APOLLO Housctl (Houston Control) Operation and Design

Last saved by Eric L. Michelsen

Contents

1 Introduction

Many issues are flagged in the text with "??". So search for "??" (without the quotes) to find open issues.

Using This Document

This document is for APOLLO system operators, and software developers. Operators need only chapters 2 and 3. Developers need the whole thing.

It's too hard to maintain a separate developer's document, because developers must constantly update both the operator sections and the implementation sections to agree. That's only feasible in a single document.

Open Issues

1. System too risky with environmental controls running off terminal server. Switch to DAQ or parallel port. There seems to be a high rate of failure of terminal server lead controls (~1 in 10).
2. Long standing problem with hours-long TCP timeouts. See housctl design chapter.
3. The prediction software is way too strict on agreement with the ephemeris, and this causes us to require two polynomials per night, instead of one. This is a huge pain, and should be fixed by loosening the error criterion to ~1 ns (from ~1 ps).
4. Is the resynchronization logic for the diffuser phase motor reliable?
5. The gravimeter will have its own independent data recorder off site. No need to bother houston/ICC with this. What is its recording mechanism? Computer readable?

Stuff That Needs To Be Added To This Document

Include copy of daily/polytest file.

SVN link.

Pending Software Changes

TUI

1.

housctl

1. Turn off default logging of laser serial traffic
2. housctl: fix the text of pr_read 88: status 0 when setting the PRs when laser is powered off.
3. Fix the initial GPS string to always be right.
4. Make scripts to run housctl.
5. Add statename parameter to send to TUI
6. A few times, TR position returns -141893611 or so on 'tr sync'.
7. Automatic laser and STV control.
8. Fix STV display update.
9. Make TCP timeouts be seconds.

10. Interrupt housctl to get to known state?
11. Write multiple copies of log & data files
12. Shut down CAMAC if overtemp
13. safety shutter command
14. Thread STV commands
15. Smoother Utah temperature control
16. GPS clock stats during RUN with threading, so we don't lose 1 or 2 shots.

As expected, the new feature for getting the GPS DAC stats mid-run (which is important and good) results in some missed fids. They look like laser dropouts (1 fid missing):

```
missing 1 fids, prior to shot 383
missing 1 fids, prior to shot 583
missing 1 fids, prior to shot 783
missing 1 fids, prior to shot 983
missing 1 fids, prior to shot 1384
```

But the pattern is obvious: every 200 shots one fid goes missing.

17. Add periodic power control retry on failure of WTI and maybe DAQ. Is 1 retry enough for term serv? Move critical controls to DAQ output for greater reliability (no network dependence).
18. Streamline the process of switching filenames (e.g., no longer necessary to issue readblock) when we switch targets, the spooks will be happier with its robustness. Perhaps we can name multiple block files similar to naming multiple poly files (which I'm sure we can get rid of). Then we can have housctl read the right block file when it receives a 'refl' command.

laserblock1	for target 1
laserblock2	for target 2, etc.

19. Why did housctl crash twice when temp thread blocked ? Or lost contact with terminal server.
20. Does DAQ needs mutual exclusion, because DAQ is accessed concurrently for power control and temperature data? Or does driver handle this OK?
21. renumbering thunt offset shot numbers
22. Built in self-test?
23. A weird problem prevented one of the run files from being processed by the analysis software. It turns out that the gps0 record after fid shot 2202 in 070529-062657.run had an extra (^A) character in the DOY:

```
gps0 gpstrig="^A149:06:28:49.00677"; cpserr=0
```

Don't know where this came from, but it messed things up. This could be a one-time error.

24. A new problem crops up on occasion. Don't know what to make of it yet, but it's always at the very end of a run.

```
******** WARNING *************
Number of dTWS-dFRC errors detected: 4
TWS-FRC Error: at shot 14997: 2*dfrc, dtws, twsfrcerr =
 266627604, 3498516, 63129088
TWS-FRC Error: at shot 14998: 2*dfrc, dtws, twsfrcerr =
 266627610, 3498522, 63129088
TWS-FRC Error: at shot 14999: 2*dfrc, dtws, twsfrcerr =
 266627616, 3498528, 63129088
TWS-FRC Error: at shot 15000: 2*dfrc, dtws, twsfrcerr =
 266627622, 3498534, 63129088
******************************
```

25.

housctl Change Log

This log is now continued in the SVN comments, and is no longer maintained here. Dates are executable file date, not necessarily the date when run on houston.

