The Evolution of AutoCAMS

By David Samuels

AutoCAMS is a term loosely used to describe the automated or autonomous workflow processing of a CAMS station. CAMS has evolved from some simple batch scripts to help automate the daily processing of video files throughout the night to nearly full autonomous operation for months at a time.

The purpose of the NASA/SETI CAMS project is to confirm the known meteor showers and discover new ones. (see http://cams.seti.org). The project has proven to be accomplishing its original goals as well as revealing more information about the complex dust distribution in our solar system.

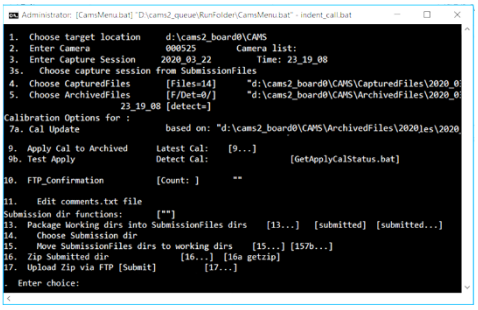
The purpose of this paper is to provide a history of AutoCAMS from the perspective of its original creator. AutoCAMS was originally a checklist style menu-based set of batch scripts or subroutines, run in a windows command shell console window. The checklist menu items would be performed in the order they needed to be performed (in computing, we call this a "workflow"). While the term AutoCAMS still pertains to the workflow, it also includes the subroutines and programs written to create an almost completely autonomous data collection system for a valuable research project.

INTRODUCTION

AutoCAMS was originally a checklist style menu-based set of scripts or subroutines run in a windows command shell console window that could be performed in the order they needed to be performed (in computing, we call this a "workflow"). It was originally called CamsMenu.bat in 2011. The menu included the ability to execute each of the steps that needed to be performed from manually restarting capture to post-capture processing, to sending the results to the NASA server. The menu included the ability to execute each of the numbered steps that needed to be performed. There is a main menu and a Utility menu. Eventually, it was renamed to AutoCAMS in 2011 when it was apparent that it was evolving into automating CAMS processing.

Circa 2006-2010 - Peter Jenniskens, world-renowned meteor scientist, and member of the Fremont Peak Observatory Association, near Salinas, California, used to bring college students and interns up to the Observatory to teach them how to perform meteor shower observations and recording. He'd bring a group of people with lawn chairs

Peter Jenniskens



Original Cams Menu

and hot drinks and they'd set up and manually record meteors on their sky maps. He was teaching them an astronomy skill of observing meteor showers. Very often, Peter would also point his DSLR at the sky and record the showers with photographic evidence. At the time, I was a board member at the observatory and Peter and I had some discussions about his work. Sometimes, he would invite me to participate in broader campaigns in order to use the camera images from different locations so that he could triangulate the meteor's positions. I was able to help with triangulation with my own photographic images from my house in Pleasanton, CA.

It was probably 2007 when one night in the dark, Peter and I were discussing how he'd ideally like to be able to perform multi-site video triangulation using highly-sensitive and expensive WATEC 902 H2 Ultimate security cameras. It was during that discussion that I mentioned to him that it might be possible to configure a computer to have multi-port video cards and record video from multiple cameras per computer to a hard drive. I told him how I had a Windows Media Server computer with a capture card that had two video ports for recording two simultaneous NTSC TV channels at the same time to a hard drive, but that I had also heard about cards with four ports. About a year or so went by when Peter told me about his new research project, which he called CAMS (acronym for Cameras for All-sky Meteor Surveillance). Of course, they needed sites for triangulation and Fremont Peak could be the first. Peter and I were able to convince the FPOA board in 2008-2009 that it would be good for FPOA to have a real NASA/SETI research project associated with the FPOA.

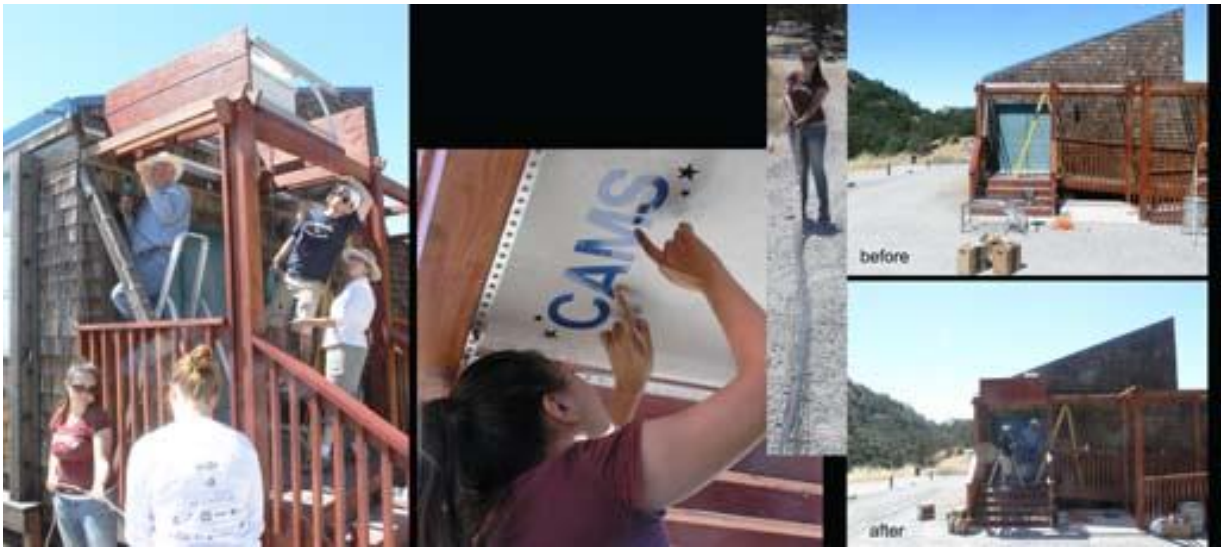
During that time span, in 2008, Pete Gural, under contract from NASA for Peter Jenniskens, wrote some C++ programs that perform capture, calibration, meteor detection, and so on, to run on BCSI Linux servers that had 4 port capture cards.

In 2010, we had attached the first 20 camera CAMS box to the East side of the FPOA observatory building and 20 coax cables were very tightly squeezed through a 4-inch conduit to the 5 Linux servers inside the observing room. No internet connection was provided. The servers were configured under contract

FPOA "Challenger" 1-meter f/3.58 telescope has been in operation since 1986.

from BCSI out of Colorado. The software for capture, calibration, and post-capture detection was written by Pete Gural. First light for FPOA was August, 2010.

5 Linux based BCSI servers and equipment for a single CAMS station



July, 2010 - Work party for installing the first CAMS station at Fremont Peak, California.

In 2011, Pete ported those Linux programs to Windows and made them available to Windows computer users in the form of what was called "single-CAMS". AutoCAMS became a set of scripts that, among other things, called Pete's programs in the correct order and with command-line parameters that reflected the selected capture session. In essence, AutoCAMS was a wrapper around the programs developed by Pete Gural. In addition, AutoCAMS included several additional utilities and reporting facilities as it related to managing a CAMS site.

Fremont Peak's overlapping stations were Lick Observatory site and Sunnyvale, CA.

Update: Late autumn 2017, *Peter J eventually replaced the 5-BCSI Linux-based servers per site at Fremont Peak, Lick, and Sunnyvale with single windows-based computers. Where there were 5 BCSI servers for 20 cameras before, we now have 1 computer with three 8-camera Sensoray boards with 20 cameras. And we installed the current AutoCAMS system that we designed for the Single-CAMS sites.*

Lick Observatory CAMS station (seen with Jim Albers) is about 70 miles from Fremont Peak, California.

SINGLE-CAMS PRINCIPLES

With all the things that could go wrong with an amateur operated single-CAMS station, it was apparent from the start that these **three principles of single-CAMS** should be adopted:

1. Capture is the first priority. No matter what, capture.
2. Don't send bad data to NASA.
3. Avoid causing requests to resend data.

Capturing is indeed the single-**most important function**. If you capture but don't do the other stuff right away, you can always do it later. But if you don't capture, then you'll never be able to re-create what you've missed… the data is lost forever. If you are triangulating with another site, then if you don't capture, it makes the work that the other site is doing useless. There have been numerous times when it was raining or cloudy in the evening (but captured anyway), to find out that it cleared up not too much later and the station was able to contribute valuable data that night. In some cases, a fireball would be captured through a hole in the clouds. Originally, I had my camera set up on a tripod under my back patio. Eventually, I purchased a set of security camera enclosures so that I'd have more flexibility as to the pointing. Enabling the camera only when you believe that it is good enough clear sky works against this first principle and the data proves that point. Automation is the way to achieve the goals of this principle.

