Control and systems software for the Cosmology Large Angular Scale Surveyor (CLASS)

Matthew A. Petroff,[a] John W. Appel,[a] Charles L. Bennett,[a] Michael K. Brewer,[a] Manwei Chan,[a] David T. Chuss,[b] Joseph Cleary,[a] Jullianna Denes Couto,[a] Sumit Dahal,[c,a] Joseph R. Eimer,[a] Thomas Essinger-Hileman,[c,a] Pedro Fluxá Rojas,[d] Kathleen Harrington,[e,a] Jeffrey Iuliano,[f,a] Tobias A. Marriage,[a] Nathan J. Miller,[a,c] Deniz Augusto Nunes Valle,[a] Duncan J. Watts,[g,a] and Zhilei Xu[h,a]

[a]Department of Physics & Astronomy, Johns Hopkins University, Baltimore, MD 21218, USA
[b]Department of Physics, Villanova University, Villanova, PA 19085, USA
[c]NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
[d]Facultad de Física, Pontificia Universidad Católica de Chile, 7820436 Macul, Santiago, Chile
[e]Department of Astronomy & Astrophysics, University of Chicago, Chicago, IL 60637, USA
[f]Department of Physics & Astronomy, University of Pennsylvania, Philadelphia, PA 19104, USA
[g]Institute of Theoretical Astrophysics, University of Oslo, 0315 Oslo, Norway
[h]MIT Kavli Institute, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

ABSTRACT

The Cosmology Large Angular Scale Surveyor (CLASS) is an array of polarization-sensitive millimeter wave telescopes that observes $\sim 70\%$ of the sky at frequency bands centered near $40\,\mathrm{GHz}$, $90\,\mathrm{GHz}$, $150\,\mathrm{GHz}$, and $220\,\mathrm{GHz}$ from the Atacama desert of northern Chile. Here, we describe the architecture of the software used to control the telescopes, acquire data from the various instruments, schedule observations, monitor the status of the instruments and observations, create archival data packages, and transfer data packages to North America for analysis. The computer and network architecture of the CLASS observing site is also briefly discussed. This software and architecture has been in use since 2016, operating the telescopes day and night throughout the year, and has proven successful in fulfilling its design goals.

Keywords: telescope control, software, cosmic microwave background, telescopes

1. INTRODUCTION

Polarization anisotropy in the cosmic microwave background (CMB) provide a wealth of information about the early Universe. The Cosmology Large Angular Scale Surveyor (CLASS) is an array of polarization-sensitive millimeter wave telescopes that observes $\sim 70\%$ of the sky at frequency bands centered near $40\,\mathrm{GHz}$, $90\,\mathrm{GHz}$, $150\,\mathrm{GHz}$, and $220\,\mathrm{GHz}$ from the Atacama desert of northern Chile.[1,2] The primary science goals of CLASS are two-fold—to search for polarization B-modes[3] as strong evidence of cosmological inflation and to make a measurement of the optical depth due to reionization, via polarization E-modes, with a sensitivity that is near the cosmic variance limit.[4,5] The inflation paradigm postulates that the Universe underwent a period of exponential expansion in its first moments. Such inflationary expansion would produce tensor perturbations in the form of gravitational waves, which would leave an imprint on the CMB.[6] The photons of the CMB were emitted during the period of decoupling that followed the epoch of recombination. Photons streaming through space collided with these free electrons, producing linear polarization via Thomson scattering. In the presence of the tensor perturbations induced by gravitational waves, this scattering produces B-mode polarization in addition to E-mode polarization, which is additionally produced by scattering in the presence of the dominant scalar perturbations. Thus, B-mode polarization in the CMB is evidence for gravitational

---

wave-induced quadrupolar fluctuations during recombination, which is evidence for inflation. After a period known as the Dark Ages, the first stars formed, which reionized the neutral hydrogen in the Universe. This resulted in more Thomson scattering, which, in the presence of primordial gravitational waves, produces a second B-mode feature in the CMB, as well as additional scalar-sourced E-mode features.[7] This polarization signal is at large angular scales, which CLASS is uniquely sensitive to among ground-based experiments.

CLASS, which has been successfully observing since 2016, seeks to accomplish its science goals by mapping the polarization of the CMB at large angular scales ($2 \lesssim \ell \lesssim 200$). Fast front-end polarization modulation, in the form of a variable-delay polarization modulator (VPM) as its first optical element,[8,9] gives CLASS the stability necessary to measure the largest angular scales on the sky while also allowing most forms of instrument polarization to be rejected. Furthermore, the sky is observed at multiple frequencies to enable Galactic foreground signals, those from synchrotron and thermal dust in particular, to be disentangled from the primordial CMB signal.

In order to perform these measurements, software and computing and networking equipment are necessary to perform observations, collect time-ordered data, and package and transfer data for further analysis. To this end, this paper describes the architecture of the software used to control the telescopes, acquire data from the various instruments, schedule observations, monitor the status of the instruments and observations, create archival data packages, and transfer data packages to North America for analysis. The telescopes operate at a high-altitude site in northern Chile located on Cerro Toco and are connected to the internet via a facility in the nearby town of San Pedro de Atacama. Data are then transferred to a data server located on the Johns Hopkins University campus in Baltimore, Maryland for further analysis. The remainder of this paper is structured as follows. Section 2 gives an overview of the network and software architecture, Section 3 describes how this software is interfaced to some key subsystems, Section 4 details the data packaging and transfer pipeline, and Section 5 outlines the web interface used to monitor status and schedule observations. Finally, we present some lessons learned in Section 6 and conclude in Section 7.

2. NETWORK AND SOFTWARE ARCHITECTURE

With multiple receivers and numerous housekeeping systems, more than a dozen computers are used to operate CLASS. Networking and a cohesive software control suite are therefore required to coordinate the operation of these systems such that science data can be collected, transferred off-site, and later analyzed.

2.1 Network layout

The CLASS network is divided into two main components, the site network on Cerro Toco and the supporting systems down the mountain in San Pedro de Atacama. These components are connected using a wireless network link operating at 5 GHz. This approximately 43 km line-of-sight radio link is provided by two Ubiquiti AF-5X transceivers* and is able to sustain data rates in excess of 100 Mbps. The site end of the wireless link is located on a tower ∼170 m from the telescopes, at the edge of a cliff and with a clear line-of-sight to the San Pedro de Atacama end of the link. The site end of the link is powered via a solar panel and batteries and is connected to the rest of the site network via a fiber optic cable, providing complete electrical isolation, to protect against lightning damage. All computers are powered via uninterruptible power supplies (UPSs) so that they remain online during the brief power losses sustained while switching between the site's two generators. Similarly, the equipment in San Pedro de Atacama is powered with a UPS; this UPS is capable of providing several hours of backup power, which is necessary due to the frequency and duration of power outages at this location. A overview of the CLASS network is given in Figure 1.

The San Pedro de Atacama portion of the network consists of a router, an analysis computer, and a remote access computer. The ANALYSIS MACHINE is used for doing preliminary analysis of data collected by the telescope. Additionally, it is used to store telescope data once it is transferred down the mountain, until the data are copied to North America. The Cerro Toco portion of the network consists of three major segments: the control room, the first mount, and the second mount. The CONTROL ROOM and MOUNTS all use
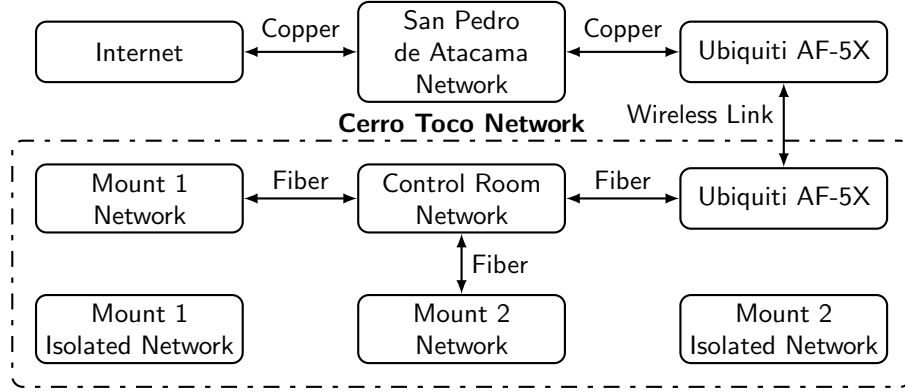
---

* Ubiquiti Inc.; `https://www.ui.com/`

Figure 1. Network architecture overview. Network segments in San Pedro de Atacama and on Cerro Toco are connected via a wireless link; other segments are connected via either copper or fiber optic Ethernet cables.

managed Ethernet switches, which provide gigabit connectivity over both copper and fiber optic cables, and are interconnected via fiber optic cables. Each MOUNT has an additional switch for communication between the mount computers and the mount servo motors, which are isolated from the primary network. The radio link to San Pedro de Atacama is connected to the CONTROL ROOM switch. The MOUNTS each contain two computers for controlling the mount; two housekeeping computers, one for each cryostat; a star camera; four servo motors, which are on an isolated network; and two VPM CONTROLLERS, one for each telescope. The CONTROL ROOM contains a server rack with a CENTRAL SERVER, a display and status computer, a command terminal, a star camera host computer, and detector readout computers, one for each detector readout unit.[*] Additionally, there are laptop computers that can be moved around as needed, to fulfill miscellaneous needs. A Wi-Fi access point is also installed, although this is turned off during observations.