4/29/2008	Changed time hunting to use 'huntrun' command.
4/16/2008	New block-file format (old still works). ‘status’ updates laser block status. Fixed bug when “unblock” overlapped block-file time, which caused permission time to be wrong. Eliminated long delays when blocking during a run. ‘dphase_target’ is preserved across housctl restarts.
2/20/2008	Added the time-search feature (‘thunt’). Fixed the problem of 'topt' nonzero causing debug info on every shot in the data file. The debug option is now properly in the 'debug' settable parameter.
12/3/2007	Switch to parallel port instead of terminal server for power 9-16.
11/28/2007	Fixes WARMUP crashes. ILE temp alarm changed to t_ile_hi+3 (now 28C), rather than t_ile_hi+1 (was 26C).
11/21/2007	Added fakedata (state=13).
11/14/2007	Fixes the inadvertent power cycling of the passive PG cooling pump/fan.
11/11/2007	Escape STV display non-printing as “%XX”. Allow poly files to have day of 0, which means “today”.
10/27/2007	Eliminated “shot time out of range” on fid -1 records. Fixed ‘help’ on session 2. Normal ‘disconnect’ does not alarm. Forced disconnect now works, and generates alarm.
10/17/2007	RTD calibration. Compliance with new alarm scheme, including new Laser blocking updates to TUI. ICC command error no longer generates alarm. blockremaining/releaseremaining clean and accurate. Lots of cleanups to make reportable keywords consistent.
10/3/2007	Norens come on at WARMUP, and go off at IDLE. Norens come on during Utah heating. Fix formatting of m33/m75 response. Changed RTD name “Laser_rack” to “MV70”. Removed oscvolt_r from the cums file. Added ‘laser oscvolt up v’ and ‘laser oscvolt down v’ commands to change arbitrary volts; reports oscvolt_offset for how much the digipot is decreasing the laser voltage.
9/25/2007	Added chiller temperature records. Temperatures taken immediately on startup, avoids bogus Laser flow low alarm. Removed Laser power on from WARMUP.
8/29/2007	Swapped CAMAC and Utah_heat plugs (now 1 & 0). Changed tilt calibration.
5/22/2007	The first GPS trigger time corresponds to the first fiducial (shot 1), not the first shot = -1.
5/10/2007	Turns the IR camera on at WARMUP, and off at COOLDOWN.
4/25/2007	(Put on Houston 5/9/2007 13:50 PT) Progress diagnostics in temperature and GPS threads (thread_temp_status, thread_gps_status). Attempt/Error counters in power control (pwr_ctls, pwr_ctl_failed, pwr_ctl_first_failed).
3/23/2007	All symbolic links are now relative, and moved to “fileprefix1/” directory.
3/1/2007	Added oscillator voltage digipot control. Fixed WARMUP TR sync by setting divide ratio before enabling diffuser.