First single-CAMS site was in Brentwood, CA August 2011

EVOLUTION

I believe it was in June 25, 2011, I met Pete Gural and Peter Jenniskens at the Fremont Peak observatory. They told me that Pete had just released his Windows version of the Single-CAMS software that was used on the Linux based systems. Peter J knew that I had a Watec camera already, and that getting started would not be any great expense for me as a kind of test site. Also, Peter knew that I lived within the range of being able to overlap with the CAMS California network.

Brentwood, CA: August 10, 2011, was first light for the first Single-CAMS. Starting capture manually, and performing all the post-capture processing the next morning was obviously time-consuming and onerous to do manually. The first piece of automating CAMS was done the first or second night - creating a scheduled task in Windows to launch a script that launches the capture program. That way, I wouldn't miss any part of the night due to forgetting to start capture or just not being home in time to start it. So, for me, CAMS has been automated from the beginning. **AutoCAMS evolved from that point**.

Pete Gural

Pete modified his programs so that it would take command line arguments from scripts in order to parameterize the startup. I also helped Pete design some multi-threaded programming so that he could do capture and meteor scanning at the same time, which Pete has implemented brilliantly in his capture programs and other programs he has developed

since then. (However, in the evolution of AutoCAMS, we later disabled the real-time meteor detection (with INI file settings) because it is not resilient to power outages and other interruptions and we had to figure out a way, outside of capture, to handle power interruptions). Also, Pete G improved the speed of detection by about 70x, which made it feasible to do post-capture detection in a reasonable amount of time.

AutoCAMS was originally called “CAMS Menu”. That’s because it was a menu that executed the

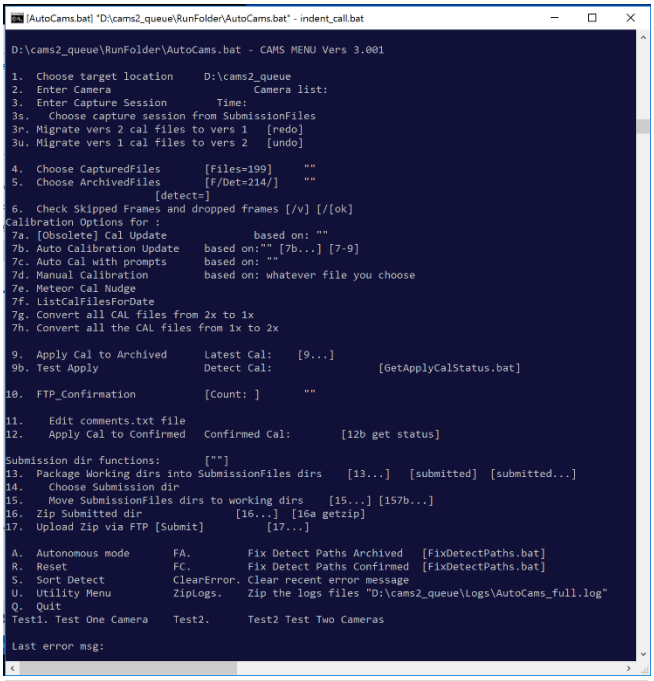


Fig 8. Screenshot of AutoCAMS.bat checklist menu. Notice how steps 1-17 are in the order they should be performed. Detection is actually #45 on the utility menu, since at that time we were using real-time detection.

necessary CAMS functions by entering the checklist item number. A few months later, it was renamed to AutoCAMS *(figures 8 and 9)*. AutoCAMS started as a simple set of Windows batch scripts and Windows scheduled tasks. When batch language was too slow, a batch script would create a VBScript, which was called and executed. There are only about 8 of those VB scripts. I wanted AutoCAMS to be “open-source”. The reason that batch script language

was chosen was so that anyone, in any time zone or location in the world, at any time of the day or night, could modify them if there was a problem. I could have written AutoCAMS in C, C++, Java, or VB, since I have decades of experience in those languages, but I chose Windows Batch language because nothing would need to be installed on people’s computers and there shouldn’t be versioning or DLL or runtime library version issues. With the other languages, not everyone would be able to program, compile, and link without having to

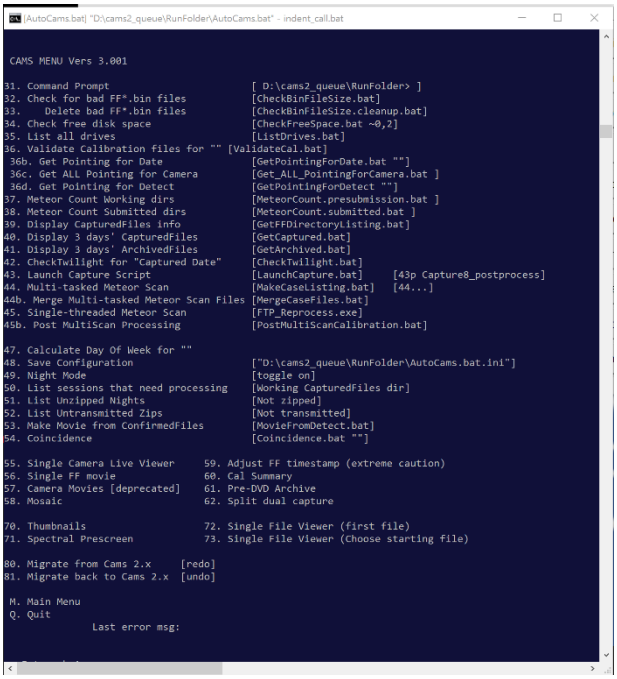


Fig 9. Screenshot of AutoCAMS utility menu.

deal with licensing issues, training, and such. From 2011 – 2017, the scripts evolved into over 100+ batch scripts that would call each other to get the job done. (By 2021, there are over 300 script files with over 90,000 lines of code). Over time, some of the scripts have become obsolete and no longer used. Some effort should eventually be put into removing all that are no longer useful. One of the **first scripts was LaunchCapture.bat**, which was called by a scheduled task.

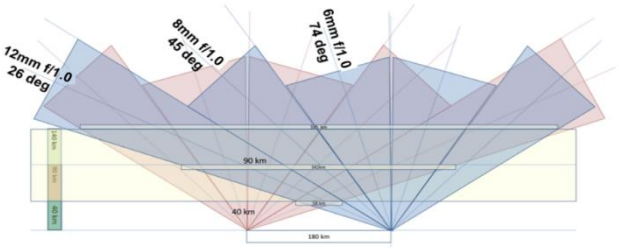
Jim Wray

Then Jim Wray joined the CAMS project and first light for his site was December 13, 2011, starting with a single old Watec camera, just in time to capture the annual Geminids meteor shower, and he has been providing data since then. Jim is the author of the 1967 book "The computation of orbits of doubly photographed meteors", which he wrote when he was director of the Institute of Meteoritics at the University of New Mexico in Albuquerque. Jim's site was over 100 miles from my house (until I recently moved to Texas), and it triangulated well with me, Lick Observatory, and eventually Sunnyvale. With Jim's station being the second single-CAMS station, but the first remote station, AutoCAMS had to evolve to be more user-friendly and reliable.

2012 saw CAMS sites installed in the BeNeLux sites (that's Belgium, Netherlands, and Luxembourg) started with 4 cameras. By February 8, 2014, there were 30 BeNeLux cameras operational.

October 28, 2013, AutoCAMS was configured and tested in Sunnyvale for the professional New Zealand arrays before shipping them out. This was the first time that the single-CAMS software and AutoCAMS was used on one of the professional CAMS stations. The boxes arrived in New Zealand December 12, 2013 - one damaged during transport. These stations were the last to use the expensive $1,500 16-port Sensoray board. The 8-port boards are only about $220 each.

In 2014, Jim Wray came up with the idea that we should be able to prove that cheap 1/3 inch cameras could now be sensitive enough to be used for CAMS data collection. So Jim purchased 2 eight-port Sensoray boards (about $220 each) and 18 of those cameras (average about $45 each), various models and brands, and I purchased a 4 port Sensoray grabber and a few different models of Chinese based 1/3" cameras and he and I tested the efficacy of using those cameras with CAMS so that people would be able to set up sites with more cameras for less money than using the Watecs. The Watec cameras cost about $400-$500 plus the $120 lens, depending on where you buy them from, availability, tariffs, etc. A typical 16-camera site would commonly top $10,000 using the Watec cameras. Some of these 1/3" based cameras purchased from Ali-Express were as low as $25 each. Pete G modified his FTP_CaptureAndDetect program and produced FTP_Capture8AndDetect.exe and FTP_Capture4AndDetect.exe so that we could run our tests. Jim ran two Sensoray boards at the same time for several months and we got good results. A paper was published for the at the 2014 IAU conference in Giron, France (Samuels et al., 2015 "IMC2014_Paper_ThirdInchCamerasForMeteorSurveillance_Samuels-Wray_Final.pdf") that shows how we can attain *almost* equal sensitivity to the Watec cameras (+5.9 meteor limiting magnitude) when the Watecs are using f/1.2 lenses and the 1/3" cameras are using f/0.8 lenses. Jim had f/0.75, and f/0.9 lenses. I had originally used f/1.2 lenses and I eventually switched to f/1.0 lenses. The f/0.8 lenses are not readily available in the 1/3" format, so we typically settle for f/1.0 lenses. Jim and I also calculated the spatial resolution for an average Watec camera with a 12mm lens pointing at roughly 45 degrees elevation. We came up with about 100-110 meters per pixel at the 90 km layer at that distance. Therefore, we aimed to achieve that same spatial resolution with the 1/3" cameras. Doing the math, to achieve the same spatial resolution, we decided to use 12mm lenses when pointing at 26 degrees, 8mm lenses when pointing 45 degrees, and 6mm lenses when pointing above 74 degrees.



