

Additionally, using `GitHub` allowed for issue tracking of bugs, features, and potential enhancements to the code repository. Current open issues of the repository can be found at https://github.com/spacetelescope/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 (van Rossum, 2001), (2) The `PEP257` `python` guide for module and function docstring conventions (Goodger, 2001), and (3) the `numpydocs` documentation standard (NumPy Documentation, 2017). More details on each of these standards and conventions are given below.

The `PEP8` Style Guide for `python` code (abbreviated for 'Python Enhancement Proposal #8') documents `python` coding conventions including variable naming, spacing, line length, module layout, function layout, comments, and design patterns. Only in specific cases were these conventions not followed, such as using a single line of code, even if it exceeded the recoomended 80 characters, to allow for greater readability. By following these conventions, the style of the `acsql` code is constistent amongst each module and attempts to reflect the style of industry-grade `python` code.

The `PEP257` guide for docstring conventions describes standard conventions used for function and module docstrings (i.e. the API documentation found in block comments at the beginning of modules or immediately after function declarations). Like `PEP8`, following these conventions ensure consistency amongst the `acsql` code documentation. Furthermore, the `numpydocs` documentation convention provides some additional details on top of the

```python
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. 4: An example of the `PEP257` and `numpydoc` docstring conventions, using the `get_proposal_type` function from `acsql.ingest.ingest`.

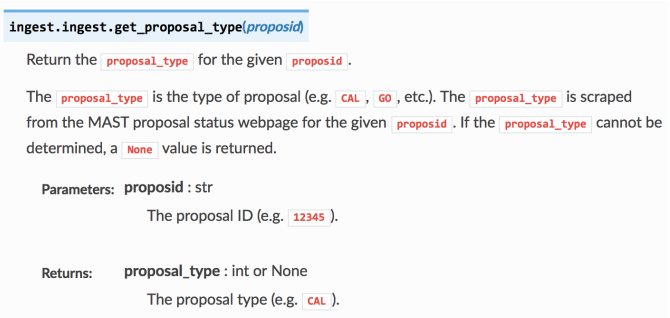

Fig. 5: The `readthedocs` documentation for the `acsql` example function seen in Figure N.

`PEP257` conventions and is used in many `python` packages including the `numpy` (numerical `python`) and `scipy` (scientific `python`) packages (van der Walt et al., 2011). Figure N shows an example of these conventions, taken from the `ascql.ingest.ingest.get_proposal_type` function.

Another benefit to using `PEP257` and `numpydoc` docstring conventions is that API documentation creation tools such as `sphinx` (Brandi et al., 2007) or `epydoc` (Loper, 2004) can automatically convert the docs into other output formats such as `HTML` and `PDF`. For this project, we use `sphinx` to convert API documentation to `HTML`, and host the webpages online using the `readthedocs`, which is an open-source, community supported tool for hosting and browing documentation (Read the Docs, 2017). The documentation for `acsql` is hosted at http://acsql.readthedocs.io/. The output documentation as seen on `readthedocs` for the example function in figure N is provided in Figure N.

3.3 Filesystem: Archive of ACS data

The `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 N shows an example of this directory structure: The parent directory is the first four characters of the 9-character `rootname`,

```
filesystem/
    jcp0/
        jcp001kwq/
            jcp001kwq_flc.fits
            jcp001kwq_flt.fits
            jcp001kwq_raw.fits
            jcp001kwq_spt.fits
            jcp001kwj_jit.fits
            jcp001kwj_jif.fits
        jcp001010/
            jcp001010_asn.fits
            jcp001010_drc.fits
            jcp001010_drz.fits
            jcp001010_jif.fits
            jcp001010_jit.fits
        jcp014tyq/
            ...
        jcp001ktq/
            ...
        ...
    jcp3/
        ...
    jcp7/
        ...
    ...
```

Fig. 6: A representation of the directory structure within the `acsql` filesystem, using a few observations as an example.

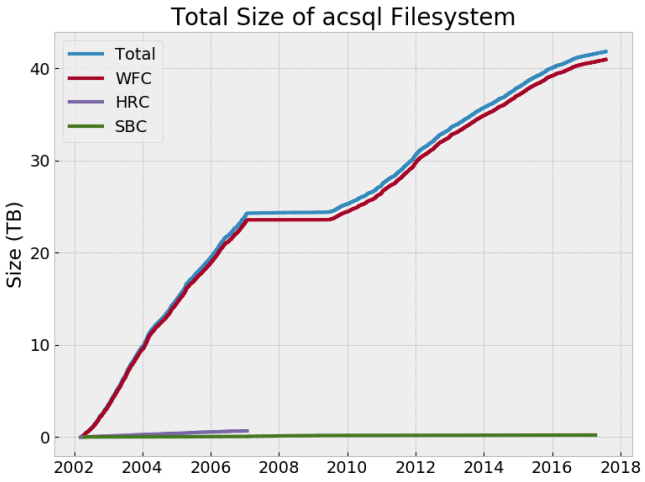

Fig. 7: The size of the `acsql` filesystem as a function of observation date.

which maps directly to an individual `PROPOSID`. The subdirectories of the parent directories are named after the full 9-character `rootname` such that each parent directory contains the `rootname` subdirectories that were observed for that particular `PROPOSID`. Each `rootname` subdirectory contains every available filetype (as described in Section 1.1.4) for the particular observation is stored.

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

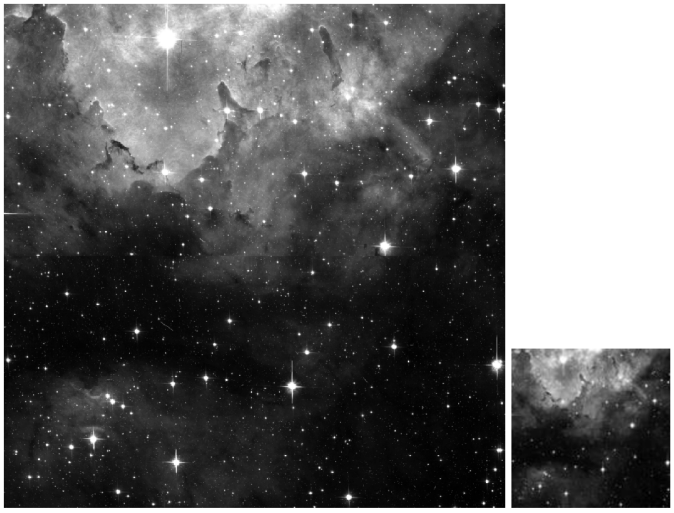

Fig. 8: An example of a `JPEG` image (left) and its corresponding `thumbnail` image using example dataset `jcs718koq`.

3.4 Filesystem: Archive of JPEGs and Thumbnails

In addition to the ACS data products described in the last section, the `acsql` filesystem also stores "Quicklook" `JPEG` and `thumbnail` images of each `RAW`, `FLT`, and `FLC` filetype (when applicable) in an organized directory structure. These images are used by the `acsql` web application to allow users to quickly and easily view the contents of the data without having to pysically open the corresponding `.fits` files.

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

Unlike the ACS data products portion of the filesystem, the `JPEG` and `thumbnail` portions of the filesystem are organized based on the 5-digit `PROPOSID` of the corresponding observation instead of the first four characters of the `rootname`. This design was chosen as a means to simplify the design of the web application; users often which to view data based on the 5-digit `PROPOSID` and less often on the details of the `rootname`. An example of this sturcture is shown in Figure N. Note that the `thumbail` filesystem only contains thumbnails created from `FLT` filetypes, since thumbnails are only inteded for navigation and quick-viewing.

3.5 Database: Relational Schema

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

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

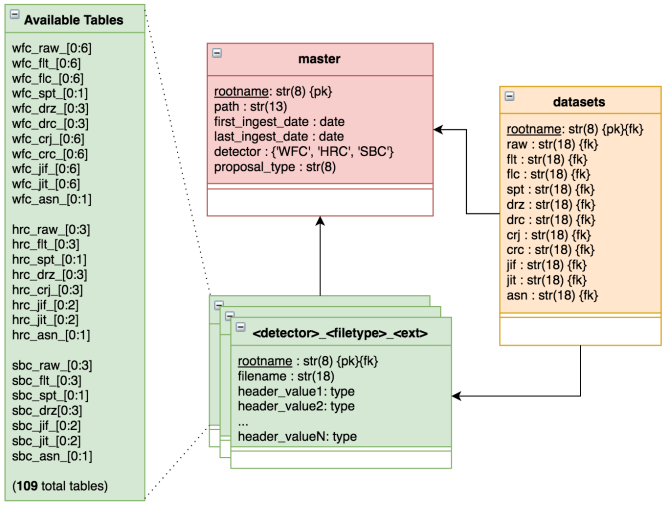

Fig. 10: The relational database schema for the `acsql` database.

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

The `master` table contains information that is particularly useful for maintaining and using the `acsql` database. Its primary key is the first 8 characters of the 9-character `rootname` for the partiuclar 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 recenly 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 correspondance with the header metadata key/value pairs found in observations files; each column is named in the same manner as the header keys, with the value of that column reflecting the header value. There is one table for each `detector`, `filetype`, and `extension` combination; collectively, these are referred to as the 'header' tables. Like with the `datasets` table, the `rootname` column serves as a primary key for the header tables as well as a foreign key that maps to the `rootname` of the `master` table.

3.6 Database: `MySQL` + `SQLAlchemy`

The `acsql` database is stored on a `MySQL` server (`Version 5.6`) (Oracle, 2017) that is hosted at STScI. The database schema was implimented using `SQLAlchemy`, which is an open-source SQL toolkit and Object Relational Mapper (ORM) for `python` (Bayer, 2006). As an ORM, `SQLAlchemy` enables `python` classes to be easily translated to `SQL`-based database tables, and vice versa. Additionall, `SQLAlchemy` provides `python` methods for connecting to a SQL-based database and performing typical `SQL` tasks such as inserts, updates, and queries.

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

The `engine` object contains the Python Database API Specification (also known as `DBAPI`), which provides a low-level API for `python`-specific, commonly-used database tasks (Lemburg, 2017). It is created from the `sqlalchemy.create_engine` method, which requires a user-supplied `connection_string`. The `connection_string` is a string that contains information about the type of database, the specific database dialect being used, and the user credentials (e.g. username, password, port number, and host server name). In the case of the `acsql`, this connection string takes the form of `'mysql+pymysql://username:password@host:port/acsql'`.