2.2 Software structure

CLASS's software infrastructure is currently built around the Ubuntu 16.04 Linux distribution[†] and the Python programming language,[‡] specifically version 3.5. These releases are both supported through at least 2021.[§] Software is developed using the Git[¶] distributed version control system. All machines run the Ubuntu 16.04 operating system, except for the cryostat virtual machines that run Windows 7 for compatibility with vendor-provided proprietary software and the mount computers that run the VxWorks[‖] real-time operating system (RTOS). With the exception of some C code required to interface with some hardware and a few shell scripts, all control and data acquisition software running on the Linux servers is written in Python. The control scripts are run as `systemd`[**] services so that they start on boot.

The basic software structure consists of Python scripts running each of the telescopes' subsystems, with a central COMMAND SCRIPT to coordinate operations. Since control and data acquisition software is running on multiple computers on the network simultaneously, a control and communications system for these distributed systems is required. The PYRO[††] package, version 4, was chosen for this purpose. Each subsystem is operated by a separate Python script, which includes a PYRO interface. These PYRO interfaces expose the scripts' control function to the network. The COMMAND SCRIPT is then able to call these exposed functions as if

---

[*] The detector readout units are on the mount and interface with the detectors; they are connected to the readout computers via fiber optic cables.

[†] Canonical Ltd.; `https://www.ubuntu.com/`

[‡] Python Software Foundation; `https://www.python.org/`

[§] An upgrade to a newer version of Ubuntu is planned.

[¶] `https://git-scm.com/`

[‖] Wind River Systems Inc.; `https://www.windriver.com/`

[**] `https://www.freedesktop.org/wiki/Software/systemd/`
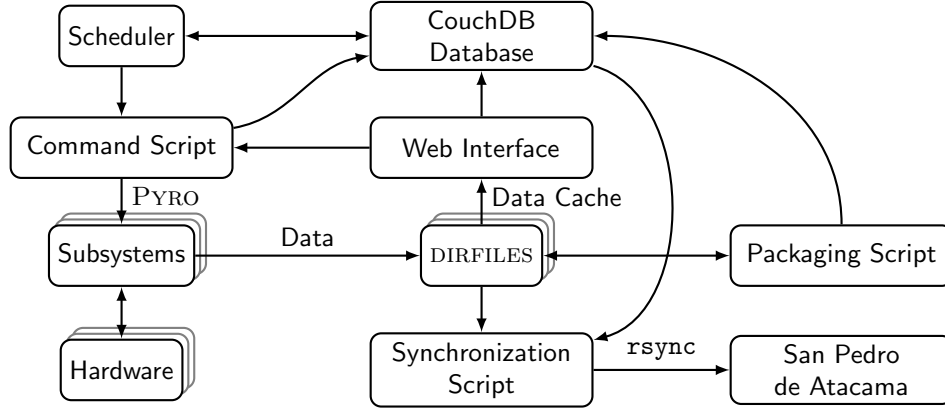
[††] `https://pyro4.readthedocs.io/`

Figure 2. Software structure overview. Boxes represent various components or locations, and arrows represent data or command flow. The software is structured around Python scripts, with a web interface used for user interaction and DIRFILES used for data recording.

they were local functions, with PYRO seamlessly taking care of all of the network communications. Thus, subsystems are controlled by calling the COMMAND SCRIPT, which in turn uses PYRO to call a function on the corresponding subsystem.

Figure 2 contains an overview of the software structure. All subsystems write data to the CENTRAL SERVER over the network, and a centralized database is used to store metadata. A SCHEDULER is used to execute preplanned observing routines, and a web interface is used to monitor system statuses and control the SCHEDULER. The standard data format is the DIRFILE standard for time-ordered data.[*] This filesystem-based, column-oriented format has been used for previous cosmology experiments, including the Atacama Cosmology Telescope[10] and the EBEX[11] balloon-borne experiment. It is supported by its reference GETDATA implementation,[12] which includes the `pygetdata` Python bindings, and can be easily visualized using the Kst plotting software.[†]

2.3 Scheduling

While a central COMMAND SCRIPT is sufficient, and preferable, for testing, its use for regular observations is highly inefficient and error-prone. Instead, a scheduling system is needed so an observation can be planned in advance. For a survey experiment like CLASS, once a reasonable observation strategy is found and tested, it will be used again and again. Therefore, a scheduling system that facilitates running such preplanned observations needs to be devised. One way to implement such a system would be to replace the COMMAND SCRIPT with a scheduler, calling the same PYRO functions as the command script. Although this method has the appeal of being simple and straightforward, it has the disadvantage of possibly producing different behavior than the COMMAND SCRIPT, which would lead to much confusion once a schedule is created using commands that were first tested by hand using the COMMAND SCRIPT. Therefore, the CLASS SCHEDULER calls the COMMAND SCRIPT, to ensure the exact same behavior.[‡] The results of the commands are logged to a database. A schedule entry is created using the arguments to the COMMAND SCRIPT, exactly what would be typed into the command terminal. These entries are combined with lines containing a simple time delta syntax, relative to either the schedule's start time or the last time directive, as well as optional comment lines. An example of the SCHEDULER syntax can be seen in Figure 3.

---

[*] `http://getdata.sourceforge.net/dirfile.html`

[†] KDE e.V.; `https://kst-plot.kde.org/`

[‡] More specifically, it calls a symbolic link to the COMMAND SCRIPT. The COMMAND SCRIPT uses this difference in the name of the script called to log the fact that the call was made by the SCHEDULER. There is also an additional mode that evaluates the command arguments but does not execute anything, which allows command syntax to be verified.

```
# This is a comment
MCEQ -autobias

# The following sets the commands after it to be run 2 minutes
# after the last time directive
/+2m
MCEQ -acqdata 50000000

# The following sets the commands after it to be run
# 2 days, 20 hours, 5 minutes, 30 seconds after the start time of the schedule
/2d20h5m30s
MCEQ -stop
```

Figure 3. Example of SCHEDULER syntax. Examples are shown of individual commands, comments, and relative and absolute time deltas.

2.4 Databases

While the science data are written to disk, it is helpful to keep various sorts of metadata in a database for easy access and manipulation. For CLASS, metadata about chunks of science data are stored in a database as well as a log of all commands executed and SCHEDULER data. Metadata for the science data collected are entered into a database to facilitate both data transfer and analysis. Therefore, a distributed database system is needed such that it can be accessed locally at the telescope site, in San Pedro de Atacama, and in North America. Three desirable properties for distributed computer systems, in this case a distributed database, are consistency, availability, and partition tolerance. However, Brewer's theorem states that only two of the three can be achieved by any one system.[13] Consistency requires all copies of the database to return identical results when queried, availability requires the database to always be accessible, and partition tolerance requires the system to still function even if the network link between the different copies is severed. For example, if a database system enforces both partition tolerance and consistency, the database will be read-only, and thus only partially accessible, if the network link goes down, since the link is needed to keep the different copies consistent. For CLASS, availability and partition tolerance are the most important properties, since both the radio link to the telescope site and the internet connection to North America might be unreliable, but this should not stop data collection, which also involves adding metadata to the database. Although this could in general lead to merging problems when the database system tries to restore consistency once a severed network link is reestablished, the software running at CLASS's three locations modify different database fields, so no conflicts will arise. Most data are entered at the telescope site, with the software at the two auxiliary sites only modifying fields pertaining to their local data storage locations.