This shows how to tune the focal length to match the elevation angle. In this figure, we show how two 16 camera stations, when designed for each other, can achieve full-sky coverage.

That strategy achieves full-sky coverage from 13 degrees elevation to 90 degrees.

Jim's 16-camera station was the first array with 2 Sensoray boards with 16 cheap 1/3-inch cameras in 2014. His site contributes almost as many meteors as any other around the world. Which proves its sensitivity.

San Mateo College became the third single-CAMS station December 23, 2011.

Steve Rau, in BeNeLux, had been trying to use the AutoCAMS scripts for his stations and a few others. He believed that automation was the key to getting consistent and reliable results.

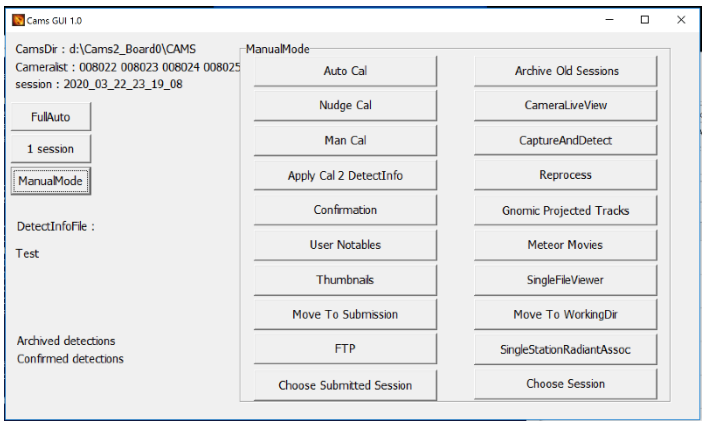
CAMS 2

During 2015-2017, there was a long-protracted period when I was away and it was nearly impossible for me to support remote CAMS stations. I either had very poor and intermittent access through a mobile hot-spot or I was at work behind a corporate firewall, which blocked my ability to support CAMS sites. Steve Rau stepped in and helped out quite a bit and he saved the day, taking over for me in keeping everything up and running. It was during this period (circa 2016-2017) when Pete G released CAMS 2. CAMS 2 expanded the camera number format from 3-digit to **6-digit** camera numbers and he **changed the file format** and a few other things that made the AutoCAMS scripts incompatible. After Steve got tired of waiting for me to return and fix AutoCAMS to adjust to CAMS 2 format changes, he reverse-engineered the AutoCAMS scripts using Delphi – Pascal. You could say that the "open-source" nature of AutoCAMS was successful due to his ability to do that.

AutoCAMS 2

Steve Rau

Steve's goal was to use Delphi/Pascal and create parity with the AutoCAMS checklist style menu system while making his system compatible with Pete's new CAMS 2 format. As he worked on things, his understanding of how I had coded the algorithms increased and he started coming up with good ideas on how to improve the workflow. Eventually, Steve had LaunchCapture.exe and CamsGUI.exe working, which appears to be what he still calls "AutoCAMS" in his documentation. I wished there was a way to keep Steve's code open-source too, but that is entirely up to Steve.



Screenshot of CamsGUI.exe. Notice how the buttons are in approximately the same order… to create parity with AutoCAMS.bat. Reprocess is the same as Meteor Scan or Meteor Detect.

I eventually returned home from my trip and I started working closer with Steve to get everything working smoothly. One of the issues we had was that the BeNeLux workflow was different than the rest of the world's CAMS workflow. This is in part because, in BeNeLux, they were required to do manual meteor confirmation. This made it impossible to fully automate and run autonomously. So, there were differences between his system and mine. We collaborated and we made the necessary changes to allow us (non-BeNeLux) to be able to use Steve's LaunchCapture system. So BeNeLux uses the LaunchCapture.exe and CamsGUI.exe to perform all of their duties, where there are some manual steps required to perform using CamsGUI each day. They don't use any of my AutoCAMS scripts anymore. The rest of the world (except DC and Sunnyvale) use a hybrid of LaunchCapture along with my AutoCAMS scripts, without much use of CamsGUI, for full autonomous operation. So, what should we call them? I think they can both be called AutoCAMS, since to me, AutoCAMS means to automate CAMS operations. How do we differentiate one from the other? I'm not quite sure and it probably doesn't matter.

Ideally, all configuration changes are governed through .ini files. Steve and I worked together to decide on which variables to use for these configuration variables and how they'd be used. Steve's LaunchCapture.exe and CamsGUI.exe programs are governed using a CamsGUI.ini file for each CAMS Instance. There is also a backup of CamsGUI.ini that is used in case you lose network connection while modifying it. There is one of these INI files for each CAMS instance. My AutoCAMS scripts now mostly use the "?:\cams2_queue\RunFolder**Cams2Global.ini**" file, although some of the scripts do read the CamsGUI.ini file variables and adapt accordingly. The non-BeNeLux systems do not use CamsGUI.exe very often. But it is still there and configured.

Steve and I worked out a way for his program to deal with multiple Sensoray capture cards, as each board would have its own "instance" of LaunchCapture.exe and CamsGUI.ini. Since none of the BeNeLux stations had multiple Sensoray boards at the time, it was difficult for Steve to visualize the issues and test his code. So, you have to appreciate what Steve accomplished. Steve's solution was for his LaunchCapture program to keep track of how many boards there are running and, during post-capture processing, one LaunchCapture instance sleeps until the next higher one ends before it starts doing its own post-capture processing.

My thoughts about how the workflow should work also evolved over time as to how to make the system more resilient to power outages and unexpected restarts.

LAUNCHCAPTURE

Like the original LaunchCapture.bat script, LaunchCapture.exe is a program that is somewhat sophisticated. Unlike the original LaunchCapture.bat script, which was a script called by a scheduled task, the LaunchCapture.exe program is called directly

Steve's LaunchCapture.exe for board 0 waiting until 8:57 to start the capture program.

from a scheduled task. There will be one scheduled task for each CAMS instance. A "CAMS instance" is a copy of the entire CAMS working directory tree, complete with a copy of the main capture programs, DLL libraries, star database, CAL files, runtime libraries, and a few other things. There is no need in a CAMS instance to use my scripts or a RunFolder, Transmitted directory, my .BAT scripts, or configuration files to be copied to the CAMS instance directory. Those files are in a separate common directory structure used across all CAMS instances.

Steve's LaunchCapture.exe program does a lot. Let's just outline its workflow:

1. It launches at the same time each afternoon (around 5 pm), reads the configuration files, and it checks the time and the sun-angle defined in the CamsGUI.ini file. The SunAngle variable defines the angle of the sun when we want to start capture. Effectively, we want to wait until it is dark enough to capture before launching the FTP_Capture8andDetect.exe programs. This value is displayed in LaunchCapture as "Sunset", but it means "Capture Start". Pete Gural's FTP_Capture*.exe programs do the same thing; except they interfere with the user's keyboard, once a minute, during the time between when it started and the time capture actually begins.
   LaunchCapture.exe's method of waiting for the sun angle is a workaround for that issue with Pete's waiting algorithm. An example is like this: Let's say it's June 20 – shortest night of the year. If we run LaunchCapture at 5pm each day (a time before it is astronomically dark all year), then the time to start capturing video would be, let's say, 9 pm. So, for 4 hours, the program needs to do a well-behaved "wait" before cameras start capturing. Meanwhile, you are free to use the computer during this time.
2. When it is finally time to start capturing, LaunchCapture awakens, and calls the FTP_Capture*.exe program specified in the .ini file, with the correct command line arguments. Then it does a well-behaved wait until the child capture process ends (that will likely be just before sunrise the next morning, as determined by Pete's capture program using the sun angle passed to it from LaunchCapture). This time is displayed in the LaunchCapture window as Sunrise, but it should be labeled as "Capture End". While it is waiting, LaunchCapture instance is not consuming any CPU resources. It is not polling, it is waiting.
3. Whenever capture ends, LaunchCapture awakens and performs post-capture processing from the highest board number to the lowest. It starts with the most recent capture session in the CapturedFiles directory, then it works backward and performs post-capture processing for any backlog of unprocessed capture sessions until the CapturedFiles directory is empty.
4. LaunchCapture's post-capture processing is done in phases for each capture session for that board:
   4.1. Validation
   4.2. AutoCal
   4.3. Detection
   4.4. Apply calibration to detect file
   4.5. Manual Confirmation (only for BeNeLux users)
   4.6. Migrate to SubmissionFiles
   4.7. Create the Transmission Zip (only for BeNeLux users)
   4.8. Upload the Transmission Zip (only for BeNeLux users)
   4.9. Archive (only for BeNeLux users)

   Once it is done with the Apply step, it "moves" all the files to a directory that uniquely identifies the board number and the capture session under SubmissionFiles.