```
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. 11: The `load_conenction` function, which is used to build a connection to the `acsql` database

.

The `connection_string` is imported from a user supplied `config` file within the `acsql` library (as will be discussed in section 3.9).

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

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

The `master` and `datasets` tables were implemented via explicit class definitions in `database_interface`, and are shown in Figures N and N, respectively. Each table column is defined using the `sqlalchemy.Column` object, which is a class that can be initialized with the datatype that will be stored in the column (e.g. `String`, `Float`, `Integer`, etc.) as well as parameters that set SQL-like constraints and parameters on the column values. These include, but are not limited to, primary keys (e.g. the `primary_key=True` parameter in the `master.rootname` column), foreign

```
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. 12: The class definition for constructing the `master` table via `SQLAlchemy`.

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

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

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

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.

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 N. Similar to the `Master` and `Datasets`

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

classes, some of the columns in the `orm_factory` function are explicitly defined via the `SQLAlchemy Column` class. However, the columns that correspond to header key/value pairs are defined in a separate function named `define_columns`, shown in Figure N.

The purpose of the `define_columns` function is to define `SQAlchemy 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`. An portion of an example text file is shown in Figure N.

Furthermore, the 109 text files used to define the header table columns are also generated in an automated fashion via the `acsql.database.make_tabledefs.py` module. This module uses a set of example FITS files to scrape its header contents, determine all of the header keywords and their datatypes, and write the results to a text file. Similarly, the `acsql.database.update_tabledefs.py` is used to add new header keywords by comparing the header contents of a given FITS file and the existing column definition text files[3].

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

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

3. New header keywords are occaisonally introduced to ACS data proceeding updates to its calibration software

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

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

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

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

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

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

    return data_dict
```

Fig. 15: The `define_columns` function, used to define columns used in the header tables.

3.7 Data ingestion software: Algorithm

Another critical component of the ACS Quicklook system are the modules that are used to ingest data into the `acsql` database and to create the "Quicklook" JPEGs and thumbails. By the term 'ingest', we refer to the following algorithm:

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

```
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. 16: 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. 17: An example of how the `orm_factory` function is called to create class defintions for the header tables.

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

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

```
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. 18: 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.

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

4. ***Loop over the available `filetypes` for the given `rootname`:*** The available `filetypes` are determined by traversing down a level in the tree structure of the filesystem and identifying which files are present. Once determined, the ingestion algorithm processes each `<rootname>_<filetype>.fits` file individually. Please

note that steps 5 through N are written from the perspective that a single file is being ingested (i.e. inside of the next nested loop).

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

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

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

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

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

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

3.8 Data ingestion software: `logging`

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

When the ingestion module gets executed, an empty log file is created with the filename `<module_name>_<timestamp>.log`, where `<module_name>` is the name of the ingestion module (in production, this is `ingest_production.py`, as will be discussed in section 3.9), and `<timestamp>` is the current time in the format `YYYY-MM-DD-HH-MM`. The naming convention of the log file allows system maintainers to determine which log file corresponds to which ingestion run.

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

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

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

3.9 Web Application

3.9.1 Overview

The front-end of the `acsql` system is the web application. The web application is built using `Python` and `Flask`, which is a `Python` based web framework (Ronacher, 2010). Currently, the web application has two main features/-modes of use: (1) viewing JPEGs and image metadata for any publicly-available ACS `raw`, `flt`, and `flc` image (when applicible), and (2) performing relational queries on the `acsql` database. To visiualize these features, we show examples of some of the different webpages that make up the web application in Figures N through N, and further describe each below.