To this end, the CouchDB database* was chosen for handling metadata, with a server at each location running a database server instance that hosts a full copy of the databases to allow for fast, low-latency database queries. CouchDB is a NoSQL database that stores data as JSON documents and is highly available and partition tolerant, with eventual consistency. NoSQL databases differ from traditional SQL databases by not requiring a fixed, predefined table structure; instead, each document can have any data keys it needs, functioning more like a dictionary than a spreadsheet. Instead of using structured queries to retrieve data, mapping and reduction functions are written and used; in the CouchDB parlance, these are known as "views" and are written in JavaScript.

The CouchDB instance hosts databases to store commands executed, schedules, the results of commands executed for schedules, and metadata about the science data products. The COMMANDS DATABASE records commands executed by the COMMAND SCRIPT along with a timestamp and Git revision hashes for both the COMMAND SCRIPT and the target script; this logging happens regardless of how the COMMAND SCRIPT was

---

* Apache Software Foundation; `https://couchdb.apache.org/`

called—directly or by the SCHEDULER. Each commit in a Git repository has a unique 160-bit SHA-1 hash[14] associated with it; since all of the site software is contained in a Git repository, recording these revision hashes uniquely identifies the exact version of the scripts that were executed, for future reference. The SCHEDULES DATABASE contains schedules for use by the SCHEDULER. Each entry contains the actual schedule entered, a parsed copy of the schedule for use by the SCHEDULER, an ID number, descriptive tags, the schedule's author, a start time, and a status field. Once the schedule finishes running, or fails, an end time is added. Failed schedules also record the line that failed. These entries are created by the SCHEDULER's web interface and are modified by it and the actual SCHEDULER. The SCHEDULER also records the results of commands executed in a JOBS DATABASE; each entry contains the command executed, a timestamp, the command's output, and if it failed. The final database contains metadata for each packaged chunk of CLASS science data. In addition to this standard metadata, which are also kept with the data packages, the database stores the location of the data packages at each site: Cerro Toco, San Pedro de Atacama, and North America. Each site only modifies the location field for itself, so synchronization issues after a network outage are avoided. These location fields are used to facilitate data transfer and analysis.

3. HARDWARE INTERFACES

To capture data, a physical instrument is required. To record these data and to control these instruments and other hardware, computer interfaces to the hardware are required, as is software to use these hardware interfaces. Although the hardware of different CLASS subsystems differ considerably, their software interfaces were designed to be similar, using a Python script that interfaces either directly with the hardware or through vendor-provided software to expose a PYRO control interface. Where custom electronics are used, such as for cryogenic diode-based thermometry, warm thermometry in the telescopes' optics cages, and for control of the telescopes' VPMs, these are designed around an Ethernet interface. While TCP is used for sending commands, UDP is used to stream data in a manner that is robust to connectivity interruptions and equipment restarts. For equipment that uses an RS-232 serial interface, such as AC resistance bridges (used for reading out ruthenium oxide cryogenic thermometers), magnetometers, the diesel tank level sensor, and the site weather station, Ethernet-to-serial converters are used to allow access over the site network; this simplifies operations, since the control scripts for these equipment can be run from the CENTRAL SERVER.* Along with scientific instruments, other site infrastructure, such as the generators, can also be controlled remotely.

The only exceptions to this general control architecture are the cryostats. They are controlled using a proprietary GUI software provided by their manufacturer, Bluefors,† which only runs under the Windows operating system and does not provide much in the way of a scriptable interface. Therefore, the cryostats are operated manually, with the REMOTE DESKTOP PROTOCOL used to access the GUI. However, as the cryostats only need to be controlled when they are being cooled down or warmed up for maintenance, not during normal operation, this lack of integration with the rest of the site operations software does not impede observations.

3.1 Detectors

The telescopes' transition edge sensor detectors[15, 16] are read out via time-division multiplexing using Multi-Channel Electronics (MCE) units[17] developed at the University of British Columbia.‡ There is one MCE per receiver and currently one or two receivers per telescope mount, although the two mounts will eventually have two receivers each. The detectors are connected to the MCE using a set of SQUID multiplexers at the focal plane and a SQUID amplifier series array at the 4 K stage of the cryostat. The MCE is triggered using a FRAME PULSE, delivered with a clock signal over fiber optics from a SYNC BOX, one per telescope mount, which is a device used to synchronize data collection by multiple MCEs and the telescope mount. The FRAME PULSE assigns a 32-bit ID number to the readout frame, which consists of a full focal plane readout, and

---

* A USB-to-serial interface is used instead for the weather station, since it is mounted on the outside of the control room, a short distance from the CENTRAL SERVER.

† Bluefors Oy; https://bluefors.com/

‡ https://e-mode.phas.ubc.ca/mcewiki/index.php/Main_Page

delivers pulses to trigger reading out each detector. The FRAME PULSE is delivered at slightly over 200 Hz.* The MCE, mounted on the cryostat, communicates with a HOST COMPUTER in the CONTROL ROOM via fiber optics and a PCI readout card. This HOST COMPUTER uses MCE ACQUISITION SOFTWARE (MAS) SCRIPTS, provided by the University of British Columbia, to control the MCE and record data. A PYRO interface allows use of these scripts over the network and also provides a routine for measuring I–V response and applying appropriate detector biasing. Data are written to an NFS mount on the CENTRAL SERVER.

3.2 Mount

Each telescope mount is controlled by an industrial MOUNT COMPUTER running the VxWorks RTOS, which loads its software over the network at boot time. The MOUNT COMPUTER, also known as the antenna control unit (ACU), controls the mount's four servo motors, two for azimuth and one each for elevation and boresight angle, over an isolated Ethernet network and also reads in various encoders and tiltmeters. The MOUNT COMPUTER also receives FRAME PULSE information from the SYNC BOX over RS-485 (instead of the fiber optics used for the MCEs) for synchronization with the MCE units and has a GPS receiver to obtain precise timing information; as the MCEs only record FRAME PULSE information and not time, the mount's timing information is also used to assign times to detector data when the data are packaged. The mount's position is determined using encoders that feed a pointing model, and these data, as well as timing and status information, are written to an NFS mount on the CENTRAL SERVER. The pointing model is derived from Moon and planet observations,[18] although the initial pointing model was constructed using star camera observations processed using the Astrometry.net blind astrometric solver.[19]† The MOUNT COMPUTER exposes a Telnet interface and a TCP interface to the site network. The Telnet interface allows for interactive control via a terminal and status monitoring, while the TCP interface is used for scripted control of the mount. A PYRO interface running on the CENTRAL SERVER translates this TCP interface into a standard PYRO interface for the COMMAND SCRIPT.

The mount monitor and control system, which is written in C++, implements the programs used for scanning in azimuth, with sun avoidance to preclude the boresight from coming within 20 deg of the Sun; sky dips, for atmospheric analysis; and drift scans of planets and the Moon, which are used for pointing and beam analysis. The JPL DE430 ephemerides[20] are used by the monitor and control system for tracking the Moon, Sun, and planets. Parameters used by these programs are passed to the monitor and control system via the PYRO interface.

3.3 Variable-delay polarization modulator

The VPM consists of a fixed wire array and a movable mirror, to modulate the incoming polarization signal. The mirror is driven by a closed-loop motion control system. This system consists of voice coils to actuate the mirror and encoders to measure its position. The encoder signals are duplicated, with one copy going to the VPM CONTROLLER and another copy going to the ACU. The VPM CONTROLLER, built around an industrial applications processor and running a network-booted RTOS, uses this information to maintain proper mirror position, while the ACU logs the encoder data. This duplication is necessary because the VPM CONTROLLER is not connected to the SYNC BOX, and the VPM encoder positions need to be recorded synchronously with the detector data. A control script running on the CENTRAL SERVER communicates with and configures this CONTROLLER via TCP over an Ethernet connection. Finally, a PYRO interface exposes a standard network interface for VPM control.

---

* Running slightly fast is preferable to running slightly slow, since it ensures the recorded data chunks are less than ten minutes in length. This simplifies the data packaging process, since it guarantees that a data chunk only needs to be split once.

† The star camera consists of a networked machine vision camera in a waterproof enclosure that is rigidly attached to the telescope mount.

4. DATA PIPELINE

In order to measure the polarization of the CMB, data need to be acquired from the telescopes' detectors and various housekeeping systems, combined, packaged, and transferred to North America for analysis. This packaging needs to happen in real time, to prevent a backlog and maximize observing time; furthermore, the process should be lossless and verifiable to prevent data loss and identify potential data corruption.