Each SubmissionFiles directory will have subdirectories for ArchivedFiles, CapturedFiles, optionally ConfirmedFiles, FTP, and Logs. The FTP directory is where a few files are pre-staged for transmission, such as the FTPdetectinfo file, local-midnight FF files, CAL files, and weather forecast file. If you are a BeNeLux user, you are required to use the CamsGUI program to launch the Confirmation program, do the confirmation step, and then manually upload the files to the server (by selecting the appropriate button in CamsGUI). If you are not a BeNeLux user, then confirmation is not performed and all required steps are handled autonomously.

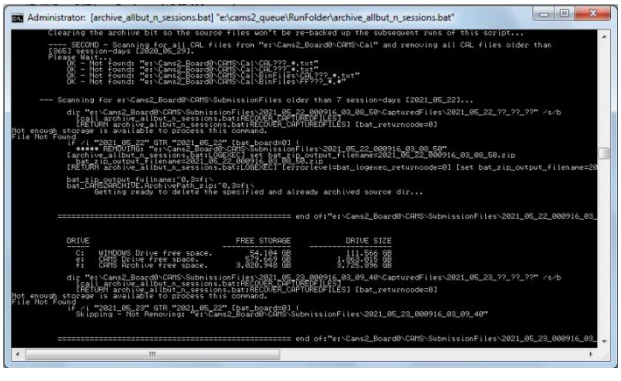
Then it repeats these steps for each capture session if there is a backlog of capture sessions remaining in the CapturedFiles directory. (The backlog could have been the result of power outages or manual restarts).

When it is able to complete all its work, then the CapturedFiles directory for that CAMS instance should be empty.

5. Once LaunchCapture has completed post-capture processing for all the capture sessions in the instance's CapturedFiles directory, it ends.
6. Once LaunchCapture ends, if there are multiple CAMS instances, the remaining LaunchCapture instances will have been polling the system processes every 5 minutes to see how many LaunchCapture.exe instances there still are.
7. If there are multiple CAMS instances, there will be 2-3 instances of the LaunchCapture.exe process to start with. When each LaunchCapture.exe process started, it kept a count of how many other LaunchCapture processes there were at the time and it assigned itself a number based on that count. The post-capture processing happens from the highest CAMS instance to the lowest (as determined by board number starting with 0). When a higher board number's LaunchCapture instance is performing post-capture processing the lower board number's LaunchCapture instances poll once every 5 minutes to see if the instance count has dropped and if it is that LaunchCapture instance's turn to run.
8. If the instance count drops to its assigned count number, then that LaunchCapture instance awakens and performs its own post-capture processing, using the same steps as above.

Non BeNeLux sites have been using a hybrid of Steve's LaunchCapture.exe with the addition of my AutoCAMS scripts, which are now all situated in the "?:\cams2_queue\RunFolder" for full autonomous operation. Unlike with per-instance LaunchCapture.exe and CamsGUI.exe, the queue system is one system for the entire computer. No matter how many CAMS instances there are, there is only one "cams2_queue" directory. Each CAMS instance location is specified in the "Cams2Global.ini" file, which contains the variables used by the queue-based scripts to locate each of the CAMS instances.

STORAGE MANAGEMENT/ARCHIVING



Screenshot of the Archive program just getting started. This Mars Hill station has drive E: as the CAMS drive and F: as the Archive drive.

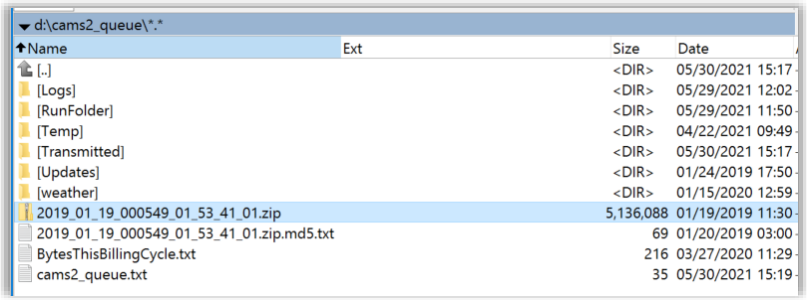
A huge issue to deal with is the massive amount of data produced by a CAMS station. Each

camera can produce about 6.5 GB of data files during capture each winter night (depending on the latitude). Less on summer nights. While most of the BeNeLux sites didn't have 16-20 camera sites with dual or triple Sensoray boards, most of my sites did. By "my sites", I am only using that term as a way of referring to the stations that I've been managing. With a 20-camera site producing over 120 GB per night, and with some of their cameras sometimes being noisy because of faulty wiring or tarnished connections, even a 6 TB hard drive couldn't hold enough archived data. With a background in working for EMC, which was a leader in the n-tier storage world, I realized that AutoCAMS needed similar storage management - some 2-tier or 3-tier storage management. You see, I kept getting pulled into fixing people's disk-full issues. So, I redesigned the Archiving scripts and protocols to not only archive data, but to also keep the data management under control using an n-tier strategy. Originally, we would wait until MaxDaysToKeep INI file parameter to trigger culling of the CapturedFiles directory. Culling the CapturedFiles directory at that time was possible because a copy of any FF file that had an event (meteor) was also stored in the ArchivedFiles directory. Therefore, none of the FF files in the CapturedFiles directory are needed after a certain amount of time. Culling the CapturedFiles FF files removes up to 120 GB of unneeded disk usage. Then we would ZIP the remaining files of the capture session (including all the FF files in the ArchivedFiles and ConfirmedFiles directories) into a zip file and move that to the archive drive/directory. That way, if a capture session was ever needed, we could unzip an archived capture session and do research on it as needed. The INI files specify a separate Cams Archive variable for the archive drive. Ideally, the archive drive is a separate removable hard drive that, when full, can be swapped out for a new one. The Arizona stations were the first to implement my external archive drive design, and that has proven to be effective… and **recommended**. Eventually, I learned that, for reliability of the backup/archive, the archive scripts should instead create the archive zip file as soon as it was possible instead of waiting MaxDaysToKeep. It was only the culling that needed to wait. Then I leave a flag file, marking the capture session as "AlreadyArchived.txt", so that it does not need to be zipped and archived again – only the session's CapturedFiles dir need to be culled upon MaxDaysToKeep. This makes the archiving more resilient to power outages, aborts, and other interruptions and it protects against not getting a chance to run when there is a backlog.

To re-archive a session, simply delete the AlreadyArchived.txt file from that capture session's CapturedFiles directory. If needed, you can always unzip the archive zip file to its location in the SubmissionFiles directory. When you do that, you may also want to copy all the FF files from its ArchivedFiles directory to its CapturedFiles directory, so that the programs and scripts and re-process, as if it's a freshly captured session.

Using this storage management scheme, we have some 16 camera stations with small 1 TB hard drives that have run for months or nearly a year, without having to intervene. Note: 16 camera stations are difficult if they only have 1 TB of storage, so we recommend obtaining more storage for the CAMS drive in those situations, as well as having a separate removeable archive drive. Sadly, we can't archive all the data like this forever. We have also found that we cannot depend on humans to periodically cull their archives or to move them off-site. So, I also had to implement the ability to automate the culling of very old archive zip files when the hard drive starts to fill

up with archive files. We have automated this too. We delete everything that is too old (about 2 years), from the archive's Transmitted, CAL, and SubmissionFiles. We use the INI file setting [CAMS2ARCHIVE].MaxYears_archived_SubmissionFiles, and MaxYears_archived_TransmittedFiles. The CAL files are always archived when they are older than MaxDays_Cal days, which defaults to 365 days.



cams2_queue *directory with one transmission zip (not yet transmitted) and its corresponding .md5 file. Also shown is the* ***cams2_queue.txt*** *file with the name of the zip file. Once upload is verified, the zip and its .md5 file will be moved to the Transmitted directory. All the scripts are placed in the RunFolder.*

UPLOADING

The upload routines are another set of routines that have evolved over the years. When we upload to the NASA server, we have always collected just a very small part of the capture session, compressed it into a zip file, and uploaded it to the NASA server to a directory specific to the station's CAMS network. This is why a **reliable internet connection is required**. That file is called the "transmission zip file". The file consists of the detect file, CAL files that were used in this session, local-midnight FF files, and a few other files that indicate how it was configured during that session. An 8-camera transmission zip file is typically about 2 MB - 5 MB. With a noisy camera, these can be 50 MB – 80 MB. BeNeLux sites use Steve's LaunchCapture to create and upload the transmission zip during its post-capture processing. But because the non-BeNeLux users run autonomously, the procedure needed to be more resilient. Therefore, non-BeNeLux sites use my new queued system AutoCAMS scripts to create the transmission zips and to upload. The AutoCAMS scripts create the transmission zip files by calling the ZipCams2FromFTP.bat script.