Figure N shows the `acsql` homepage. The homepage, as well as all other webpages in the web application, contains a menu bar at the top containing four links: (1) to the database query page, (2) to the archive links page, (3) to the `acsql` code repository on GitHub, and (4) to the `readthedocs` documentation page. Clicking the "ACS Quicklook" button in the menu bar allows the user to return to the homepage, regadless 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.

```
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. 19: An example log file for the ingestion of a single file (`j8zh21xv`).

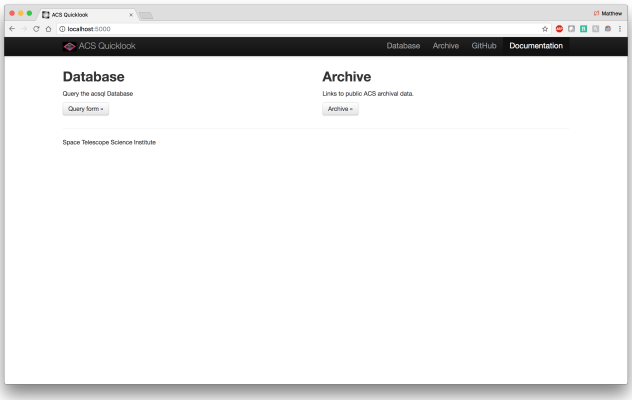

Fig. 20: The homepage of the `acsql` web application.

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

In this example, we use the form to find all data that were taken with the `WFC` detector, are of Observation Type `IMAGING`, and were observed by the Principle Investigator Zolt Levay[4]. Note that there are three output options: (1) an `HTML table`, which returns a webpage showing the selected 'Output Columns' (in this case, the Rootname, PI last/first name, Proposal ID, Aperture, Date/Time of observation, and the Target Name) (see Figure N), (2) a CSV file, which is downloaded to the user's machine, containing a comma-separated table of the selected output columns (see Figure N), or (3) a webpage showing the Thumbnail images

4. 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.

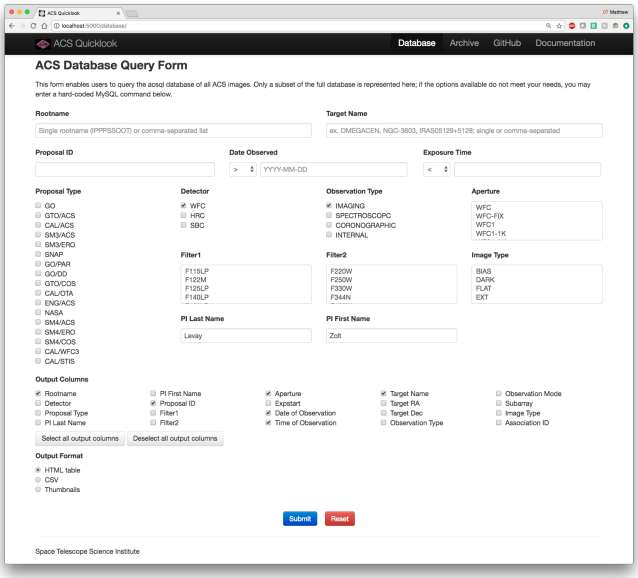

Fig. 21: A portion of the database query page of the `acsql` web application.

of all of the resulting images (see Figure N).

Figure N shows the archive links page. This page contains links to every proposal that contains publicly availble ACS data via the 4-5 digit `PROPOSID`. Clicking one of the links opens a new window to a 'view proposal' webpage (see Figure N). Similar to the database query results shown in Figure N, the 'view proposal' webpage shows thumbnail images of every ACS dataset available in the given proposal. The 'view proposal' pages contain several buttons and filters near the top of the page; using these will sort and/or filter out the thumbnail images based on the chosen paramters. Figure N 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 particular image. An example of this webpage is shown in Figure N, using the dataset `jcs718kmq`.

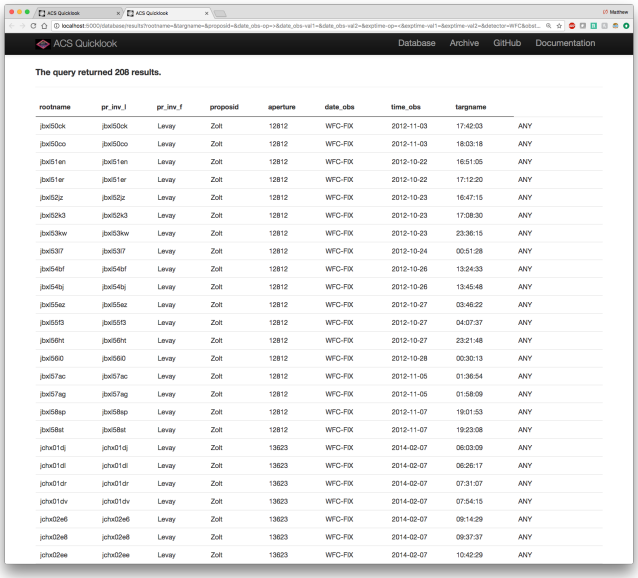

Fig. 22: The results of the database query shown in Figure N in the form of an HTML table.

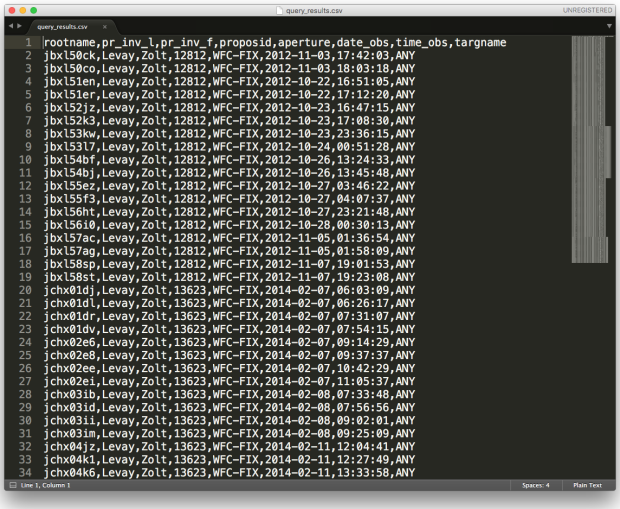

Fig. 23: The results of the database query shown in Figure N in the form of an CSV file.

This webpage contains the "Quicklook" JPEG of the image along with useful metadata of the dataset near the top of the page. It also contains an additional link (`proposal`) to view the other images within the particular proposal via the appropriate 'view proposal' webpage.

3.9.2 *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`

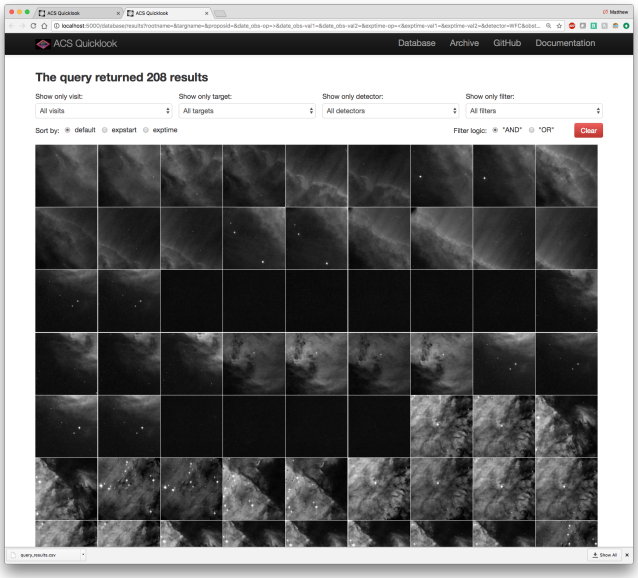

Fig. 24: The results of the database query shown in Figure N in the form of thumbnail images.

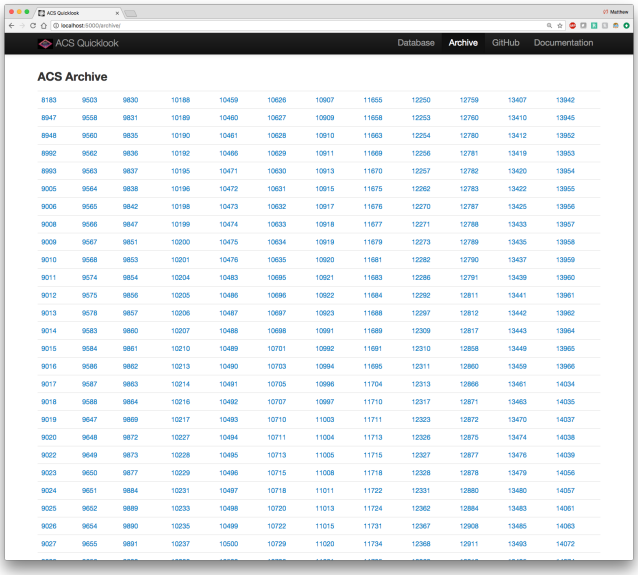

Fig. 25: The archive links webpage of the `acsql` web application.

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

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

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

The `acsql_webapp` module is the main module for running the `acsql` web application; it is this module that must be executed in order to serve the web application (either on a dedicated web server, or locally on a user's machine,

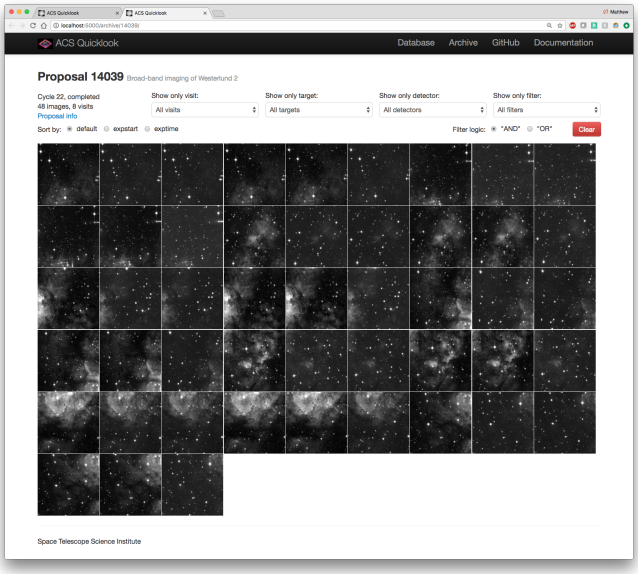

Fig. 26: An example of a 'view proposal' webpage, using proposal 14039.

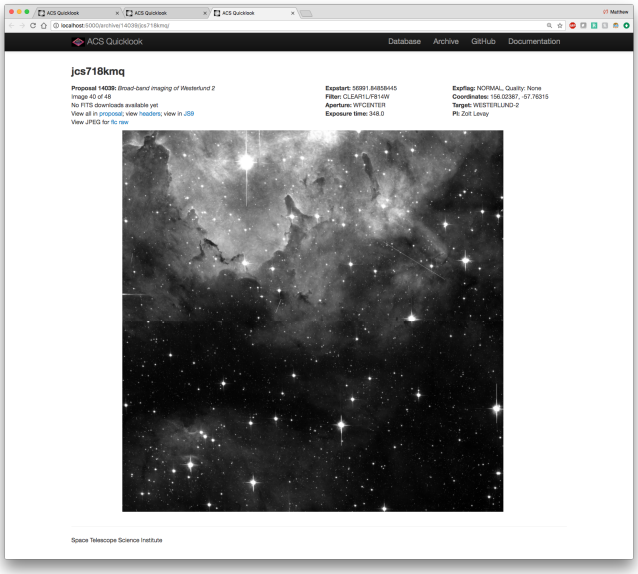

Fig. 27: An example of a 'view image' webpage, using dataset `jcs718kmq`.

accessed through the user's `localhost`). This module uses the `Flask` web framework to receive incoming `HTML` requests. It is structured such that each `acsql` webpage relates to an individual function within `acsql_webapp` (for example, accessing the `/archive/` webpage results in the execution of the `archive` function). Each of these functions contain the appropriate `Python` logic and rendering of `HTML` templates to return the desired webpage.

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