2/?/2007	Changed tr_motor_position to always return 0-3999. Fixed read_polynomial( ) to return proper status. WARMUP powers on motors first, with 1s delay, then other stuff, as a test to fix the TR sync problem. Added 'naf' and 'nafl' commands.
1/30/2007	Added 90 min idle ICC dropout from STANDBY or WARMUP to COOLDOWN. Fixed fd leak in read_polynomial( ).
1/27/2007	Removed 'circulate_laser', added dphase/TR motor pos & motion display every 0.5s during sync_diffuser. Fixed log file roll time. DI circulation check now done in all states (not just IDLE).
1/26/2007	Added TCP maxfd.
1/20/2007	Catch SIGPIPE, so dead ICCs don't kill housctl. Added 'unblock' command. Fixed fd leak on errors in wti_util.c and ts_util.c. Added more ICC session diagnostic info.
1/5/2007	'alarms' and 'alarms_unack' come out on all exc0 records. Removed stdout/stderr buffering, so nohup.out is up-to-the-second, for better diagnosis of problems.
1/4/2007	Added 'daq' diagnostic command. Start of operator power override. Added powerstate of -1 for unused ids. Fixed readblock day of year problem. Added dphase to 'tr' diagnostic. Made 'set alarms_unack...' echo par0 to log file. Changed DAQ library to only retry when reads NaN. Increased power-on delays for TR motor to try to fix TR sync failure in WARMUP.
11/16/2006	symbolic links in data directory to log file are now relative. Temp safety on ILE temp when cabinet RTD fails. More ICC session debugs: ICC connects/disconnects now `printf()` to nohup.out. t0, tf now 16 decimal places. Added SecFocus memory parmeter. On log-file rollover, invalidate DRK, FLT, airtemp, pressure, humidity, SecFocus, reflector, guideOff, boreOff[0], axePos[0], slewtarget[0]. Reduce lost photons during RUN when taking temperatures. Fixed "mirror queued message" to be i= (info), rather than text= (error). runfidgw defaults to 7 instead of 6. WARMUP increased TR motor power-on delay from 1 to 3 sec, to see if it makes TR/diffuser sync work.
8/10/2006	New directory structure. Processes any `debug=`, `doptions=`, or `fileprefix1=` command line arguments before opening log file.
7/2006	Dark/Flat records now in data file. Normalize DARK/FLAT data when taken.
7/14/2006	Allow any whitespace (e.g. tabs) to lead housctl.blk lines.
6/20/2006	Exceptions now come out to ICC session 2. Removed bogus FRC mismatches from shot -10. Added STV commands. Removed laser statusline reporting. bolopos was put in a while ago. Updated lots of operator messages to 'i=xxx', and errors to 'text=blah'.
6/19/2006	STV left on with laser flashing. On in WARMUP, off in COOLDOWN. DARK/FLAT/FIDLUN/STARE set TR as needed, all but FIDLUN leave clear; FIDLUN leaves dark. RUN still moves TR to clear space. Added CALTDC to start and end of RUN.
6/~3/2006	Added wildcards to 'get'; modified rxx scale factors; added delay to laser 'activate' button.
5/11/2006	Programmable resistor for laser power control: laser power 0/1, ampdelay_low, ampdelay_high
4/~25/2006	Threaded rxx/rxy commands
4/1/2006	No foolin': added fakerun and caltdc
3/31/2006	Allow 2 simultaneous ICC sessions.

3/28/2006	Fixed 'status' output for unknown state 'powerstatus' variables.
3/16/2006	Log files now start with parameter list, like data files.
3/14/2006	Added 0x05 startup reply to laser.
3/9/2006	Added powerstate variable. Made laser keep alive/display thread.
3/7/2006	STARE moves TR to clear, if TR powered on. Added 'tr' command. Fixed diffuser phase tracking in RUN. Added symbolic link creation. Cleanup ICC connection messages. Enable diffuser motor after all camac_Z( ). dphase_target defaults to 850.
3/4/2006	changed to fid0 w/ dphase, & lun0. No exit from COOLDOWN/passive-cool. Split housctl ICC code into housicc.c. fid/lun format comment in fidlun data files.
3/2/2006	Moved STV on/focus from end of RUN to start of STANDBY. Changed STV power-on delay from 1->2 s. Changed to t_utah_center, t_utah_push, & t_utah_limit. Changed to char guideOff[32], boreOff[32], axePos[64].
3/1/2006	TR motor to reflective in FIDLUN. Moved "switch to TR laser zap" to right after gate widths set, to avoid diff tracking conflict.
2/27/2006	Laser block file no longer needs the "Z" after times.
2/25/2006	Changed all times to UTC. Log file includes comment stating such. Changed log file roll to 20:00 UTC always (12:00pm PST); each log file now spans 24-hours, regardless of housctl start time. Data files now named with UTC (normal log files unaffected). housctl build date/time is local time, as defined by C-language standard.
2/19/2006	Changed initial airtemp, pressure, humidity to -99. Poly files and laser block file now come from daily/ directory. Laser blocks by default, needs override housctl.blk. Fixed laser blocking off by 1 year. Alternating gate widths for fiducial/lunar. Added pulse energy measurement. Major laser control updates. Fixed finding poly file in RUN.
2/18/2006	Added 'flat' command. M75 turns on with 'laser preprun' command. New filenames, same old directories. Separate run files, w/o WARMUP. Diffuser synchronization across all states. TR initialization. Removed shot #'s -2. Diffuser phase in par0 after each fid in data file. Added statusline reportable. Added GPIB semaphore. Laser block file lines with 'AZ' or 'TARGET' in them get logged.
2/6/2006	Alarms in hex, READONLY parameters.
1/14/2006	Day of years works through 12/30/2008. Chillers powered through numbers 17 & 18. Fixed 'info' command. M33 used when dome_air > t_utah_low - 6, or when CAMAC is on.
12/5-9/05	Sends last drk0, if any, on 'status'. Added 'camac' command. Fixed initial detection of power states 0-7. Added bunch new RTDs. Polynomials come from polys/ directory, but I hope to change that to daily/ soon.
12/3/2005	laser & STV turned on in WARMUP; FIDLUN/STARE/DARK return to either IDLE or STANDBY, depending on whether IDLE or WARMUP was last entered; 50 MHz counters enabled earlier in RUN; DARK positions TR motor to block telescope light; read polynomial files from "`poly/*`"; changed apdtoffs name to apdtofpd & updated with latest numbers; removed 'camac 0/1' command (use 'power'), added 'camacz' command.
11/26/2005	Added Alarm system. Added power control records, 'pow0'. Added retry on terminal server power controls. Added APD time offsets to fast photodiode, "apdtoffs".
11/23/2005	added diffuser phase tracking, retry on TS power controls, apdtoffs, pow0 records