Originally, we had used the Microsoft FTP program to upload the transmission zip files to the NASA server. However, we learned that we needed a tool that is more resilient. FTP had many issues and was not reliable at all. We switched to WinSCP for that. It performs retries and it returns error codes. But, WinSCP can still fail. We have encountered numerous situations where the FTP upload failed for all manner of reasons. The reasons range from local network being unavailable, to power outage before transmission is complete, to NASA server being down for a few days. I wanted to design a system that could handle this automatically. So, I developed the "queued" upload. Regardless of the ability to upload successfully, all transmissions that are ready are placed in the queue directory. Along with the zip file is an MD5 hash file for that zip file. (A hash file is a small text file with a large hash number that uniquely represents the contents of a file. It uses cryptographic algorithms to come up with the hash number). The upload script creates a text file that lists all the transmission zip files in the queue. Then it sorts them and then uploads them from oldest to newest. Once a transmission zip file is uploaded, for validity testing, it is immediately downloaded to a local temp directory. For validity testing, we rehash the downloaded zip file with MD5 and compare the two before/after hash files. If they are different, then we know the file got corrupted and we move on to the next transmission zip file, leaving the original transmission zip still in the queue, since we don't really know the

reason the upload failed. We will attempt to re-upload the failed zip file on the next go-round. If the MD5 matches, then the temporary zip file is unzipped. If the unzipping function fails, then we know that the zip file is corrupt in some other way. Otherwise, the upload succeeded. If the upload succeeded, then we upload the transmission zip file's .md5 file to the NASA server and then MOVE the transmission zip file and its .md5 file out of the queue directory and into the Transmitted directory. The people at NASA/SETI could write scripts such that if a zip file were present but its matching .md5 file is not present, then they'd know either the zip file on the server is corrupt or it is not finished uploading – in other words, they shouldn't use it yet. They should be able to use the .md5 files to do their own hash-check before incorporating the data into the day's Coincidence processing. Before a zip is added to the queue, the Transmitted directory is checked to ensure that it is not already there - to avoid uploading it a second time. If you really want to upload it a second time, then you must delete its md5 file and zip file from the Transmitted directory first.

This protocol is not too dissimilar to the Amazon AWS S3 upload protocol (a fact not discovered until years after I developed it). However, since Amazon's server is based on web services, Amazon avoids the download step and just compares the hashes. We can't do that because our server is just a dumb FTP server. Because of that, Amazon AWS S3 upload protocol is much faster. I had to come up with a different workaround with all the logic on the **client-side**. Once the queue is processed, we wait about an hour and start the next go-round, by checking the queue for any remaining zip files in the queue directory. The queue might not be empty an hour later due to failure to upload one or more zip files the first time or, while we were processing the queue, other scripts had completed post-capture processing and new transmission zip files would be available in the queue. If the queue is empty, then uploading is done for that day. If there are still zip files in the queue, then we keep trying every hour until 3 pm, when we need to perform archiving and then start getting ready for the next night's capture.

There are two other optional features that were designed with our uploading protocol. One is that the zip files that we upload can be sequenced with a sequence number suffix. We might need this if we had to upload to a server that was read-only. The other feature is the ability to send split zip files. A split zip file is a zip file that is sent in small chunks and reassembled by someone at the receiving end. This prevents the need to re-transmit very large files in the case of an upload failure. At some sites with unreliable internet, it is more likely to be able to consistently transmit ten 2 MB files than one 20 MB file. And if the one of the 2 MB files had an issue during the transmission, only the failed split files would need to be retransmitted. Both these features of the protocol are built-in to the uploading scripts; however, we almost always disable them in the INI file. Especially since, at the server end, they have not written the code to be able to handle sequenced file names or to reassemble the split zips.

This protocol has proven to be reliable and resilient. Anyone else, such as RMS, could adopt a similar protocol to ensure reliable uploads.

What's cool about this approach is that if there is a problem, then it kind of fixes itself. We recently had an issue at one very remote 20 camera array, where there was about 1.5 months of data that had not been processed. I don't remember the exact reason why. But once the problem was resolved, the AutoCAMS routines simply started working as they should. It took 7 days for the system to automatically catch up by itself. One of the issues we were

having at the time was that the cameras had gotten so noisy, that each capture session was expanding to about 200+ GB and taking too long to do post-capture processing. We fixed the problem with ground-loop baluns, because the wiring was already in place. However, moving forward, using Cat6 with Video Baluns at each end instead of coax with ground loop baluns is recommended.

UNIQUE NAMING CONVENTION

A lot of thought was put into the naming convention for the archives and the transmission files that we upload to NASA. This is something that has also changed and evolved from 2011 – about 2014. The naming convention is important because (A) it must allow us to upload multiple sessions in a night in case there were power glitches or other restarts, either expected or unexpected; (B) it had to allow these multiple uploads without them overwriting previous uploads; (C) it allows the Coincidence process to include all of the capture sessions, no matter how short, for its triangulation procedure; (D) a similar naming convention was created for the archive files.

For capture sessions, transmission zips, and archive zips, the convention is to use the capture session start time with a unique camera number:
"**yyyy_mm_dd_<camera>_HH_MM_SS.zip**".
We separate the start date from the start time with the camera number in order to facilitate sorting and grouping. For "camera", we use just the first 6-digit camera number for the CAMS instance. The file names do not need to have the entire camera list as we once tried. Neither do they need the first and last camera numbers. All we are trying to accomplish, and this is important, is to come up with a unique name that does not conflict with a file name from our own or from another site. So, for other capture session-related files, such as a detect file, a unique naming convention uses something like this:
"[prefix]_yyyy_mm_dd_[camera]_hh_mm_ss_[suffix]"
where <prefix> is the name of the file, such as "FTPdetectinfo", and [suffix] would be what other information is needed. <suffix> would be any other information that needs to be conveyed and/or the type/extension of the file. It's important to add a dot three letter extension on the file so it can be sorted, grouped, and/or associated with an appropriate application. The CAMS date/time format uses only underscore "_" as separators instead of ":" and ".", "-", and ",", etc. That way, no localization of parts of dates and time values is required. "yyyy_mm_dd" would be the capture session UTC date of the start of the capture session and "hh_mm_ss" would be the capture session UTC time of the start of the capture session. Capture session directories in CapturedFiles are already named like this: "yyyy_mm_dd_hh_mm_ss", where that indicates the UTC time of the start of the session. So, you can determine the capture session start time by parsing the directory name.

For non-capture session related files, a unique naming convention would be something like this:
"[prefix]_yyyy_mm_dd_[station]_hh_mm_ss_[suffix]"
or **"[prefix]_[station]_[suffix]"** for files that don't require date/time. Where "yyyy_mm_dd" and "hh_mm_ss" would be the date/time, in CAMS format, when the file was created and [station] would be some code that uniquely identifies the station. For example, you could use the first camera of the first CAMS instance or something like the "CODE" column in the CameraSites.txt file, or the lat/long of the site, or anything else, just as long as it uniquely identifies the site worldwide. Another option, if the time is not important, is to use this convention:
"**[prefix]_[station id]_[suffix]**"
For example:
"Status_000957_MC_Meteor Crater.txt"

It's important to use a naming convention that performs its own grouping when sorted. For example, sorting by station name would not group all the Arizona stations together. However, including their first camera accomplishes that.

REMOTE AUTOCAL

While this is no longer an issue in the current AutoCams, there is an interesting bit of history that should be included. After the CAMS network started expanding around the globe, CAMS started to be deployed outside of the USA. There was a time when the NASA contract under which the CAMS programs were originally developed prohibited the deploying and executing of certain parts of CAMS outside the USA, specifically, the calibration routines. So, in keeping with the letter of the contract, I had to invent a protocol for calibrating a non-USA station's capture sessions on a USA-based server and sending the results back to the non-USA station.

An Amazon AWS server was configured and deployed, in a USA datacenter, to sit there and wait for calibrations requests. These calibration requests were uploaded into a queue folder. It would AutoCal the requested session, and put the results into a folder where the non-USA station could download them from. The protocol was very complex.

Thankfully, Pete G had earlier created the FTP_CalStarExtractor.exe FTP_CalStarsReconstitution.exe programs, which I employed for this purpose. FTP_CalStarExtractor.exe program would create a text file, named "CALSTARS*.txt", that represented where many of the calibration stars were in the FF files. FTP_CalStarsReconstitution.exe could then be used on the remote calibration server to reconstitute into low-res FF files that would be compatible with and used by FTP_MeteorCalAutoUpdate.exe to perform autocal. The CALSTARS files would be created on the local station, zipped, and then uploaded to the remote calibration server's queue directory. A strict naming convention was used to identify whether the files were going to the server or coming from the server. Then, the local station would wait for the results to appear in the remote calibration server's "ready" folder.