Along those same lines, the `form_options.py` module is simply a container module for storing the possible form options (e.g. apertures, filters, output column, proposal types, etc.) for the database query form page. These data could have been defined within the `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 that validate form entries. Several of the form objects are extensions of components (via subclassing) provided by the `wtforms` library, which is a form validation and rednering library for `Python` web development (wtforms). The query_lib module contains functions that parse, build, validate, and return `SQLAlchemy query` objects from the results of a submitted database query form. These queries are then executed on the `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 webpage, as well as a template that houses `HTML` that is common to each webpage (e.g. the menu bar), named `base.html`. Aside from some minor tweaks, we use templates from `Bootstrap`, which is a front-end component library and open source toolkit for developing `HTML`, `CSS`, and `JS` (bootstrap).

3.10 `acsql` Package

All code 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:

`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 installtion script. Executing this script with `python setup.py install` installs the package into the software environment.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`ingest.py`: A `Python` module for performing the ingestion of a single file (see Section 4.7).

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

make_jpeg.py: A Python module for creating a JPEG image from an individal file (see Section 3.7).

make_thumbnail.py: A Python module for creating a Thumbnail image from an individal JPEG (see Section 3.7).

scripts/: The scripts subpackage, containing Python modules for ingesting multiple files from the acsql filesystem, as well as storage place for possible future instrument calibration and monitoring routines.

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

utils/: The utils subpackage, containing Python modules that are useful for general acsql operations (e.g. configuring loggging, 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. Specifially, 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 reterning various 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 databse 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).

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/.

4 RESULTS

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

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

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

4.1 Filesystem Deliverable

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

For the former, we utilzed an already-existing filesystem of ACS data internal to STScI known as the "MAST public cache"; this filesystem is organized in a very similar way to that shown in Figure N[5]. 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 N 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 occurr for nearly every observing mode. On the other hand, we see a small amount of crj and crc files, which is also unsurprising considering that these filetypes are only triggered for specific observing modes.

As per item (2), the filesystem of JPEGs and Thumbnails was constructed during the ingestion of all publicly-available ACS data (via ingest_production.py, see section 3.10). The location of the JPEG/Thumbnail filesystem is

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

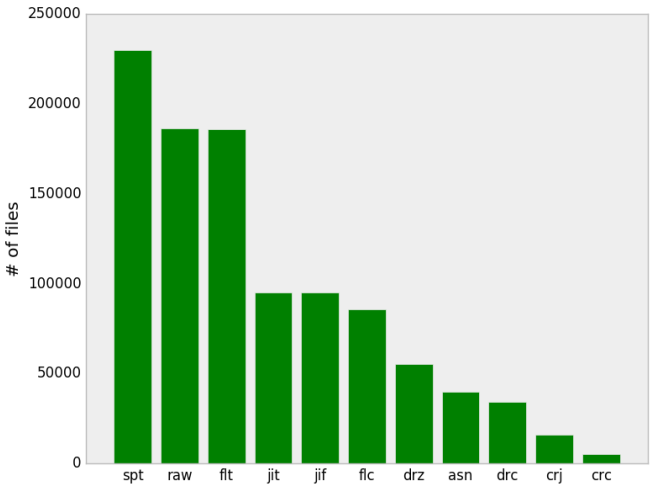

Fig. 28: The number of files in the `acsql` filesystem by filetype.

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[6].

4.2 Database Deliverable

As described in Sections 3.5 and 3.6, the `acsql` database is a `MySQL` database that stores the header information of every public ACS dataset. Currently, this database is hosted on a server that is internal to STScI. However, external users may build their own copy of the database via the `ingest_production.py` and `database_interface` modules. All 111 tables of `acsql` database are up to date as of the time of this writing (September 2017), and it is intended that the database will be kept up-to-date via periodic executions of `ingest_production.py`.

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

4.3 Web Application Deliverable

As described in Section 3.9, the `acsql` web application consists of a series of `Python` modules and static files. Currently, the web application is not yet hosted on a dedicated web server (neither externally nor internally to STScI[7]), 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).

6. recall that Thumbnail images are only generated for `flt` filetypes, hence the descrepency in number

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

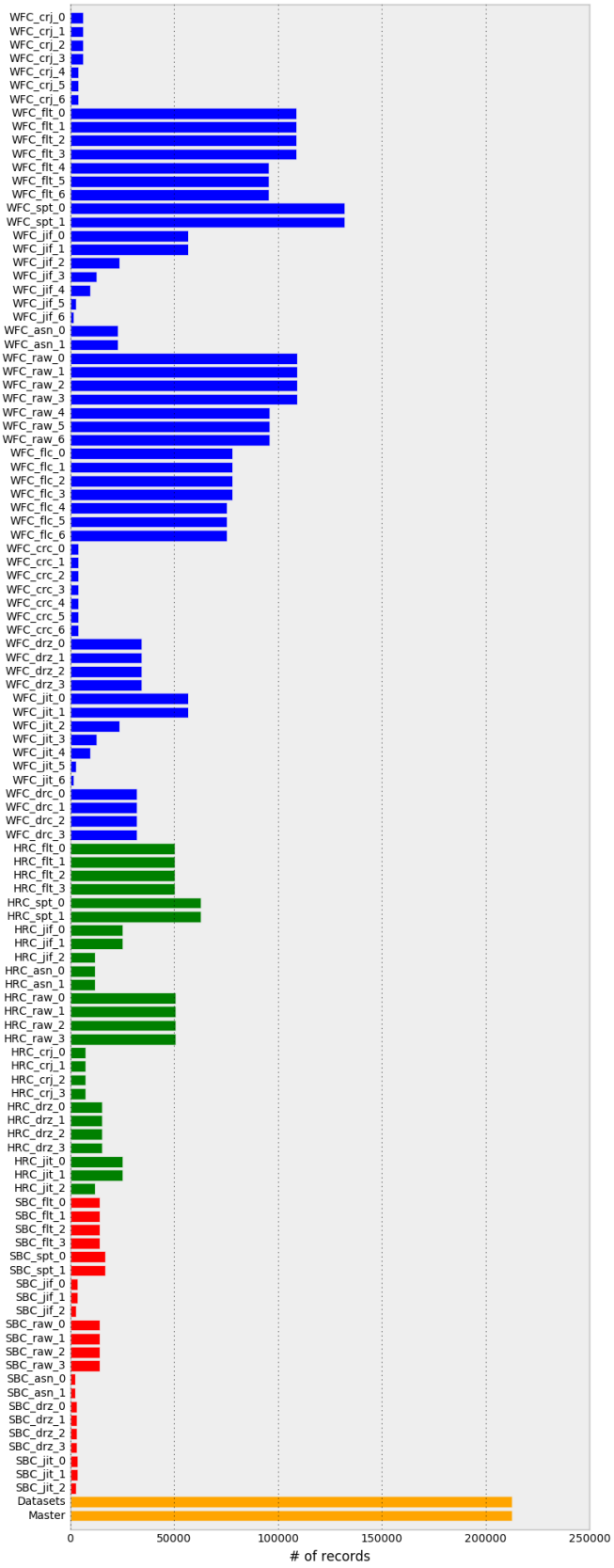

Fig. 29: The number of records in each of the `acsql` database tables.

4.4 Software Package Deliverable

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

4.5 Software Documentation Deliverable

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

5 Discussion

5.1 Possible enhancements to the `acsql` system

