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

Matthew Bourque[1], Sara Ogaz[1], Alex Viana[2], Meredith Durbin[3], 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—What is an abstract, really?

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 discussed in section 1.1.

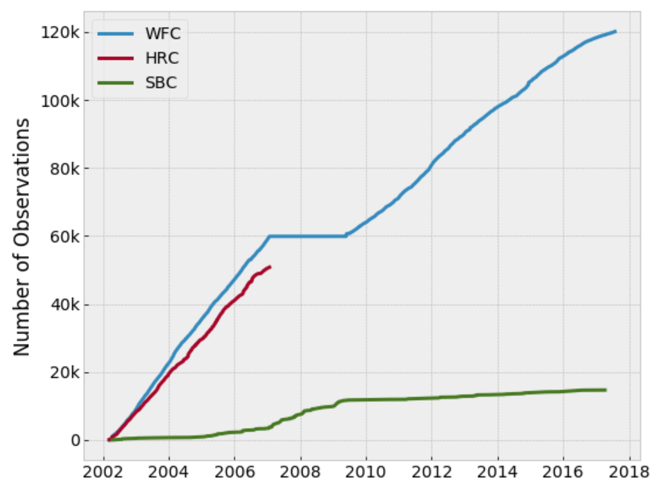

Fig. 1. The number of observations over time for each of the three detectors on ACS.

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

• *M. Bourque, S. Ogaz, A. Viana, and M. Durbin were with the Space Telescope Science Institute, Baltimore, MD, 21218.*
E-mail: bourque@stsci.edu

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.

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.

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'.

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

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).

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

```
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.

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.

TARGNAME PROPOSID DATE-OBS EXPTIME EXPSTART APERTURE DETECTOR OBSTYPE FILTER1 FILTER2 TIME-OBS SUBARRAY RA DEC EXPFLAG IMAGETYP rootname proposal_type

1.3 Motivation

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

Data retreival letency: Currently, users who wish to retreive data from the MAST archive 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 is a non-significant amount of time. In the fastest scenario of the `sftp` option, a typical request can take minutes to hours to be completed. The ACS Quicklook system attempts to circumnavigate this retreival process by making the full data products instantly available via read-only access of the filesystem subystem, as well as a subset of the data products (and corresponding metadata) instantly available to view through the web application.

File I/O: Users who

Data redundancy: Something.

Data discovery: Something

1.4 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. View

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

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.

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

ingest.ingest.get_proposal_type(*proposid*)

Return the `proposal_type` for the given `proposid`.

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

Parameters: **proposid** : str

The proposal ID (e.g. `12345`).

Returns: **proposal_type** : int or None

The proposal type (e.g. `CAL`).

Fig. 4. The `readthedocs` documentation for the `acsql` example function seen in Figure N.

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 `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.

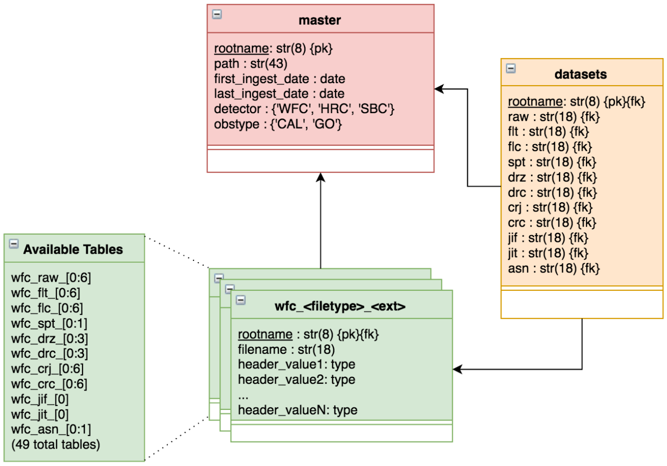

Fig. 5. The relational database schema for the `acsql` database.

3.3 Filesystem: The MAST public Cache

3.4 Filesystem: Archive of JPEGs and Thumbnails

3.5 Database: Relational Schema

3.6 Database: MySQL + SQLAlchemy

3.7 Database: ORMs

3.8 Data ingestion software

3.9 Website:

4 Results

Topics to discuss: 1. GitHub repository 2. ReadTheDocs documentation repository 3. Quantification of Database records 4. Quantification of Code repository 5. Website location

5 Conclusion

The conclusion goes here.

6 Discussion

Topics to discuss:

1. Possible simplification based on MAST archive 2. Possible extensions to other insturments

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgments

The authors would like to thank...

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] Brandi, G., et al., 2007, *Sphinx: Python Documentation Generator*, available at http://www.sphinx-doc.org/en/stable/.
[4] *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.
[5] *git*, [Online; accessed 2017-08-05], available at https://git-scm.com.
[6] *GitHub*, [Online; accessed 2017-08-05], available at https:/github.com.
[7] Goodger, D., 2001, *PEP 257 – Docstring Conventions*, Python Developer's Guide, available at https://www.python.org/dev/peps/pep-0257/.
[8] Loper, E., 2004, *Epydoc: Generating API Documentation in Python*, Proceedings of the Second Annual Python Conference, available at http://epydoc.sourceforge.net/.
[9] *Read the Docs*, [Online; accessed 2017-08-05], available at https://readthedocs.org.
[10] Robitaille, T.P., et al., 2013, *Astropy: A community Python package for astronomy*, Astronomy & Astrophysics, 558, A33.
[11] Smith, E., et al., 2011, *Introduction to the HST Data Handbooks*, Version 8.0 (Baltimore: STScI)
[12] *The Barbara A. Mikulski Archive for Space Telescopes*, [Online; accessed 2017-07-30], available at https://archive.stsci.edu/.
[13] 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.
[14] 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/.