4.1 Acquisition

Data acquisition starts with scheduling an observation and acquiring data from hardware instruments. As data are acquired, they are written as DIRFILES on the CENTRAL SERVER, either directly or using NFS mounts. These are structured with a directory for each computer acquiring data, with subdirectories for each acquisition system. Individual DIRFILES are named using a timestamp formatted as `%Y-%m-%d-%H-%M-%S`, e.g., `2020-03-22-20-00-00`. DIRFILES consist of a directory that contains a plain-text FORMAT FILE and separate binary files for each stream of time-ordered data. Additional derived fields can be defined as linear combinations of existing fields, e.g., to scale raw data to SI units; aliases, e.g., to label the location of thermometers without having to edit the acquisition code; and extracted bit-fields, e.g., to combine multiple status flags in one raw data field.

Data are divided into ten-minute chunks as they are acquired. For asynchronous housekeeping data, which are not acquired using the SYNC BOX, a new chunk is started at clock-aligned intervals, e.g., at 00:00, 00:10, 00:20, etc., so the first data chunk in the series will be less than ten minutes in length, e.g., six minutes if data collection started at twenty-four minutes past the hour. These divisions are done when the time in seconds since the Unix Epoch* modulo 600 seconds rolls over. Synchronous data are acquired in arbitrarily-aligned chunks of approximately ten minutes in length.† All timestamps are in Universal Time, with asynchronous data using the computer's NTP-synchronized system clock in UTC and the mount using UT1, derived from GPS time; the detector data are not acquired with timestamps, but they are synchronized with the mount data using the SYNC BOX. The time is stored in the DIRFILES as seconds since the Epoch.

4.2 Packaging

Once data are acquired, they must be packaged into a standard data product. The standard CLASS data product is a clock-aligned ten-minute DATA CHUNK consisting of time-ordered data, with one DIRFILE per mount for synchronous data and individual DIRFILES for each asynchronous data acquisition system. Synchronous data are combined in this data product, since this can be done without altering any data or losing any information. Since the DIRFILE format keeps each data field in a separate file and the GETDATA library does not load the data until a specific field is read, this combining of data from multiple receivers does not add any overhead when only one receiver's data need to be read. Asynchronous data are kept separate due to the losses inherent to resampling and interpolation. This is a trade-off between making things simpler during analysis and preventing data loss, since interpolation eventually needs to be done. A packaging script is run every ten minutes, at 00:02, 00:12, 00:22, etc. It processes data from two data periods prior and before, e.g., the 00:32 run will only process data from the 00:10 period and before; this ensures that the non-ten-minute-boundary-aligned synchronous data collection for the data period has finished. For example, synchronous data collection might start at 00:19, which would include data in the 00:10 period; this collection would not end until 00:29, unlike the asynchronous data collection, which would start a new data chunk at 00:20. Therefore, the maximum delay, plus a few minutes of buffer, is used.

4.2.1 Synchronous

The detector and ACU data sampling on a given mount is synchronized, but the two mounts are not synchronized with each other. An overview of the synchronous data packaging process can be seen in Figure 4. The first step in packaging synchronous data is to divide them into CLASS's standard ten-minute data intervals. This process starts with dividing each data chunk collected by the ACU into two parts, A and B,

---

* `1970-01-01T00:00:00Z`

† The chunks are 120 000 frame pulses long, or just under ten minutes, "redefining" the second as 200 FRAME PULSES.

where seconds since the Epoch modulo 600 seconds rolls over; the ACU data must be the first processed, since they are the only synchronous data that contain timestamps, instead of just FRAME PULSE numbers. These divided chunks are then combined to form ten-minute chunks aligned with the Epoch, with zero-padding added before, after, and between chunks when needed to form a full ten-minute chunk. The field containing the FRAME PULSE number is filled in with the appropriate values instead of zero-padding like the rest of the fields; this allows for easier synchronization with the detector data. A status field is also added to the resulting DIRFILE that shows where the ACU data are valid, as opposed to where they are zero-padded, stored in an 8-bit integer field.

Next, the detector data from each MCE unit on the mount are divided using the FRAME PULSE numbers recorded in both the MCE data and the now aligned ACU data. As with the ACU data, these chunks are then combined to form aligned ten-minute chunks, again with zero-padding added before, after, and between chunks as needed; again, a status field is added, but different bits in the 8-bit integer field are used. Care must be taken, since the FRAME PULSE counter occasionally rolls over.[*] Finally, the raw data files in each of the MCE DIRFILES are moved into subdirectories in the ACU DIRFILE. The ACU DIRFILE FORMAT FILE is altered to include the MCE DIRFILE FORMAT FILES as child fragments, which prefix the MCE DIRFILE field names with the appropriate MCE name, and the data valid status fields are combined using a bitwise OR. After this process is complete, the resulting DIRFILE can be treated as an asynchronous data chunk in the next step of the packaging process.

The merged data are then compressed, with `gzip` compression used for the mostly floating-point ACU data and FLAC compression[†] used for the fixed-precision MCE data. As FLAC was designed as an audio compression codec, it does not support the 32-bit data recorded by the MCE, but GETDATA avoids this limitation by splitting each 32-bit integer into two 16-bit integers and compressing the data as two separate channels. However, this is still not optimal, since FLAC will treat these integers as smaller than 16 bits if the most significant bits are never used. CLASS operates the MCEs in DATA MODE 10, which uses the lower 25 bits of the 32-bit integer to store low-pass-filtered detector data as a signed integer and the upper 7 bits to store a flux jump counter as another signed integer. Thus, when the flux jump counter is zero or positive, the upper half of the 32-bit integer is treated as having fewer than 16 bits, often as having as few as 9–10 bits, but is treated as having the full 16 bits when the flux jump counter is negative. Therefore, the data packaging script removes the 7-bit flux jump counter from the combined data stream and converts it to a separate 8-bit signed integer field, which is FLAC-compressed. The filtered detector data that remain in the original 32-bit field is then treated as a 25-bit integer when FLAC compression is applied. This scheme performs well on the CLASS data, reducing disk space usage while still providing rapid decompression and data access.

4.2.2 Asynchronous

Asynchronous data are recorded to timestamped DIRFILES in a directory structure based on the computer recording the data. However, the final data product structure starts with a timestamped directory, with subdirectories for different types of data, a hierarchy divorced from the computer systems used to collect the data. Figure 5 contains an overview of this structure. Therefore, the first step of the asynchronous data packaging process, which includes the merged synchronous data, rearranges the asynchronous DIRFILES from their original structure to the final data product structure, in a new location. As the vendor-provided software for operating the cryostats only records CSV files, these data are converted to DIRFILES at this step. While the DIRFILES are originally recorded uncompressed, at this stage they are compressed using the `gzip` compression format to save space.[‡] Images from each of the telescope site's monitoring cameras are also packaged, since this provides a method of checking for bad weather or unusual site activity should artifacts be found in the telescope data during analysis.

---

[*] As the FRAME PULSE is a 32 bit unsigned integer, this happens roughly every eight months. This counter also resets any time the SYNC BOX is power-cycled.
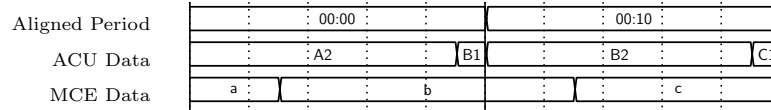
[†] Xiph.Org Foundation; `https://xiph.org/flac/`

[‡] FLAC compression is not used for the asynchronous data, since many of the data fields involve floating point numbers, which FLAC compression does not support.
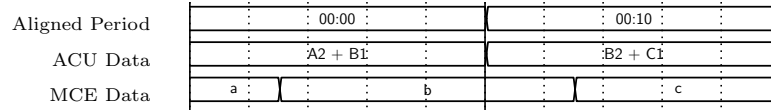
The process starts with misaligned synchronous data.
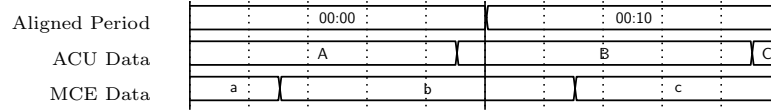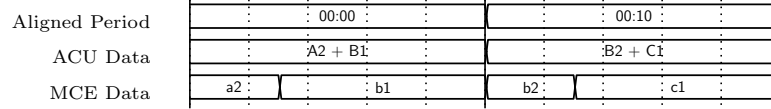

ACU data are then split at the proper alignment based on timestamps...

...and then joined to form aligned chunks.