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

5.2 Possible simplifications based on MAST archive

5.3 Possible extensions to other instruments or missions

6 Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thanks 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 project, without which there would be no inspiration to create this similar project (Varun Bajaj, Ariel Bowers, Michael Dulue, Meredith Durbin, Jules Fowler, Catherine Gosmeyer, Heather Gunning, Harish Khandrika, Catherine Martlin, Abhi Rajan, Clare Shanahan, Ben Sunnquist, Alex Viana); members of the Instruments Division, who support the creation and maintenance of an internal web server (Francesca Boffi, Norman Grogin); Members of the Information Technology and Services Division who provided valuable resources such as the database server and disk space used to store project materials; and finally, Named Person, who provided insightful comments and suggestions on this document.

References

[1] *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

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

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

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

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

[6] *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.

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

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

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

[10] 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/.

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

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

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

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

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

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

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

[18] 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.

[19] 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/.

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

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')
```

```
279
280 WFC_spt_0 = orm_factory('WFC_spt_0')
281 WFC_spt_1 = orm_factory('WFC_spt_1')
282
283 WFC_drz_0 = orm_factory('WFC_drz_0')
284 WFC_drz_1 = orm_factory('WFC_drz_1')
285 WFC_drz_2 = orm_factory('WFC_drz_2')
286 WFC_drz_3 = orm_factory('WFC_drz_3')
287
288 WFC_drc_0 = orm_factory('WFC_drc_0')
289 WFC_drc_1 = orm_factory('WFC_drc_1')
290 WFC_drc_2 = orm_factory('WFC_drc_2')
291 WFC_drc_3 = orm_factory('WFC_drc_3')
292
293 WFC_crj_0 = orm_factory('WFC_crj_0')
294 WFC_crj_1 = orm_factory('WFC_crj_1')
295 WFC_crj_2 = orm_factory('WFC_crj_2')
296 WFC_crj_3 = orm_factory('WFC_crj_3')
297 WFC_crj_4 = orm_factory('WFC_crj_4')
298 WFC_crj_5 = orm_factory('WFC_crj_5')
299 WFC_crj_6 = orm_factory('WFC_crj_6')
300
301 WFC_crc_0 = orm_factory('WFC_crc_0')
302 WFC_crc_1 = orm_factory('WFC_crc_1')
303 WFC_crc_2 = orm_factory('WFC_crc_2')
304 WFC_crc_3 = orm_factory('WFC_crc_3')
305 WFC_crc_4 = orm_factory('WFC_crc_4')
306 WFC_crc_5 = orm_factory('WFC_crc_5')
307 WFC_crc_6 = orm_factory('WFC_crc_6')
308
309 WFC_jif_0 = orm_factory('WFC_jif_0')
310 WFC_jif_1 = orm_factory('WFC_jif_1')
311 WFC_jif_2 = orm_factory('WFC_jif_2')
312 WFC_jif_3 = orm_factory('WFC_jif_3')
313 WFC_jif_4 = orm_factory('WFC_jif_4')
314 WFC_jif_5 = orm_factory('WFC_jif_5')
315 WFC_jif_6 = orm_factory('WFC_jif_6')
316
317 WFC_jit_0 = orm_factory('WFC_jit_0')
318 WFC_jit_1 = orm_factory('WFC_jit_1')
319 WFC_jit_2 = orm_factory('WFC_jit_2')
320 WFC_jit_3 = orm_factory('WFC_jit_3')
321 WFC_jit_4 = orm_factory('WFC_jit_4')
322 WFC_jit_5 = orm_factory('WFC_jit_5')
323 WFC_jit_6 = orm_factory('WFC_jit_6')
324
325 WFC_asn_0 = orm_factory('WFC_asn_0')
326 WFC_asn_1 = orm_factory('WFC_asn_1')
327
328
329 # HRC tables
330 HRC_raw_0 = orm_factory('HRC_raw_0')
331 HRC_raw_1 = orm_factory('HRC_raw_1')
332 HRC_raw_2 = orm_factory('HRC_raw_2')
333 HRC_raw_3 = orm_factory('HRC_raw_3')
334
335 HRC_flt_0 = orm_factory('HRC_flt_0')
336 HRC_flt_1 = orm_factory('HRC_flt_1')
337 HRC_flt_2 = orm_factory('HRC_flt_2')
338 HRC_flt_3 = orm_factory('HRC_flt_3')
339
340 HRC_spt_0 = orm_factory('HRC_spt_0')
341 HRC_spt_1 = orm_factory('HRC_spt_1')
342
343 HRC_drz_0 = orm_factory('HRC_drz_0')
344 HRC_drz_1 = orm_factory('HRC_drz_1')
345 HRC_drz_2 = orm_factory('HRC_drz_2')
346 HRC_drz_3 = orm_factory('HRC_drz_3')
347
348 HRC_crj_0 = orm_factory('HRC_crj_0')
349 HRC_crj_1 = orm_factory('HRC_crj_1')
350 HRC_crj_2 = orm_factory('HRC_crj_2')
351 HRC_crj_3 = orm_factory('HRC_crj_3')
352
353 HRC_jif_0 = orm_factory('HRC_jif_0')
354 HRC_jif_1 = orm_factory('HRC_jif_1')
355 HRC_jif_2 = orm_factory('HRC_jif_2')
```