11/12/2005	Added weather and mirror move functions.
11/3/05	Added "circulate_laser" function to avoid freezing.
10/15/05	Fixed "Laser enabled" msg on every shot. Don't turn on STV at end of FIDLUN.
10/15/2005	housctl.readblock: Added readblock command. Changed default dskew to -48 ns.

Simulator

1. Separate prepulse rates for fids and luns? Does it matter?

Prediction Software (mkpoly.c)

1. Fix round off error in start time
2. Make files more self-documenting
3. Fix instability problems
4. Eliminate the need for sections: loosen accuracy to 1ns. When I did my polynomial fit to the simulated data points, I couldn't use 'double' for the regression matrix sums. I had to use 'long double', because summing over 6000 points causes more than 3 digits of precision loss, which does not allow for ps accuracy. After the sums, I truncate them to 'double' and do the linear regression equation solutions in 'double' and all is well to ~1 ps.
5. Save the raw ranges in "index time range" file, for easy import to housctl on-the-fly poly fits.

References

APOLLO home page: http://physics.ucsd.edu/~tmurphy/apollo/apollo.html

APOLLO observing manual:
http://cfa-www.harvard.edu/~jbattat/apollo/docs/observingManual/observingManual.pdf

APOLLO documentation: http://physics.ucsd.edu/~tmurphy/apollo/doc/ (not linked from home page).

housctl svn: svn://svn.apo.nmsu.edu/Apollo/Houston/trunk/

Russell has written a first draft manual on how to write an Instrument Control Computer. It is available at: http://rowen.astro.washington.edu/ICCManual/. This should be served at APO, once Craig Loomis has a chance to review it. P.S. Craig: the statement that Reply ID is obsolete and should not be supported by new instruments may be controversial. Other than that, I think it's mostly a matter of fact checking and filling out some details to come up with the final version.

Here are some example command sets that are simple and possibly worth emulating:

- DIS <http://tycho.apo.nmsu.edu:81/DIS/DIS_Commands.html>

- expose and tlamps: see links from <http://tycho.apo.nmsu.edu:81/MC2/>

```
From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Thursday, March 16, 2006 10:39
```

I've updated my tui install notes, available at:
http://www.cfa.harvard.edu/~jbattat/apollo/TUIInstallNotes.txt

Also, I've updated the notes in the .zip file that includes all necessary install files for a TUI install on windows: http://www.cfa.harvard.edu/~jbattat/apollo/TUIInstallation.zip

Info about TUI (APO's telescope user interface) can be found here: http://www.astro.washington.edu/rowen/TUIHelp. To get a copy of the program, read "Installation". (This is a copy of the html help built into TUI).

The hub tries to do very little other than authorize users and route commands. Authentication and some of the scripts are documented at http://tycho.apo.nmsu.edu:81/MC2/, and in particular, the link Authentication. The expose info is a bit outdated (especially since it is being overhauled right now).

The TCC is the Telescope Control Computer. It also lives at APO in the computer room. See http://www.apo.nmsu.edu/Telescopes/TCC/TCC.html for info on the commands and expected replies.

Lantronix Terminal Server Manual http://www.lantronix.com/pdf/ets_ref.pdf

2 Using housctl (Houston Control)

"houston" is a computer. "Houston Control" or "housctl"is a program that runs on houston. This section describes how to run and use Houston Control (housctl). 'housctl' is one program that runs continuously, so normally, you don't need to start it. housctl requires houston's Unix clock to be synchronized with UTC. A cron job should do this every hour, but the laser blocking times depend on it, so you should verify it.