MCE data are then split at the proper alignment based on FRAME PULSE numbers in the ACU data...
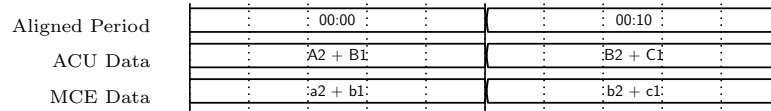

...and then joined to form aligned chunks.

Figure 4. Overview of synchronous data packaging process. The procedure by which ACU (mount) and MCE (detector) data are aligned and joined is shown.

Next, the DIRFILES are converted to uncompressed ZIP file* archives. An uncompressed ZIP file concatenates all of the files it contains together and then includes a file offset table that allows for its contents to be located and read in a manner that allows for random reads. Encapsulating the DIRFILE into a ZIP archive reduces the number of files in the data package by more than an order of magnitude, which provides significant speed improvements for data transfer and backup operations, due to the small size of many of the files contained in the unencapsulated DIRFILES.† Finally, a JSON metadata file is created that contains basic information about the DATA CHUNK, including the files included, their sizes, and SHA-1 checksums, for later verification; this file is saved in the root of the DATA CHUNK. The location of the DATA CHUNK on disk is then appended to the metadata information, and this information is saved to the database.

4.3 Transfer

Once the data are packaged, they need to be transferred off the mountain to San Pedro de Atacama and then to North America. Data integrity needs to be verified after each transfer, and old data eventually need to be deleted from the CENTRAL SERVER to free up disk space. Data are transferred from the telescope site's CENTRAL SERVER to the ANALYSIS MACHINE in San Pedro de Atacama via the CLASS network's radio link. A synchronization script runs on the site's CENTRAL SERVER every ten minutes, which transfers the data files using rsync.‡ This script uses a lock file to ensure only one copy of itself is running at a time, and a certificate is used to authenticate with the ANALYSIS MACHINE. A verification script is also run every ten minutes on the ANALYSIS MACHINE in San Pedro de Atacama. This script checks each unverified DATA CHUNK on the machine against the local copy of the database and sees if the DATA CHUNK has a local path assigned to it. If it does not have a local path, the DATA CHUNK is verified against the database, and the CHUNK's local path is added to the database. If the verification fails, an error is thrown, and the local path is not added to the database, so it is treated as if the data were never transferred. The corrupted data might be fixed by rsync with the next synchronization run—else, manual intervention is required to delete the corrupted data. If the DATA CHUNK already has a local path, the data have already been verified, so the CHUNK is ignored. Since the synchronization script uses rsync's delayed updates feature, files do not appear until the transfer is complete, so issues with incomplete transfers are mitigated. This process is repeated to transfer data from the ANALYSIS MACHINE in San Pedro de Atacama to a data server located in Baltimore.

4.3.1 Archiving of data and deletion of old data

While a permanent copy of the data will be kept on the file server in Baltimore, the data will not be permanently kept on disk on-site or in San Pedro de Atacama, as disk space will need to be freed for new observations. On both the CENTRAL SERVER on Cerro Toco and the ANALYSIS MACHINE in San Pedro de Atacama, a script is run on a daily basis to free up space by removing old DATA CHUNKS that have already been transferred and verified, to ensure a minimum amount of free disk space; as long as enough space is free, data will not be deleted. When disk space needs to be freed, the database will be checked for DATA CHUNKS that have already been transferred to San Pedro de Atacama in the case of the telescope site's CENTRAL SERVER and transferred to North America in the case of the ANALYSIS MACHINE. Of these DATA CHUNKS, data will be deleted in chronological order, with the oldest data first, until enough free disk space is available; currently, this threshold is set at 75% of the disk capacity. If this process is not able to free enough space, an error will be thrown, and manual intervention will be required. The only existing copy of a DATA CHUNK will never be automatically deleted.

---

* ISO/IEC 21320-1:2015

† Unfortunately, this extended DIRFILE functionality is only available as a patch, since the GETDATA library seems to be abandoned. The author of this manuscript (M.A.P.) has not received a response from the GETDATA maintainer (D.V.W.) with either postings to the getdata-devel@lists.sourceforge.net mailing list or via private correspondence in more than two years, nor has there been any public activity on the project in that time frame.

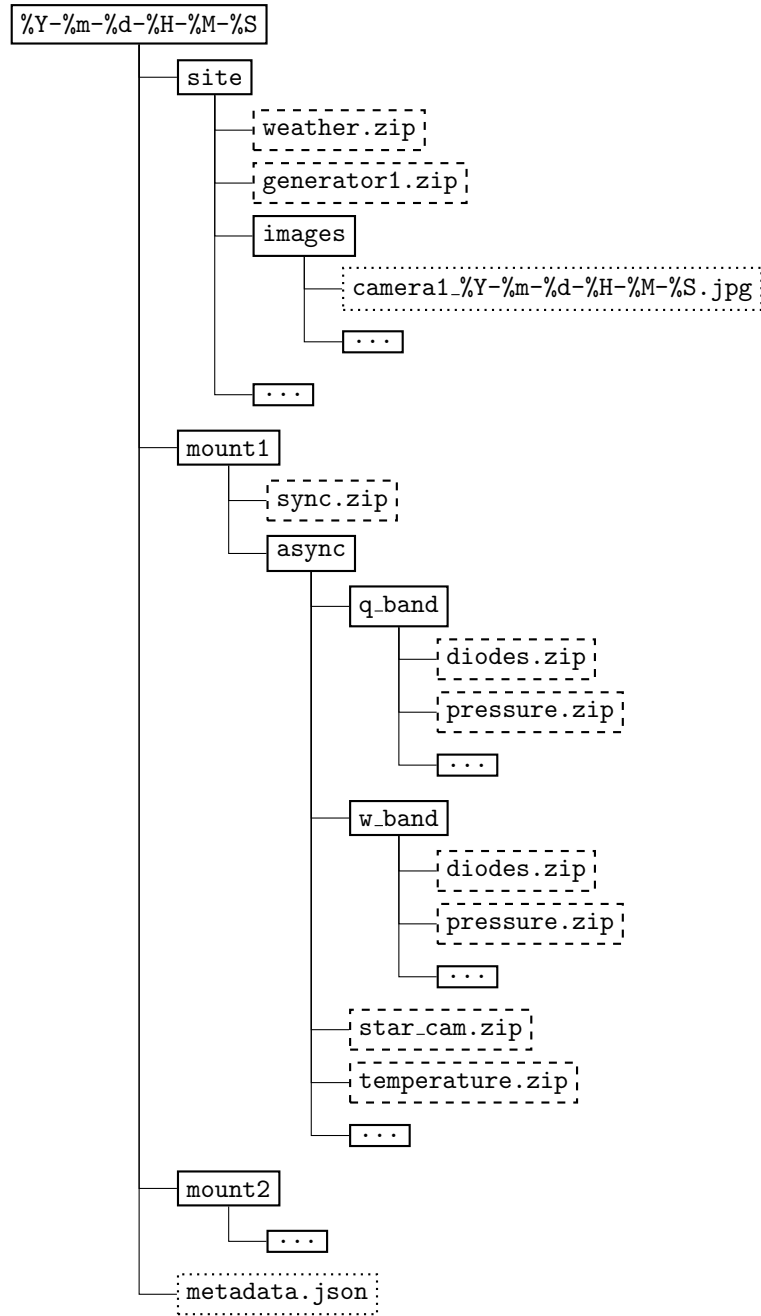‡ https://rsync.samba.org/

```
%Y-%m-%d-%H-%M-%S
├── site
│   ├── weather.zip
│   ├── generator1.zip
│   ├── images
│   │   ├── camera1_%Y-%m-%d-%H-%M-%S.jpg
│   │   └── ...
│   └── ...
├── mount1
│   ├── sync.zip
│   └── async
│       ├── q_band
│       │   ├── diodes.zip
│       │   ├── pressure.zip
│       │   └── ...
│       ├── w_band
│       │   ├── diodes.zip
│       │   ├── pressure.zip
│       │   └── ...
│       ├── star_cam.zip
│       ├── temperature.zip
│       └── ...
├── mount2
│   └── ...
└── metadata.json
```

Figure 5. Overview of the CLASS data product structure. Solid boxes represent directories, dashed boxes represent DIRFILES, and dotted boxes represent other files. Only a subset of the DIRFILES and images are shown. The subdirectories in the `async` directory contain asynchronous data associated with a specific receiver, while the `sync.zip` DIRFILE combines ACU data for a given mount with detector data from both of its receivers.