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

acsql/database/make_tabledefs.py

```
1  #! /usr/bin/env/ python
2
3  """Creates static text files that hold header keywords and keyword
4  data types for each ACS filetype (and each extension).  Each text file
5  corresponds to a header table in the ``acsql`` database.
6
7  Authors
8  -------
9      - Sara Ogaz
10     - Matthew Bourque
11
12 Use
13 ---
14     This module is intended to be run via the command line as such:
15     ::
16
17         python make_tabledefs.py
18
19 Dependencies
20 ------------
21     External library dependencies include:
22
23     - ``acsql``
```

```
    - ``numpy``
    - ``stak`` (``https://github.com/spacetelescope/stak``)

Notes
-----
    The ``stak.Hselect`` dependency still depends on Python 2 at the
    time of this writing.
"""

import glob
import os

import numpy as np
#from stak import Hselect

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


def make_tabledefs(detector):
    """
    Function to auto-produce the table_definition files.

    Note that due to how ``hselect`` handles ``ASN`` files, they must
    be handeled separately.

    Parameters
    ----------
    detector : str
        The detector (e.g. ``wfc``).
    """

    file_exts = getattr(utils, '{}_FILE_EXTS'.format(detector.upper()))

    for ftype in file_exts:

        all_files = glob.glob('table_definitions/test_files/test_{}_{}*.fits'\
            .format(detector, ftype))

        for ext in file_exts[ftype]:

            filename = 'table_definitions/{}_{}_{}.txt'.format(detector,
                ftype, ext)
            hsel = Hselect(all_files, '*', extension=(ext,))

            print('Making file {}'.format(filename))
            with open(filename, 'w') as f:
                for col in hsel.table.itercols():
                    column_name = col.name
                    if column_name in ['ROOTNAME', 'Filename', 'FILENAME', 'Ext']:
                        continue
                    elif col.dtype in [np.dtype('S68'), np.dtype('S80')]:
                        ptype = 'String'
                    elif col.dtype in [np.int64]:
                        ptype = 'Integer'
                    elif col.dtype in [bool]:
                        ptype = 'Bool'
                    elif col.dtype in [np.float64]:
                        ptype = 'Float'
                    else:
                        print('Could not find type match: {}:{}'.format(
                            column_name, col.dtype))

                    # If the column has a hyphen, switch it to underscore
                    if '-' in column_name:
                        column_name = column_name.replace('-', '_')

                    f.write('{}, {}\n'.format(column_name, ptype))


if __name__ == '__main__':

    make_tabledefs('wfc')
    make_tabledefs('sbc')
    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


```

```
def rootnames_for_target(targname):
    """Queries for the rootname and filename for a given target.

    Parameters
    ----------
    targname : str
        The target name (e.g. 'NGC104')

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

    query = session.query(WFC_raw_0.rootname, WFC_raw_0.filename,
        WFC_raw_0.targname)\
            .filter(WFC_raw_0.targname == targname)
    query_results = query.all()

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

    for result in query_results:
        print(result)

    return query


def filenames_for_calibration(calibration_keyword, value):
    """Queries for the filenames that used a given calibration mode.

    The 'calibration' mode is defined by the type of calibration and
    the calibration reference file used
    (e.g. 'BIASFILE = jref$06u15056j_bia.fits')

    Parameters
    ----------
    calibration_keyword : str
        The calibration file to query on (e.g. DARKFILE, BIASFILE)
    value : str
        The calibration file value (e.g. jref$06u15056j_bia.fits)

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

    calibration_keyword_obj = getattr(WFC_raw_0, calibration_keyword)
    query = session.query(WFC_raw_0.filename)\
        .filter(calibration_keyword_obj == value)
    query_results = query.all()

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

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

    return query


def goodmean_for_dataset(dataset):
    """Queries for the GOODMEAN values for a given dataset

    The GOODMEAN describes the mean of all 'good' (i.e. non-flagged)
    pixels in the image.

    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(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.