Overview of Typical Operations

Housctl is usually run from TUI. TUI provides many high-level operations to operators, which TUI translates in to low-level housctl commands. Some operations, however, require operators to directly enter housctl commands. All housctl commands and parameters are case sensitive. Commands have the general format (as seen by the operator):

```
command parameter parameter ...
```

There are usually only 1, 2, or 3 parameters.

Operators command housctl into one of several states (e.g., idle, warmup, run, cooldown, ...).

When housctl enters WARMUP state, in anticipation of RUN, it increases the rate of some measurements (e.g., temperature), for better environmental data when collecting lunar returns.

The normal sequence for making multiple runs is:

warmup
run (automatically goes to STANDBY when done)

... repeated once for each run
cooldown (automatically goes to IDLE when done)

On entering RUN, housctl creates a data file, and copies `housctl.hed` into the data file. You can set `housctl.hed` to be a set of REM records to better describe the contents of the data file. You can put any kind of documentation, or even data, that you want in it. Each RUN gets its own data file. Note that all "log" events are also entered into the data file, so the data file alone contains a complete record of all events during the measurement period.

Changing the Length or Stopping an Operation Manually

Operators can change the length of a run, while running, by setting 'nruns' to a different value, e.g.

```
set nruns=5000
```

You can stop an operation (run, flat, dark, stare, fidlun, etc.) with

```
standby
```

Killing housctl

Killing housctl freezes everything in its current state, but the environmental page can't see any temperatures or flows if housctl is not running to fill in the log file.

Kill the running housctl with ($ is normal bash prompt, # is root prompt):

```
$ cd /home/apollo
$ su
<enter the root password when it asks>
# ps aux|grep housctl
root      5673  0.0  0.2 39316 1232 ?       S<L  Mar01   1:57 ./housctl
root      5677  0.4  0.2 39316 1232 ?       SNL  Mar01  59:10 ./housctl
root      5678  0.0  0.2 39316 1232 ?       SNL  Mar01   0:00 ./housctl
root      5679  0.0  0.2 39316 1232 ?       SNL  Mar01   0:12 ./housctl
eric      6314  0.0  0.1  1728  592 pts/5   S    03:18   0:00 grep housctl
```

The number after 'root' in the first line is the process ID (PID) we need, in this case, 5673.

```
    # kill 5673
```

Note that 'set state=0' is designed to turn everything off for hardware operations such as rewiring, or when major systems will be disrupted. It specifically leaves on houston (avoiding suicide), and the GPS clock.

Ideally, restarting housctl should turn everything back on, though we don't do this very often, so I wouldn't be surprised if something is overlooked. If so, we should fix housctl to turn everything on that needs it.

Also, housctl should expect things to be out-of-sorts for at least one environmental measurement cycle, and shouldn't turn lots of stuff on for no real reason. I know there was a problem for a long time with the flow meter causing a laser power cycle at start up, but that should be fixed now. Small things like turning on the Noren fans for a minute or two don't matter, and aren't worth considering.

Restarting housctl After a Problem

Sometimes after testing (or some unforeseen termination or problem), you need to restart housctl itself. For normal operation, start housctl with no parameters. However, it must be set to run continuously, even after you log out. There are 2 ways to do this:

```
    bash> nohup ./housctl &
    [1] 20632                      system responds with process id
```

For debugging, there's another way described below. 'nohup' has the advantage of appending stdout & stderr to the file 'nohup.out', which can help diagnose crashes.

After starting housctl, you need to increase its priority, such as

```
    ps                find the first housctl process id, call it x
    su                priority increasing requires root privilege
    renice -20 x      increase housctl's priority to the maximum
    ps lf             verify that priority is set properly
      F   UID   PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY        TIME COMMAND
    ...
    100     0   944     1  -1 -20 38964  940 nanosl S<   pts/5      0:00 ./housctl
    040     0   947   944  13   5 38964  940 do_pol SN   pts/5      0:00 ./housctl
    040     0   948   947  13   5 38964  940 wait_f SN   pts/5      0:00  \_ ./housc
    040     0   949   947  13   5 38964  940 nanosl SN   pts/5      0:00  \_ ./housc
    exit              don't be a root hog
```

You can set any parameters from the command line (the same parameters as the ICC "set" command). E.g.

```
    housctl debug=1 nruns=10000 &
```

For debugging, you might want to start housctl like this:

```
    bash> ./housctl &
```

Debug here, where you can see the normal stdout and stderr.

```
    bash> disown
    bash> jobs
```

Verify with the "jobs" command that there are no processes belonging to this shell session (note that **ps** will still show the housctl processes). You can still increase its priority, as above.