4.3.2 Ensuring data integrity

A number of steps are taken throughout the data packaging and transfer process to ensure data integrity. At each step in the process, data are not deleted until after the next step in the process has finished. Updating the database with a DATA CHUNK's location at a new location is always the last step in the process, to avoid the possibility of the database showing that the data are there when they are not actually there; if something goes wrong in the transfer process, some or all of the data might be there, but the database will be conservative and list the data as not being present.

As the last step in the packaging procedure, SHA-1 checksums are created for each file and stored in the database. These checksums are unique, but repeatable, 160-bit cryptographic hashes based on the data; if even one bit in the source file changes, a completely different hash will be generated, so transfer errors of either the data or the checksums are easily detected. After each transfer step, new checksums are generated for each file, and these checksums are compared to the copies stored in the database. If the checksums match, the transfer was successful. If they don't match, the transfer failed, and data corruption occurred; in this case, the data are treated as if they were not transferred at all, and an error is thrown. Along with being replicated across multiple locations, regular offline backups are made of the CLASS CouchDB database that stores the science data product metadata.

The primary CLASS data server located in Baltimore utilizes a ZFS file system.* ZFS provides strong data integrity guarantees, including against silent data corruption, by making extensive use of checksums and through regular integrity checking. A `RAIDZ2` configuration is used to protect against disk failures, and regular file system snapshots are taken to protect against inadvertent data deletion, caused either by human error or malware. Nightly backups are made to a replica data server, located in a different location, to provide additional protection. These backups rely on ZFS's ability to efficiently send and receive file system snapshots and are configured in such a way that even if one of the two servers is compromised by a malicious actor, it would be difficult to erase both copies of the data. Finally, the data packages are additionally backed up to a cloud storage provider. As of November 2020, CLASS has 16 TB of compressed data packages recorded to disk.

5. WEB INTERFACE

A web interface is run on the CLASS site's CENTRAL SERVER to provide status information, facilitate execution of commands and scheduling of observations, and provide a wiki to store site operations information and documentation. Except for the wiki, which uses the MediaWiki software package,† the interface runs using Python and the Django web framework‡ and uses the Bootstrap§ front-end framework.

The status display provides pertinent status information about the telescope systems, including cryostat temperatures, pressures, and flows; available disk space; and site environmental conditions. A screenshot of the status display overview section can be seen in Figure 6, which provides a color-coded dashboard for quickly evaluating the current instrument health as well as sparklines[21] for evaluating how parameters have changed over the past day. In addition to the overview, detailed pages are available for each cryostat and mount. Here, data are graphed client-side using a JavaScript charting library; multiple time ranges can be selected, and real-time updates are pushed using a WebSockets connection for the shortest time range. A script continuously reads in housekeeping data from the currently recording DIRFILES and stores it in a day-long circular cache; this cache is flushed to disk every ten minutes, so it will still contain data if the caching script needs to be restarted. This caching script exposes a PYRO interface, which the web interface back-end uses to access the data. The script additionally triggers the pushing of the real-time updates over the WebSockets connection. Due to the packaging process, only the currently recording DIRFILE is accessible, necessitating the cache, which is also useful to improve performance. Although the web interface back-end
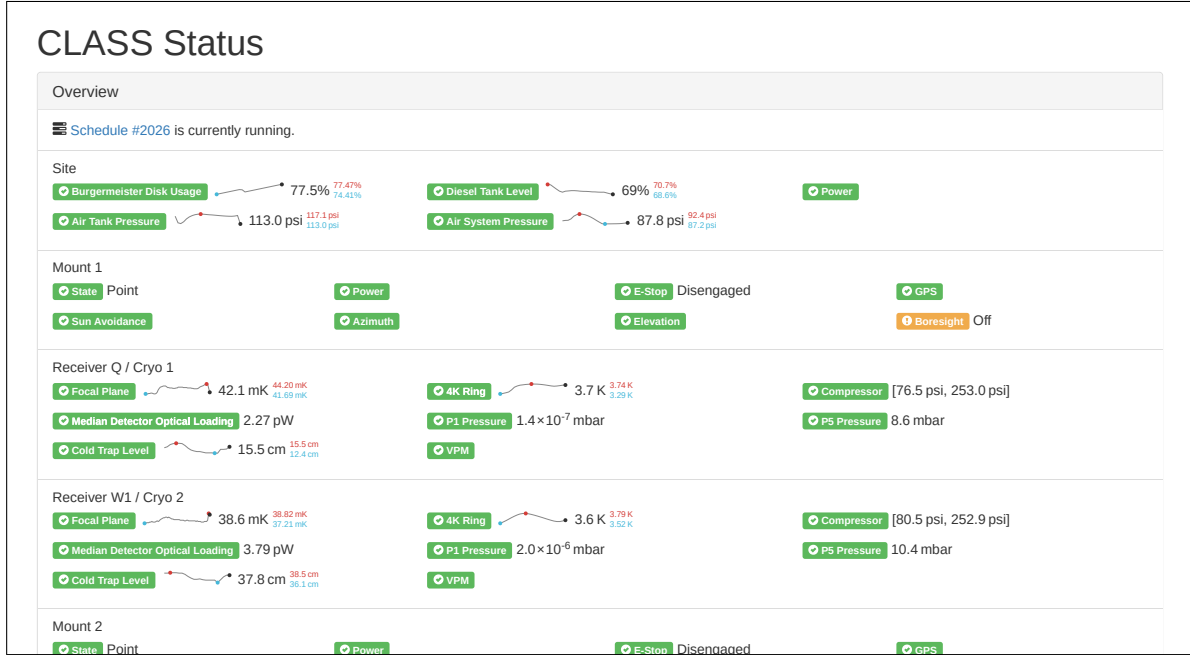

Figure 6. Web status interface. This screenshot shows an example view from the web status interface overview section, which allows one to quickly evaluate instrument health.

could directly cache the data, this is not done, since the web server runs multiple, simultaneous copies of the back-end to maintain responsiveness; this would cause duplicate disk access and, more importantly, cause duplicate real-time updates to be pushed. In addition to providing information for the status display, the caching script also sends alerts[*] for certain failures that require immediate attention, such as a cryostat warming up, although in some cases alerts are directly triggered by the corresponding hardware interface script instead. The status display also provides access to results from automated preliminary analysis scripts, which are run every morning on the ANALYSIS MACHINE in San Pedro de Atacama, and to current views from the various site observation cameras, either as static images updated once per minute or as live video.

The SCHEDULER interface provides an easy-to-use, graphical method for scheduling observations on the telescopes. Login credentials are required to access and use the SCHEDULER interface. The main screen shows the currently running schedule if one is running, the next pending schedule if no schedule is currently running but one is pending, or the last schedule run if none are currently running or pending. Figure 7 shows this interface. A sidebar provides a list of recent schedules and their status information, with a link to a complete history of schedules. When viewed, each schedule can be displayed either in the raw form it was entered in, with basic syntax highlighting, or in a parsed form. Status, start time, title, author, schedule number, and tagging information are also displayed. If a schedule completed, the end time is shown. If a schedule failed to complete, the failure time and failed command are displayed. Pending schedules can be edited, and pending or running schedules can be canceled. An interface for adding new schedules is also accessible from the main screen. This prompts a user to enter metadata including a title, start time, and, optionally, tags; a date and time picker is provided to assist the user with start time entry. The author field is automatically filled in with the user's name, but can also be edited.[†] Then comes a box for entering the schedule; basic syntax highlighting is provided. Any time in the editing process, the user can switch between the edit box and a verification tab, which verifies the syntax of the entered schedule; lines containing incorrect syntax are

---

[*] These are currently sent to a channel in the *Slack* chat tool used by the CLASS collaboration, although the destination of the alerts can be easily changed.