In the meantime, the remote calibration server awakens from a well-behaved "wait" whenever files appear in its queue directory. At that point, it would reconstitute the FF files from the CALSTARS, perform autocal, and move the results to the server's "ready" directory.

The Remote Calibration requires the following scripts:

- AAA_CleanServer.bat
- AutoCal.bat
- AutoCal_LocalDownload.bat
- AutoCal_LocalUpload.bat
- AutoCal_Remote.bat
- ftp_upload_robust_seti.bat
- Lock_AutoCamsMenu.bat
- MonitorDir_Poll_Client.bat
- MonitorDir_Poll_server.bat
- ReadRemoteConfig.bat
- Start_CalServer.bat
- Unlock_RemoteSession.bat
- Waitfor_AutoCal_LocalDownload.bat
- waitFor_AutoCams_lock.bat

We learned that files would have access contentions, so a locking protocol had to be developed for AutoCAMS too. It uses Lock_AutoCamsMenu.bat and Unlock_RemoteSession.bat.

The AutoCal_LocalUpload.bat script is kind of where it all kicks off. It performs the first phase of the AutoCal_Remote procedure by uploading

the CALSTARS and associated Cal files into a zip file and uploading it to a remote CAMS calibration server, which is USA-based, which performs the AutoCal function remotely and sends the results back.

The argument pairs can be placed in any order.

arg1 /capturedir bat_capturedfilesdir

arg2 /caldir bat_caldir

QUEUE FORMAT:

Queue Name:
".\CAMS\AutoCal_Remote_Queue\AutoCal_Remote_Queue.txt"

The queue items will be entered as follows:

CALSTARS??????_????????_??????.zip,"<bat_capturedfilesdir>","<bat_caldir>","<command line>"

The original source zip file that got uploaded will remain in the queue folder until after its results have been safely downloaded. That way, if necessary, it can be resubmitted easier.

Steps involved:

PHASE ONE

1. When AutoCal.bat is run locally, if it's run using the /remote command line switch, it behaves differently. Instead of doing autocal, it performs the AutoCal_Remote.bat procedures.
2. Run FTP_CalStarExtractor.exe.
3. Zip the CALSTARS*.txt files, along with the camerasites.txt file and the most recent CAL file for each camera into a zip file. The Zip file will be stored in the ".\CAMS\AutoCal_Remote_Queue\" dir.
4. Move the zip file to the ".\CAMS\AutoCal_Remote_Queue\" folder.
5. Add item to the remote queue. Include the zip file name, bat_capturedfilesdir, bat_caldir
6. Upload the zip file to the server [which triggers phase two on the remote server]
7. Wait for the server to create your results using the MonitorDir program locally, set to invoke AutoCal_LocalDownload.bat when it is signaled.

PHASE TWO on the remote server:

A. Moves the zip file from the TestCal\From folder to the Zip folder.
B. Unzips the zip file to the TestCal folder.
C. Reconstitutes the FF files from the CALSTARS files.
D. Move the CAL, FF, camerasites, and log files to TestCal\CapturedFiles\{date time} folder.
E. Launch AutoCal.bat /remote. This produces a CAL file for each camera into the CapturedFiles dir as well as a BinFiles dir, containing the CAL files and their corresponding FF files.
F. Package the results into a zip file and move it into the TestCal\From dir. [which triggers phase three on the local system] If there are any errors during the AutoCal_Remote.bat script, the error is logged into the “ERRORS.txt” file and that “ERRORS.txt” file is placed into the “..._Results.zip” file into the “From” dir, which triggers Phase Three.
G. Clean up any leftover files.
H. Process any remaining zip files in the TestCal\From dir that may have been uploaded either before MonitorDir was loaded or while the current session was being processed.

I. Re-launch the MonitorDir program.

PHASE THREE

1. The monitor triggers the AutoCal_LocalUpload.bat script when a file appears in the dir [remember, it may not be a file that pertains to one of the local queue items]
2. This script first checks to see if the triggered item pertains to the local queue items.
3. If it does not pertain to any of the queue items, then it should go back and relaunch MonitorDir to wait for more results
4. If it does pertain to at least one of the queue items, then it should do the following steps.
   4.1. It must download the file
   4.2. Remove the file from the server
   4.3. Unzip the file to a temp dir
   4.4. Move the contents to their appropriate areas [.\Cal, .\Cal\BinFiles, .\{capturedfilesdir}]
   4.5. Remove the downloaded file and any temp files used to do the work.
   4.6. * Run the script specified in the queue item to complete the autonomous processing.
   4.7. Loop back to step 2 above to process any remaining queue items. There can be a lot of reasons why they didn't get processed before.

   It's amazing that it all works.

REBOOTING

We have also repeatedly learned, through troubleshooting, that we needed to reboot the computers just before archiving and also before LaunchCapture each day so that (a) post-capture processing of backlogs will not interfere with current archiving procedures or capture. I have only implemented this at a few test sites so far. Rebooting before archiving solves a few problems that have been experienced. When backlog processing is active, it can slow the archiving procedure to the point where it could take more time than we've given it (usually about an hour) to complete. (b) Remember, the computer reboots again each day at 4pm to clear the system of rogue applications in order to avoid dropping frames from the video grabber during capture. Rebooting the system also clears any issues with computer locking up due to electrical glitches and such. You can always remotely regain control after 4pm. Also, rebooting just before capture is about the only way to ensure that the fewest number of other programs are running. Other programs running is a common cause of dropped frames.

Rebooting is performed by the "Cams2 Reboot PC" scheduled task, which has two triggers.

STATUS REPORTING

In 2018, I introduced the GetStatus.bat script. The idea was for each station to report and upload its status to the server. Calling GetStatus.bat is incorporated within the Upload_Queue.bat script. Each station runs the status report several times per day. It also runs after each reboot. After it is run, its report is uploaded to the NASA server to a shared location where all status reports are kept worldwide. Each status report has a unique name according to the station name, not according to the session name. Therefore, status reports on the server will overwrite the previous status report.

A status report contains a lot of information. It will contain information about your system, such as windows version number, disk space, when the report was created, how much network data you have used in this billing cycle, and when it was most recently rebooted. An issue detected here will be cause to take action, for example, to free up some disk space.

It also contains a list of the 10 most recent Transmission files and their MD5 files. This gives you an opportunity to determine if your upload procedures have been successful. If they are not, then you'll need to take action to determine why and resolve it.

It reports on how many unprocessed Capture sessions there still are in the CapturedFiles directory. If there are any that appear, then you will be approaching a disk space issue. This is an indication that some action needs to be taken to correct the problem.

A big part of the report is the SubmissionFiles section. In this section, each of the most recent 10 capture sessions are listed for each CAMS instance. For each capture session, you will see the entire size of the capture session in MB, how many files there are (you can compare these numbers with other sessions), the session name, the total number of detections, and whether the Validation, AutoCal, Detection, and Transmitted phases of post-capture processing have completed or failed.

In addition to that information, each camera's information is listed to help you identify trouble spots. For each camera, we show the capture session, the camera number, the number of detections for that camera, the FOV and image scale, the CAL file, and dropped frames information. Too many detections points you to a noisy camera. Too many dropped frames points you to a CPU contention problem in the computer. If the image scale is different than previous capture sessions, then you might have a scale flip/flop problem.

A list of the 20 most recent CAL files is then shown in the CAL files section. A camera that consistently fails to calibrate indicates that some manual calibration intervention is needed.

The archive settings are then shown, indicating how long you intend on keeping session data around outside the archive. If you are running out of disk space, you can examine these figures and look at whether to modify the MaxDaysToKeep setting. MaxDaysToKeep applies to the number of days to keep CapturedFiles FF_*.bin files before culling them.

There is also a list of the 10 most recent archive zip files, along with their sizes. Excessive sizes will inform you of issues related to archiving sessions with noisy cameras and it is possible that it takes so long to process, that you never get a chance to complete the ever-important disk management part of daily processing. Also, a list of each of the Cams_Archive subdirectories and their sizes is next, in case the information is helpful.

There is a list at the bottom that indicates which processes were still running at the time the report was generated.

The final step is it uploads the report to the status folder in the server. This provides a central place to access station status without having to connect to each station. It is helpful to network coordinators so they don't have to remotely connect to numerous stations to determine if they are OK. Also, you should be able to configure your phone or tablet to access this information via FTP when you are away from your computer. Lastly, they are used by the Status_Check.bat scripts.

STATUS_CHECK REPORT

Sometimes, the GetStatus reports contain too much information for you to *quickly* identify issues that you might be having that require you to take action. So, some Status_Check reporting scripts were created to only report on the alerts that you need to consider taking action on. The Status_Check.bat scripts download and *read* the GetStatus report(s) from the collection of status reports on the server. If nothing is wrong, they

will simply list that the status report was found, its date/time, and that it was OK.