```

```
Authors
-------
    Matthew Bourque

Use
---
    This module is intended to be used when an ``acsql`` user sees fit
    to add any new header keywords to the database.  The module can
    be used from the command line as such:
    ::

        python update_tabledefs.py <logfile>

    Required arguments:
    ::

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

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

    - ``acsql``
    - ``numpy``
    - ``stak`` (``https://github.com/spacetelescope/stak``)

Notes
-----
    The ``stak.Hselect`` dependency still depends on Python 2 at the
    time of this writing.
"""

import argparse
import os

import numpy as np
#from stak import Hselect

from acsql.utils.utils import SETTINGS


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

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

    # Create help strings
    logfile_help = 'The path to the logfile to use to determine new header '
    logfile_help += 'keywords.'

    # Add arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('logfile',
                        action='store',
                        help=logfile_help)

    # Parse args
    args = parser.parse_args()

    # Ensure that the logfile exists
    assert os.path.exists(args.logfile), '{} does not exist.'\
        .format(args.logfile)

    return args


def update_tabledefs(logfile):
    """The main function of the ``update_tabledefs`` module.  See
    module documentation for further details.

    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']))


```

```
def update_header_table(file_dict, ext):
    """Insert/update an entry for the file in the appropriate header
    table (e.g. ``wfc_raw_0``).

    The header table that get updated depend on the detector, filetype,
    and extension.

    Parameters
    ----------
    file_dict : dict
        A dictionary containing various data useful for the ingestion
        process.
    ext : int
        The header extension.
    """

    # Check if header is an ingestable header before proceeding
    valid_extnames = ['PRIMARY', 'SCI', 'ERR', 'DQ', 'UDL', 'jit', 'jif',
                      'ASN', 'WHT', 'CTX']
    ext_exists = True
    try:
        header = fits.getheader(file_dict['filename'], ext)
        if ext == 0:
            extname = 'PRIMARY'
        else:
            extname = header['EXTNAME']
    except IndexError:
        ext_exists = False
        extname = None

    # Ingest the header if it is ingestable
    if ext_exists and extname in valid_extnames:

        table = "{}_{}_{}".format(file_dict['detector'].upper(),
                                  file_dict['filetype'].lower(),
                                  str(ext))

        exclude_list = ['HISTORY', 'COMMENT', 'ROOTNAME', 'FILENAME', '']
        input_dict = {'rootname': file_dict['rootname'],
                      'filename': file_dict['basename']}

        try:
            for key, value in header.items():
                key = key.strip()

                # Switch hypens to underscores
                if '-' in key:
                    key = key.replace('-', '_')

                if key in exclude_list or value == "":
                    continue
                elif key not in TABLE_DEFS[table.lower()]:
                    logging.warning('{}: {} not in {}'\
                        .format(file_dict['full_rootname'], key, table))
                    continue

                input_dict[key.lower()] = value

            insert_or_update(table, input_dict)
            logging.info('{}: Updated {} table.'.format(file_dict['rootname'],
                                                        table))

        except VerifyError as e:
            logging.warning('\tUnable to insert {} into {}: {}'.format(
                file_dict['rootname'], table, e))


def update_master_table(rootname_path):
    """Insert/update an entry in the ``master`` table for the given
    file.

    Parameters
    ----------
    rootname_path
        The path to the rootname directory in the MAST cache.
    """