[†] The username of the user that entered the schedule is also recorded to the database as a separate field but is not displayed.
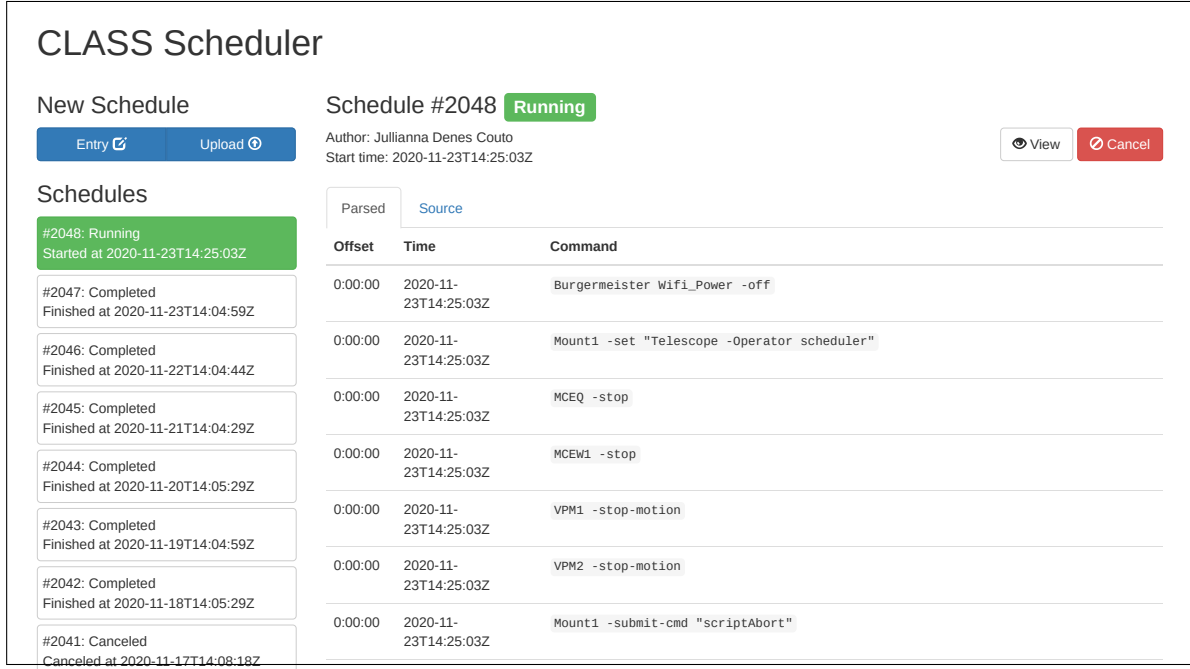
Figure 7. Web scheduler interface main screen. The main screen of the web scheduler interface is shown in this screenshot, displaying the currently running schedule.

highlighted. Once finished, the schedule can be submitted and will be run by the SCHEDULER. Previously-run schedules also provide a button for opening the new schedule interface with its fields pre-filled with data from the previous schedule. This interface for adding schedules can be seen in Figure 8. A separate interface is also provided for executing, and displaying the results of, commands outside the context of a schedule. This interface is normally locked out while a schedule is running to prevent accidental execution of commands, but it can be unlocked if necessary. Additionally, it displays a list of recently run commands and shows in real-time which other users have the command interface open, to encourage communication.

6. LESSONS LEARNED

Some parts of the architecture described above were different when the first CLASS telescope was deployed in 2016, and other parts should have been done differently in hindsight or would be done differently without certain hardware interface restrictions. Other changes have been incremental updates, such as transitioning from Python 2 to Python 3.* One lesson that was learned is that interfacing with instruments over Ethernet with a control script running on the CENTRAL SERVER is easier and more reliable than running a control script on another computer and writing data over NFS, since the former is a "pull" configuration, while the latter is a "push" configuration; "pull" configurations are less likely to run into issues if network connectivity is temporarily lost or if the CENTRAL SERVER needs to be rebooted. With this insight, thermometry data recording was transitioned from using USB interfaces to using Ethernet interfaces, which improved reliability. Ideally, this transition would be extended to its logical conclusion, using a single server to operate the telescopes and all the instruments. Unfortunately, limitations with the MCE interfaces and cryostat interfaces prevent this. MCE interface limitations additionally hamper improvements to data acquisition synchronization. Without these limitations, the detector, pointing, and VPM encoder data acquisition hardware could be configured to receive time information over the network via the precision time protocol standard, which allows for sub-microsecond time distribution. Data frames could then be synchronized with the start of GPS

---

* Although Python 3 was fairly well established when software development started in 2014, Python 2 was originally used, since, at the time, the GETDATA library did not yet support Python 3.

15

Figure 8. Web scheduler new schedule interface. This screenshot shows a new schedule being written; metadata can be added, the start time can be adjusted, and the schedule can be validated before it is submitted to be run.

time seconds, as long as the data acquisition rate is limited to an integer number of samples per second, allowing for the elimination of the dedicated synchronization hardware that is currently used. Another lesson learned is that a large number of small files makes data transfer and backup operations much slower than they would otherwise be. This led to a transition from standard DIRFILES to DIRFILES encapsulated in ZIP files, which reduced the number of files stored on disk by more than an order of magnitude and significant performance improvements for data transfer and backup operations. In hindsight, it would have also likely been better if the data packages were based on hour-long chunks instead of ten-minute chunks for the same reason, but the current data package chunking scheme is too ingrained to change now for what would be a more incremental improvement.

7. CONCLUSIONS

The current CLASS control and systems software architecture has been described. This architecture is based around Python scripts, with DIRFILES used to record data and the PYRO library used for network-based control interfaces. Data are packaged into ten-minute long chunks, which are aligned with the start of the hour. Scheduling and status monitoring are performed using a web interface. Throughout, there is a focus on ensuring data integrity. It is our hope that our documenting of the CLASS software architecture here can help inform software development for future experiments. The CLASS software architecture has been successful at fulfilling its intended purpose and has proven to be sufficiently reliable.

ACKNOWLEDGMENTS

16

Espinoza, Saianeesh Haridas, Connor Henley, Yunyang Li, Lindsay Lowry, Isu Ravi, Gary Rhodes, Daniel Swartz, Bingie Wang, Qinan Wang, Tiffany Wei, and Ziang Yan. We acknowledge productive collaboration with Dean Carpenter and the JHU Physical Sciences Machine Shop team. We thank María José Amaral, Chantal Boisvert, William Deysher, and Miguel Angel Díaz for logistical support. We further acknowledge the very generous support of Jim and Heather Murren (JHU A&S '88), Matthew Polk (JHU A&S Physics BS '71), David Nicholson, and Michael Bloomberg (JHU Engineering '64). CLASS is located in the Parque Astronómico Atacama in northern Chile under the auspices of the Agencia Nacional de Investigación y Desarrollo (ANID). The software described in this manuscript makes extensive use of NumPy.[22]

REFERENCES

[1] Essinger-Hileman, T., Ali, A., Amiri, M., Appel, J. W., Araujo, D., Bennett, C. L., Boone, F., Chan, M., Cho, H.-M., Chuss, D. T., Colazo, F., Crowe, E., Denis, K., Dünner, R., Eimer, J., Gothe, D., Halpern, M., Harrington, K., Hilton, G. C., Hinshaw, G. F., Huang, C., Irwin, K., Jones, G., Karakla, J., Kogut, A. J., Larson, D., Limon, M., Lowry, L., Marriage, T., Mehrle, N., Miller, A. D., Miller, N., Moseley, S. H., Novak, G., Reintsema, C., Rostem, K., Stevenson, T., Towner, D., U-Yen, K., Wagner, E., Watts, D., Wollack, E. J., Xu, Z., and Zeng, L., "CLASS: the cosmology large angular scale surveyor," in [*Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VII*], Holland, W. S. and Zmuidzinas, J., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **9153**, 91531I (July 2014).

[2] Harrington, K., Marriage, T., Ali, A., Appel, J. W., Bennett, C. L., Boone, F., Brewer, M., Chan, M., Chuss, D. T., Colazo, F., Dahal, S., Denis, K., Dünner, R., Eimer, J., Essinger-Hileman, T., Fluxa, P., Halpern, M., Hilton, G., Hinshaw, G. F., Hubmayr, J., Iuliano, J., Karakla, J., McMahon, J., Miller, N. T., Moseley, S. H., Palma, G., Parker, L., Petroff, M., Pradenas, B., Rostem, K., Sagliocca, M., Valle, D., Watts, D., Wollack, E., Xu, Z., and Zeng, L., "The Cosmology Large Angular Scale Surveyor," in [*Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VIII*], Holland, W. S. and Zmuidzinas, J., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **9914**, 99141K (July 2016).

[3] Kamionkowski, M., Kosowsky, A., and Stebbins, A., "Statistics of cosmic microwave background polarization," *Physical Review D* **55**, 7368–7388 (June 1997).