However, the Status_Check scripts were created to "read" the GetStatus reports of the stations in your network and to alert you of issues, such as disk space getting too low, noisy cameras, and cameras with excessive dropped frames. Excessive dropped frames and excessive detections per camera have three alert levels: WARNING, WATCH, and CRITICAL.

In addition, it will report if the station has not uploaded a status report after 2 or more days, indicating that someone needs to attend to the station to make sure that it is up and running. There are a few other things that are reported in the status_check reports, but I don't remember right now. One of them is whether the power is off to individual cameras. This is a common condition when a cable is cut or tarnished or people are trying to use Christmas tree timers in timer mode instead of day/nite mode to power the cameras and the timer is not set to the correct time.

The status report is generated as a text file, then it is converted to an HTML file with hyperlinks to the full GetStatus report that was downloaded from the server. This is a very useful tool.

The Status_Check reports are not uploaded to the server. They are kept locally. They can be run from any Windows computer that has the AutoCAMS scripts configured. They are also only run when you tell them to.

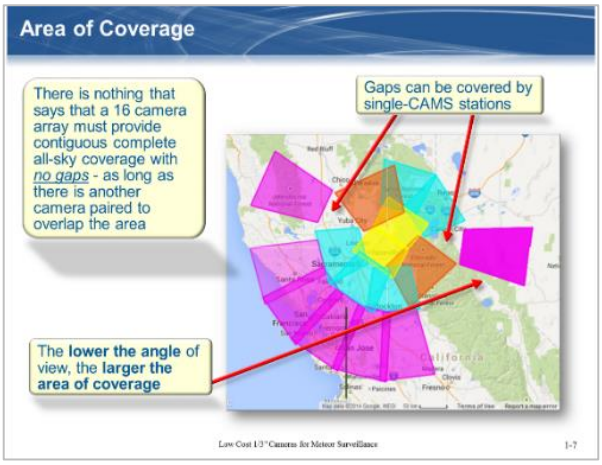
The Status_Check reports are designed to also be able to work with a Status_Check_<network>.txt file. When the status_check script is told to use the file, it produces its report for all stations in the local CAMS network. For example, **status_report_AR.txt** to produce a status report for all stations in the Arkansas network.

You can create a shortcut on your desktop to launch the script with your network file.

CAMS POINTING TOOL

While it's not specifically part of AutoCAMS, around September, 2014, I developed the CAMS Pointing Tool (available here: http://davesamuels.com/cams/camspointing/scripts/latlong.html). It can be used to create a Laydown before setting up your AutoCAMS.

The tool takes time to get used to, but it is a useful tool for site operators to help in determining where to point their cameras to attain the best overlap with other cameras. It can be used to create, what we call a "laydown" for one or many cameras from one or more sites.

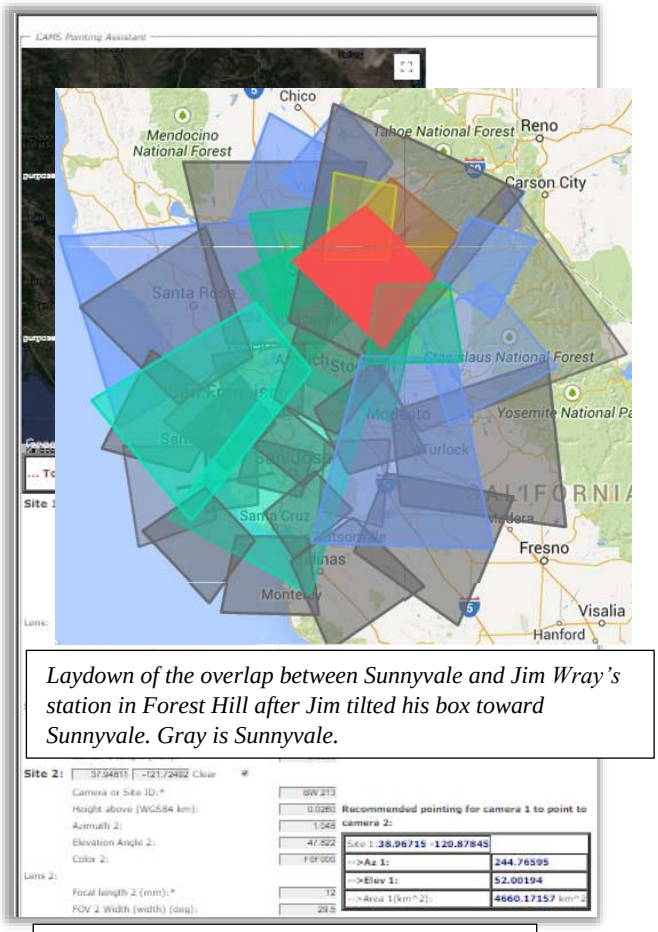


An alternative Laydown can be more flexible

Thanks to Chris Veness for providing the haversine functions and the basic layout for the page. The input fields are arranged in the order that they appear in a CAL files.

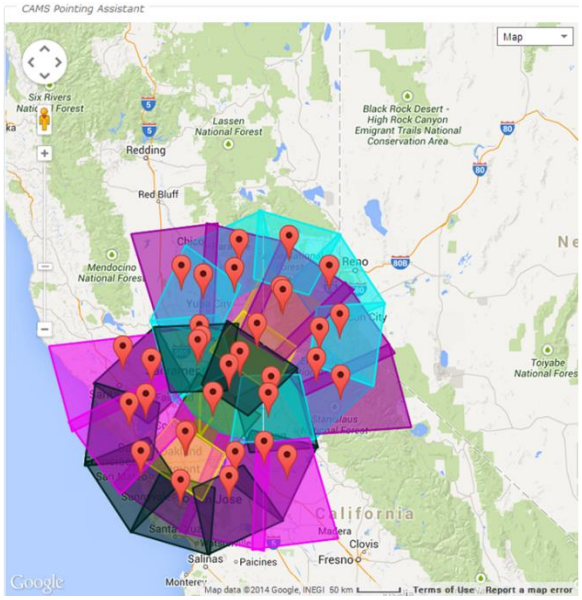
While not part of AutoCAMS, it has proven to be a useful tool by many CAMS users.

This image shows a Laydown of one 16 camera all-sky station.



Laydown of the overlap between Sunnyvale and Jim Wray's station in Forest Hill after Jim tilted his box toward Sunnyvale. Gray is Sunnyvale.

CAMS Pointing Tool web page. Used for creating "laydowns" and figuring overlap with other cameras.



This image shows a Laydown of two 16 camera all-sky stations optimized for each other. When designed for each other, full-sky coverage can be achieved.

UPDATE SCRIPT

Somewhere along the line, an UpdateScripts.bat script was invented and developed. It was developed in order to keep all the stations updated as much as possible to the same version of the software. It is launched daily by a scheduled task, "Update Scripts". The script's job is to check if there are pending updates that you don't have.

While it was a lofty goal, in some specific cases, it wasn't practical due to network problems and such. We currently distribute and install the scheduled task, but we disable it. Also, it is not compatible with Cams 2, so we must keep it disabled until UpdateScripts.bat is migrated to Cams2.

UpdateScript is more of a system than just a script. It has its own directory structure under "?.\cams2_queue**Updates**". Under the "Updates" dir, there are other subdirs for Pending, Temp, Updated, and Zips. These are places where different kinds of files in different stages of updating are stored.

The **Pending** dir is where new updates, which are downloaded from the server, that have not yet been processed are temporarily saved. Only zip files are used for this protocol. Mostly because they preserve the file modified dates. It's important that these zip files are placed in this directory in the version order. The version is determined by the CAMS timestamp on the zip file. If there is only one update in a single day, you can use "00_00_00" as the time segment of the zip file name. If there is a second one, there is no harm in using the time segment as a sequence counter, such as "00_00_01". But it was designed to hold the time, so you can use that too.

The **Temp** dir is where we unzip an update zip, one at a time, and process.

The **Updated** dir is where we place zips that have been downloaded and incorporated after completion.

The **Zips** dir is where we place zips that…
A benefit of keeping the zip files around is so that you can use them to fix regression bugs and revert back to a previous version of a file by unzipping it manually to its target location.