```

```
    rootname = os.path.basename(rootname_path)[:-1]
    path = rootname_path[-15:]
    proposid = get_metadata_from_test_files(rootname_path, 'proposid')
    proposal_type = get_proposal_type(proposid)

    # Insert a record in the master table
    data_dict = {'rootname': rootname,
                 'path': path,
                 'first_ingest_date': date.today().isoformat(),
                 'last_ingest_date': date.today().isoformat(),
                 'detector': get_metadata_from_test_files(rootname_path,
                                                          'detector'),
                 'proposal_type': proposal_type}
    insert_or_update('Master', data_dict)
    logging.info('{}: Updated master table.'.format(rootname))


def ingest(rootname_path, filetype='all'):
    """The main function of the ingest module.  Ingest a given rootname
    (and its associated files) into the various tables of the ``acsql``
    database.

    If for some reason the file is unable to be ingested, a warning is
    logged.

    Parameters
    ----------
    rootname_path : str
        The path to the rootname directory in the MAST cache.
    """

    rootname = os.path.basename(rootname_path)[:-1]
    logging.info('{}: Begin ingestion'.format(rootname))

    # Update the master table for the rootname
    update_master_table(rootname_path)

    if filetype == 'all':
        search = '*.fits'
    else:
        search = '*{}.fits'.format(filetype)
    file_paths = glob.glob(os.path.join(rootname_path, search))

    for filename in file_paths:
        filetype = os.path.basename(filename).split('.')[0][10:]
        if filetype in VALID_FILETYPES:

            # Make dictionary that holds all the information you would ever
            # want about the file
            file_dict = make_file_dict(filename)

            # Update header tables
            if 'file_exts' in file_dict:
                for ext in file_dict['file_exts']:
                    update_header_table(file_dict, ext)

                # Update datasets table
                update_datasets_table(file_dict)

                # Make JPEGs and Thumbnails
                if file_dict['filetype'] in ['raw', 'flt', 'flc']:
                    make_jpeg(file_dict)
                if file_dict['filetype'] == 'flt':
                    make_thumbnail(file_dict)

    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``
"""
```

```
33
34 from collections import OrderedDict
35 import glob
36 import os
37
38 from flask import Flask, render_template, request, Response
39 import numpy as np
40
41 from acsql.utils.utils import SETTINGS
42 from acsql.website.data_containers import get_view_image_dict
43 from acsql.website.data_containers import get_view_proposal_dict
44 from acsql.website.data_containers import get_view_query_results_dict
45 from acsql.website.query_form import get_query_form
46 from acsql.website.query_lib import generate_csv
47 from acsql.website.query_lib import get_query_results
48
49 app = Flask(__name__)
50
51
52 @app.route('/archive/')
53 def archive():
54     """Returns webpage containing links to all ACS archive proposals.
55
56     Returns
57     -------
58     template : obj
59         The ``archive.html`` template.
60     """
61
62     # Get list of all proposal numbers
63     proposal_list = glob.glob(os.path.join(SETTINGS['jpeg_dir'], '*'))
64     proposal_list = sorted([int(os.path.basename(item)) for item in proposal_list])
65
66     # rearrange list so that it appears in multiple columns
67     ncols = 12
68     if len(proposal_list) % ncols != 0:
69             proposal_list.extend([''] * (ncols - (len(proposal_list) % ncols)))
70     proposal_array = np.asarray(proposal_list).reshape(ncols, int(len(proposal_list) / ncols)).T
71
72     return render_template('archive.html', proposal_array=proposal_array)
73
74
75 @app.route('/database/')
76 @app.route('/database/results')
77 def database():
78     """Returns webpage containing a query form for querying the
79     ``acsql`` database.
80
81     Returns
82     -------
83     template : obj
84         The ``database.html`` webpage.
85     """
86
87     query_form = get_query_form(request.args)
88
89     if request.query_string:
90         if query_form.validate():
91             query_form_dict = request.args.to_dict(flat=False)
92             query_results_dict = get_query_results(query_form_dict)
93
94             results = query_results_dict['query_results']
95             num_results = query_results_dict['num_results']
96             output_format = query_results_dict['output_format']
97             output_columns = query_results_dict['output_columns']
98
99             # If something went wrong with the query
100            if num_results is None:
101                template = render_template(
102                    'database_error.html',
103                    form=query_form,
104                    msg=num_results)
105
106            # If the query returned no results
107            elif num_results == 0:
108                results = True
109                template = render_template(
```

```
                    '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()

    # Parse the results
    data_dict['detectors'] = [detector for item in results]
    data_dict['apertures'] = [item[0] for item in results]
    data_dict['exptimes'] = [item[1] for item in results]
    data_dict['filter1s'] = [item[2] for item in results]
    data_dict['filter2s'] = [item[3] for item in results]
    data_dict['targnames'] = [item[4] for item in results]
    data_dict['dateobss'] = [item[5] for item in results]
    data_dict['timeobss'] = [item[6] for item in results]
    data_dict['expstarts'] = [item[7] for item in results]
    data_dict['expflags'] = [item[8] for item in results]
    data_dict['qualitys'] = [item[9] for item in results]
    data_dict['ras'] = [item[10] for item in results]
    data_dict['decs'] = [item[11] for item in results]
    data_dict['pi_firsts'] = [item[12] for item in results]
    data_dict['pi_lasts'] = [item[13] for item in results]

    return data_dict


def _get_buttons_dict(data_dict):
    """Add data used for various buttons on the ``/archive/<proposal>``
    page or ``/database/results/`` page to the ``data_dict``.

    Parameters
    ----------
    data_dict : dict
        A dictionary containing data for the given
        ``/archive/<proposal>`` or ``/database/results/`` page, such as
        ``visits`` and ``targnames``

    Returns
    -------
    data_dict : dict
        A dictionary containing data for the given
        ``/archive/<proposal>`` or ``/database/results/`` page, such as
        ``visits`` and ``targnames``
    """

    data_dict['buttons'] = {}
    data_dict['buttons']['detector'] = sorted(set(data_dict['detectors']))
    data_dict['buttons']['visit'] = sorted(set(data_dict['visits']))
```

```
    data_dict['buttons']['target'] = sorted(set(data_dict['targnames']))
    data_dict['buttons']['filter'] = sorted(set([
        '{}/{}'.format(filter1, filter2)
        for filter1, filter2
        in zip(data_dict['filter1s'], data_dict['filter2s'])]))

    return data_dict


def _get_proposal_status(data_dict):
    """Add proposal status information (e.g. ``proposal_title``,
    ``cycle``, etc.) to the ``data_dict``.

    The proposal status information is scraped from the proposal
    status webpage.

    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.
    """

    data_dict['status_page'] = (
        'http://www.stsci.edu/cgi-bin/get-proposal-info?id={}'
        '&submit=Go&observatory=HST').format(data_dict['proposal_id'])

    req = requests.get(data_dict['status_page'], timeout=3)

    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 != ['']}
```

```
207
208     # Combine data returned from fields with operator dropdowns
209     for operator_key in operator_keys:
210         operator_dict = {}
211         for key, value in list(query_form_dict.items()):
212             if key == operator_key + '-op':
213                 operator_dict['op'] = value[0]
214             elif key == operator_key + '-val1':
215                 operator_dict['val1'] = value[0]
216             elif key == operator_key + '-val2':
217                 operator_dict['val2'] = value[0]
218         if len(operator_dict) > 1:
219             query_form_dict[operator_key] = operator_dict
220
221     return query_form_dict
222
223
224 def generate_csv(output_columns, results):
225     """Create a CSV file of the database query ouput.
226
227     Parameters
228     ----------
229     output_columns : list
230         A list of columns desired for the output file.
231     results : list
232         A list of results from the database query
233     """
234
235     header = ','.join(output_columns) + '\n'
236     yield header
237
238     for result in results:
239         if len(result) == 1:
240             yield str(result[0]) + '\n'
241         else:
242             yield ','.join(map(str, result)) + '\n'
243
244
245 def get_query_results(query_form_dict):
246     """Returns a dictionary with the results of the requested query
247     along with some additional metadata.  Calls on several internal
248     functions to build and perform the query in order to abstract
249     out its complexity.
250
251     Parameters
252     ----------
253     query_form_dict : dict
254         A dictionary containing information about the requested query,
255         such as the ``output_format``, ``output_columns`` and the
256         requested values.
257
258     Returns
259     -------
260     query_results_dict : dict
261         A dictionary containing the query results as well as some
262         metadata such as ``output_format`` and number of results.
263     """
264
265     # Determine output format
266     output_format = query_form_dict.pop('output_format')
267     if output_format == ['thumbnails']:
268         output_columns = ['rootname', 'filename', 'detector',
269                           'proposal_type', 'expstart', 'filter1',
270                           'filter2', 'exptime', 'targname', 'proposid']
271         if 'output_columns' in query_form_dict:
272             query_form_dict.pop('output_format', None)
273     else:
274         output_columns = query_form_dict.pop('output_columns')
275
276     # Remove blank entries from form data
277     query_form_dict = _convert_query_form_dict(query_form_dict)
278
279     # Build the query
280     wfc_query, hrc_query, sbc_query = _build_queries(output_columns)
281
282     # Perform filtering on the query
283     for key, value in list(query_form_dict.items()):
```

```
        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

    # Put results in a dictinoary
    query_results_dict = {}
    query_results_dict['num_results'] = num_results
    query_results_dict['query_results'] = query_results
    query_results_dict['output_format'] = output_format
    query_results_dict['output_columns'] = output_columns

    return query_results_dict


def _get_session():
    """Return a ``session`` object to be used as a conenction to the
    ``acsql`` database.

    Returns
    -------
    session : obj
        A ``SQLAlchemy`` ``session`` object which serves as a
        connection to the ``acsql`` database.
    """

    engine = create_engine(SETTINGS['connection_string'], echo=False, pool_timeout=100000)
    Session = sessionmaker(bind=engine)
    session = Session()

    return session


def _merge_query(wfc_query, hrc_query, sbc_query):
    """Merge the results from the queries from each table

    Parameters
    ----------
    wfc_query : obj
        The ``SQLAlchemy`` ``query`` object for request columns of the
        WFC table.
    hrc_query : obj
        The ``SQLAlchemy`` ``query`` object for request columns of the
        HRC table.
    sbc_query : obj
        The ``SQLAlchemy`` ``query`` object for request columns of the
        SBC table.

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

    # Turn off the query if the table is not needed
    if wfc_query:
        if str(wfc_query.statement).find('WHERE') == -1:
```

```
            wfc_query = False
    if hrc_query:
        if str(hrc_query.statement).find('WHERE') == -1:
            hrc_query = False
    if sbc_query:
        if str(sbc_query.statement).find('WHERE') == -1:
            sbc_query = False

    # If combination needed
    if wfc_query and hrc_query and sbc_query:
        query = wfc_query.union_all(hrc_query, sbc_query)
    elif wfc_query and not hrc_query and sbc_query:
        query = wfc_query.union_all(sbc_query)
    elif wfc_query and hrc_query and not sbc_query:
        query = wfc_query.union_all(hrc_query)
    elif not wfc_query and hrc_query and sbc_query:
        query = hrc_query.union_all(sbc_query)

    # If no combination needed
    elif wfc_query and not hrc_query and not sbc_query:
        query = wfc_query
    elif not wfc_query and hrc_query and not sbc_query:
        query = hrc_query
    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>

        <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> &
    nbsp; 
                <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 %}
```