[4] Watts, D. J., Larson, D., Marriage, T. A., Abitbol, M. H., Appel, J. W., Bennett, C. L., Chuss, D. T., Eimer, J. R., Essinger-Hileman, T., Miller, N. J., Rostem, K., and Wollack, E. J., "Measuring the Largest Angular Scale CMB B-mode Polarization with Galactic Foregrounds on a Cut Sky," *The Astrophysical Journal* **814**, 103 (Dec 2015).

[5] Watts, D. J., Wang, B., Ali, A., Appel, J. W., Bennett, C. L., Chuss, D. T., Dahal, S., Eimer, J. R., Essinger-Hileman, T., Harrington, K., Hinshaw, G., Iuliano, J., Marriage, T. A., Miller, N. J., Padilla, I. L., Parker, L., Petroff, M., Rostem, K., Wollack, E. J., and Xu, Z., "A Projected Estimate of the Reionization Optical Depth Using the CLASS Experiment's Sample Variance Limited E-mode Measurement," *The Astrophysical Journal* **863**, 121 (Aug 2018).

[6] Guth, A. H., "Inflationary universe: A possible solution to the horizon and flatness problems," *Physical Review D* **23**, 347–356 (Jan. 1981).

[7] Hu, W. and White, M., "A CMB polarization primer," *New Astronomy* **2**, 323–344 (Oct. 1997).

[8] Chuss, D. T., Wollack, E. J., Henry, R., Hui, H., Juarez, A. J., Krejny, M., Moseley, S. H., and Novak, G., "Properties of a variable-delay polarization modulator," *Applied Optics* **51**, 197 (Jan. 2012).

[9] Harrington, K., Eimer, J., Chuss, D. T., Petroff, M., Cleary, J., DeGeorge, M., Grunberg, T. W., Ali, A., Appel, J. W., Bennett, C. L., Brewer, M., Bustos, R., Chan, M., Couto, J., Dahal, S., Denis, K., Dünner, R., Essinger-Hileman, T., Fluxa, P., Halpern, M., Hilton, G., Hinshaw, G. F., Hubmayr, J., Iuliano, J., Karakla, J., Marriage, T., McMahon, J., Miller, N. J., Nuñez, C., Padilla, I. L., Palma, G., Parker, L., Pradenas Marquez, B., Reeves, R., Reintsema, C., Rostem, K., Augusto Nunes Valle, D., Van Engelhoven, T., Wang, B., Wang, Q., Watts, D., Weiland, J., Wollack, E., Xu, Z., Yan, Z., and Zeng, L., "Variable-delay polarization modulators for the CLASS telescopes," in [*Millimeter, Submillimeter, and*

Far-Infrared Detectors and Instrumentation for Astronomy IX], Zmuidzinas, J. and Gao, J.-R., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **10708**, 107082M (July 2018).

[10] Switzer, E. R., Allen, C., Amiri, M., Appel, J. W., Battistelli, E. S., Burger, B., Chervenak, J. A., Dahlen, A. J., Das, S., Devlin, M. J., Dicker, S. R., Doriese, W. B., Dünner, R., Essinger-Hileman, T., Gao, X., Halpern, M., Hasselfield, M., Hilton, G. C., Hincks, A. D., Irwin, K. D., Knotek, S., Fisher, R. P., Fowler, J. W., Jarosik, N., Kaul, M., Klein, J., Lau, J. M., Limon, M., Lupton, R. H., Marriage, T. A., Martocci, K. L., Moseley, S. H., Netterfield, C. B., Niemack, M. D., Nolta, M. R., Page, L., Parker, L. P., Reid, B. A., Reintsema, C. D., Sederberg, A. J., Sievers, J. L., Spergel, D. N., Staggs, S. T., Stryzak, O. R., Swetz, D. S., Thornton, R. J., Wollack, E. J., and Zhao, Y., "Systems and control software for the Atacama Cosmology Telescope," in [*Advanced Software and Control for Astronomy II*], Bridger, A. and Radziwill, N. M., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **7019**, 70192L (July 2008).

[11] Milligan, M., Ade, P., Aubin, F., Baccigalupi, C., Bao, C., Borrill, J., Cantalupo, C., Chapman, D., Didier, J., Dobbs, M., Grainger, W., Hanany, S., Hillbrand, S., Hubmayr, J., Hyland, P., Jaffe, A., Johnson, B., Kisner, T., Klein, J., Korotkov, A., Leach, S., Lee, A., Levinson, L., Limon, M., MacDermid, K., Matsumura, T., Miller, A., Pascale, E., Polsgrove, D., Ponthieu, N., Raach, K., Reichborn-Kjennerud, B., Sagiv, I., Tran, H., Tucker, G. S., Vinokurov, Y., Yadav, A., Zaldarriaga, M., and Zilic, K., "Software systems for operation, control, and monitoring of the EBEX instrument," in [*Software and Cyberinfrastructure for Astronomy*], Radziwill, N. M. and Bridger, A., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **7740**, 774007 (July 2010).

[12] Wiebe, D. V., Netterfield, C. B., and Kisner, T. S., "GetData: A filesystem-based, column-oriented database format for time-ordered binary data," (Dec. 2015). ASCL:1512.002.

[13] Gilbert, S. and Lynch, N., "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News* **33**, 51–59 (June 2002).

[14] National Institute of Standards and Technology, [*FIPS PUB 180-4: Secure Hash Standard*], National Institute of Standards and Technology, Gaithersburg, MD (Mar. 2012).

[15] Denis, K. L., Cao, N. T., Chuss, D. T., Eimer, J., Hinderks, J. R., Hsieh, W. T., Moseley, S. H., Stevenson, T. R., Talley, D. J., U. -yen, K., and Wollack, E. J., "Fabrication of an Antenna-Coupled Bolometer for Cosmic Microwave Background Polarimetry," in [*The Thirteenth International Workshop on Low Temperature Detectors (LTD13)*], Young, B., Cabrera, B., and Miller, A., eds., *American Institute of Physics Conference Series* **1185**, 371–374 (Dec. 2009).

[16] Dahal, S., Amiri, M., Appel, J. W., Bennett, C. L., Corbett, L., Datta, R., Denis, K., Essinger-Hileman, T., Halpern, M., Helson, K., Hilton, G., Hubmayr, J., Keller, B., Marriage, T., Nunez, C., Petroff, M., Reintsema, C., Rostem, K., U-Yen, K., and Wollack, E., "The CLASS 150/220 GHz Polarimeter Array: Design, Assembly, and Characterization," *Journal of Low Temperature Physics* **199**, 289–297 (Jan. 2020).

[17] Battistelli, E. S., Amiri, M., Burger, B., Halpern, M., Knotek, S., Ellis, M., Gao, X., Kelly, D., Macintosh, M., Irwin, K., and Reintsema, C., "Functional Description of Read-out Electronics for Time-Domain Multiplexed Bolometers for Millimeter and Sub-millimeter Astronomy," *Journal of Low Temperature Physics* **151**, 908–914 (May 2008).

[18] Xu, Z., Brewer, M. K., Rojas, P. F., Li, Y., Osumi, K., Pradenas, B., Ali, A., Appel, J. W., Bennett, C. L., Bustos, R., Chan, M., Chuss, D. T., Cleary, J., Couto, J. D., Dahal, S., Datta, R., Denis, K. L., Dünner, R., Eimer, J. R., Essinger-Hileman, T., Gothe, D., Harrington, K., Iuliano, J., Karakla, J., Marriage, T. A., Miller, N. J., Núñez, C., Padilla, I. L., Parker, L., Petroff, M. A., Reeves, R., Rostem, K., Nunes Valle, D. A., Watts, D. J., Weiland, J. L., Wollack, E. J., and CLASS Collaboration, "Two-year Cosmology Large Angular Scale Surveyor (CLASS) Observations: 40 GHz Telescope Pointing, Beam Profile, Window Function, and Polarization Performance," *The Astrophysical Journal* **891**, 134 (Mar. 2020).

[19] Lang, D., Hogg, D. W., Mierle, K., Blanton, M., and Roweis, S., "Astrometry.net: Blind Astrometric Calibration of Arbitrary Astronomical Images," *The Astronomical Journal* **139**, 1782–1800 (May 2010).

[20] Folkner, W. M., Williams, J. G., Boggs, D. H., Park, R. S., and Kuchynka, P., "The planetary and lunar ephemerides DE430 and DE431," IPN Progress Report 42-196, Jet Propulsion Laboratory, California Institute of Technology (2014).

[21] Tufte, E. R., [*Beautiful Evidence*], Graphics Press, Cheshire, CT (2006).

[22] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., "Array programming with NumPy," *Nature* **585**, 357–362 (2020).