Finally, the UpdateScript.log file is kept in the "?:\cams2_queue\Updates" dir. The layout of the UpdateScript.log file looks like this:

TIMESTAMP 8601 UTC	CAMS TIMESTAMP UTC	Size	Zip File Name	Entry
2018_02_07_10_55_55-08:00	2018_02_07_18_55_55	8169 BYTES	2017_12_24_00_00_00.zip	
2018_02_07_10_55_55-08:00	2017_10_28_16_12_30	399 BYTES	2017_12_24_00_00_00.zip	junk1.bat
2018_02_07_10_55_55-08:00	2016_02_15_21_29_28	5873 BYTES	2017_12_24_00_00_00.zip	junk3.bat
2018_02_07_10_55_55-08:00	2017_10_31_00_18_44	27669 BYTES	2017_12_24_00_00_00.zip	junk5.bat
2018_02_08_16_04_57-08:00	2018_02_09_00_04_57	587 BYTES	2017_12_24_00_00_01.zip	
2018_02_08_16_04_57-08:00	2016_01_10_09_04_42	374 BYTES	2017_12_24_00_00_01.zip	junk4.bat
2018_02_08_16_04_57-08:00	2016_01_10_09_04_42	331 BYTES	2017_12_24_00_00_01.zip	junk6.bat
2018_02_08_16_04_57-08:00	2018_02_09_00_04_57	102400 BYTES	2018_01_01_00_02_00.zip.001	
2018_02_08_16_04_57-08:00	2017_12_24_16_19_36	8148 BYTES	2018_01_01_00_02_00.zip.001	CAL_000525_20171224_085322_669.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_20_58	8832 BYTES	2018_01_01_00_02_00.zip.001	CAL_000526_20171224_085320_503.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_22_20	8262 BYTES	2018_01_01_00_02_00.zip.001	CAL_000527_20171224_084752_956.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_23_42	8718 BYTES	2018_01_01_00_02_00.zip.001	CAL_000528_20171224_084807_961.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_25_04	8262 BYTES	2018_01_01_00_02_00.zip.001	CAL_000529_20171224_085033_406.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_26_24	8490 BYTES	2018_01_01_00_02_00.zip.001	CAL_000530_20171224_085621_337.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_27_48	8718 BYTES	2018_01_01_00_02_00.zip.001	CAL_000531_20171224_085052_198.txt
2018_02_08_16_04_57-08:00	2017_12_24_16_29_10	8148 BYTES	2018_01_01_00_02_00.zip.001	CAL_000532_20171224_084532_929.txt
2018_02_08_16_04_57-08:00	2017_12_24_19_27_54	850932 BYTES	2018_01_01_00_02_00.zip.001	FTPdetectinfo_000525000532_2017_12_24_01_32_08.txt
		850932 BYTES	2018_01_01_00_02_00.zip.002	
		850932 BYTES	2018_01_01_00_02_00.zip.003	
2018_02_10_21_29_39-08:00	2018_02_11_05_29_39	1293976 BYTES	2018_02_09_00_00_00.zip	
2018_02_10_21_29_39-08:00	2018_02_09_21_55_00	3191808 BYTES	2018_02_09_00_00_00.zip	LaunchCapture.exe
2018_02_10_21_29_40-08:00	2018_02_11_05_29_40	1317383 BYTES	2018_02_10_00_00_01.zip	
2018_02_10_21_29_40-08:00	2018_02_09_21_55_00	3191808 BYTES	2018_02_10_00_00_01.zip	LaunchCapture.exe
2018_02_10_21_29_40-08:00	2018_02_10_20_52_30	1441 BYTES	2018_02_10_00_00_01.zip	ListUnprocessedSessions.bat
2018_02_10_21_29_40-08:00	2018_02_10_20_51_14	41706 BYTES	2018_02_10_00_00_01.zip	Submitted_LocalUpload.bat
2018_02_10_21_29_40-08:00	2018_02_10_17_10_36	39571 BYTES	2018_02_10_00_00_01.zip	UpdateScripts.bat
2018_02_10_21_29_40-08:00	2018_02_10_12_59_28	17084 BYTES	2018_02_10_00_00_01.zip	upload_queue.bat
2018_02_10_21_29_40-08:00	2018_02_10_12_59_28	17084 BYTES	2018_02_10_00_00_01.zip	upload_queue.bat

Timestamp 8601 UTC indicates a modified ISO 8601 date/time structure. It provides the UTC offset.

CAMS TIMESTAMP UTC indicates the modified date of the file, normalized to UTC.

Size is the size of the file... in Bytes.

Zip File Name is the name of a zip file. There will always be an entry for a zip file with no Entry sequence number. That represents the actual zip file.

Entry is the file within the specified zip file.

It checks a folder on the NASA server each day. It downloads any new zip files if they either (a) don't exist locally; or (b) the "UpdateScript.log" file doesn't have a matching zip file name in it. You see, the local UpdateScript.log file, in the Updates dir tracks all the changes to the system.

The workflow of the script works is like this:

24.1. It is launched without any command line arguments; therefore, it is compatible with being able to execute it by double-clicking on it. Typically, it is called daily by a scheduled task, best run an hour or so before capture starts each afternoon. Be careful setting the time of this task. Capture is not the only thing that happens during the day. You also have other scripts that run, such as archiving, uploading, status, etc. They all run at different times of the day. It might be a recommendation to disable all the tasks and scripts. It would be a good practice to launch the LaunchCapture_KILL task to stop all the running tasks. After you do the update script, you can simply reboot the computer and it should start using all the new scripts.

The files will be distributed to all the local directories identified in the "Cams2Global.ini" [UPDATESCRIPT] bat_CAMS2UPDATESCRIPT.run folder section. It will first check all the "!bat_CAMS2DIR.#!\RunFolder"] directories. It will check each of those bat_camsdir(s) as well as any "..\RunFolder" subdirs of those camsdirs.
NOTE: This is no longer the structure of AutoCams. So don't use this until it is updated.

24.2. It uses the "cams2Global.ini" [UPDATESCRIPT] section variables to control the script. Variables include:

- `[CAMS2UPDATESCRIPT]`
- `server_folder=/incoming/cams2updates/`
- `local_dir=D:\cams2_queue\Updates`
- `update_logfile=UpdateScript.log`
- `bat_CAMS2UPDATESCRIPT.runfolder.1=!bat_CAMS2QUEUE.queue_dir!\RunFolder`
- `bat_CAMS2UPDATESCRIPT.runfolder.2=!bat_CAMS2DIR.0!\RunFolder`
- `bat_CAMS2UPDATESCRIPT.runfolder.3=!bat_CAMS2DIR.1!\RunFolder`
- `//bat_CAMS2UPDATESCRIPT.runfolder.4=!bat_CAMS2DIR.2!\RunFolder`
- `And these files will be copied to the CAMS dirs.`
- `[CAMS2UPDATECAMS]`
- `bat_CAMS2UPDATECAMS.camsfile.1=LaunchCapture.exe`
- `bat_CAMS2UPDATECAMS.camsfile.2=CamsGUI.exe`
- `bat_CAMS2UPDATECAMS.camsfile.3=junk4.bat`

24.3. IMPORTANT: These downloads must be accounted for in the "?:\cams2_queue\BytesThisBillingcycle.txt" in case your system is on a metered internet connection.

24.4. Before downloading a zip file to the Pending dir from the "cams2updates/" folder on the server, it will clear out the "?:\cams2_queue\Updates\Temp" dir. Then it will download the zip file, if the file doesn't exist in the "UpdateScript.log". If it already exists there, it will not download any update zip files.

24.5. The naming convention for the update zip file will be as shown below:

2018_01_24_03_04_06.zip

The time section isn't important (unless there are more zips for that day), so you could use it as a sequence number. For example: "2018_01_24_00_00_01.zip"

24.6. After downloading the zip file, the zip file name will be added to the UpdateScript.log file. Remember, the zip file could be a split zip.

```
2018_01_24   03_04_06 UTC   0000000000 bytes   2018_01_24_03_04_06.zip
2018_01_12   02_34_02 UTC   0000000000 bytes   2018_01_24_03_04_06.zip    AutoCams.bat
2018_01_09   01_23_45 UTC   0000000000 bytes   2018_01_24_03_04_06.zip    autocams.setup.bat
2018_01_23   21_00_00 UTC   0000000000 bytes   2018_01_24_03_04_06.zip    LaunchCapture.exe
```

Then, the zip will be unzipped to the ..\Temp dir. Then, all the files in the ..\Temp dir will be added to the UpdateScript.log file. For each file in the ..\Temp dir, the log entry will look somewhat like this:

Note: The reason for using fixed length fields is to make it easier for humans to read.

24.7. It accommodates split zip files.

24.8. After updating the log file, it copies all the files in the Temp dir to all the appropriate directories.

24.9. Then it cleans out the Temp dir.

24.10. It should run it its own cmd_shell: cmd_UpdateScripts.bat.exe

OPTIMIZATION: Download a list of files first to cache locally, like "ftp.temp3.txt".

Add each of those zip files that aren't listed in the log file to the "ftp.temp.txt" command file with a "get" command.

NEEDS: Needs to update the BytesThisBillingCycle.txt file for the downloads.

About the Author:

Dave Samuels *is a computer programmer and technical curriculum developer for IBM and corporate trainer. He is an amateur astronomer and is an experienced astro-imager. He served on the board of Fremont Peak Observatory association for 13 years He has been managing global CAMS operations since 2011.*