Reverting to an Older Version of housctl

Whenever I change housctl on houston, I save a copy in houston:/home/apollo/bin directory, as "housctl.something" where "something" is a mildly descriptive word of the change in *that* version of housctl (i.e., the word matches the binary).

The new binary of housctl goes directly in houston:/home/apollo/housctl. You can always see all the backups with

```
ls -lt bin/housctl.*
```

which sorts them by time, and shows you the timestamps on the binaries.

To run an old version: First, kill the running housctl (see above instructions).

Now the old housctl is dead. Choose the old one you want to run by date by looking at:

```
   # ll -t bin/housctl.*
   -rwxrwxr-x    1 eric     llr        236568 Mar  1 19:17 bin/housctl.oscvolt
   -rwxrwxr-x    1 eric     llr        235128 Feb  1 05:52 bin/housctl.warmup
   -rwxrwxr-x    1 eric     llr        233803 Jan 26 22:42 bin/housctl.maxfd
   -rwxrwxr-x    1 eric     llr        231934 Jan  5 22:13 bin/housctl.alarm
   -rwxrwxr-x    1 eric     llr        231752 Nov 17 23:52 bin/housctl.relsym
   -rwxrwxr-x    1 eric     llr        231279 Aug 11  2006 bin/housctl.newdir
...
```

This is a list of the saved versions in reverse time order. The latest one before the run date you want to emulate is the one to use. For example, if you want the same housctl as on Jan 9th, the housctl in effect at that time was housctl.alarm dated Jan 5.

Now follow the instructions above for restarting housctl after a problem, but substitute 'bin/housctl.alarm' (or whichever one you want) for './housctl':

```
# nohup bin/housctl.alarm &
[1] 20632                         system responds with process id, e.g. 20632
# renice -20 20632                increase housctl's priority to the maximum (use the new PID)
# ps lf                           verify that priority is set properly
# exit                            don't be a root hog
```

Using housctl To Take Data

To actually record any data (real lunar shots, STARE, DARK, FIDLUN), you must send housctl through a sequence of states.

Note that only the real lunar data files include log records as well. By design, Fidlun and Stare data files do *not* include log records (why is that??).

Using housctl To Take Real Lunar Data

Real lunar data is the only data that uses WARMUP, STANDBY, and COOLDOWN states. For a single run of real lunar data:

`warmup`	enter warmup state, closes any open data file, read polynomial file, power on CAMAC crate, increase temperature rate to every 10s, start & synchronize TR & diffuser motors.
`run`	Open data file named *`yymmdd-hhmmss`*`.run`, start TR motor, firing laser 'nruns' times, take data. When done, closes data file, stops TR motor, & goes to STANDBY state. After 90 min of no ICC commands, STANDBY automatically goes to COOLDOWN. Usually, you will command 'cooldown' sooner than that.
`cooldown`	enter COOLDOWN state. APD fan, laser rack, & M75 chiller run briefly. ~30 total min of cooling, then moves to IDLE state.

For 2 or more runs of data, recorded in separate data files:

`warmup`	enter warmup state, closes any open data file, read polynomial file, power on CAMAC crate, increase temperature rate to every 10s, start & synchronize TR & diffuser motors.
`run`	Open data file, start TR motor, fire laser 'nruns' times, take data. When done, closes data file, stops TR motor, & goes to STANDBY state. After 90 min of no ICC commands, STANDBY automatically goes to COOLDOWN. Usually, you will command 'cooldown' sooner than that.

`run`	issue 'run' again to start a new data file and take data. Repeat 'run' and waiting for STANDBY, any number of times.
`cooldown`	enter COOLDOWN state. APD fan, laser rack, & M75 chiller run briefly. ~30 total min of cooling, then moves to IDLE state.

Using housctl To Take Stare Data

Stare data does *not* use WARMUP, STANDBY, and COOLDOWN states. From IDLE state:

`stare`	enter STARE state, create data file named `yymmdd-hhmmss.str`, power on CAMAC crate, take STARE data. STARE continues until commanded back to IDLE
`idle`	close data file.

Using housctl To Take Dark or Flat Data

Dark data does *not* use WARMUP, STANDBY, and COOLDOWN states. 'flat' state is loosely named, since the operator chooses what to point the APDs at, which may or may not actually be uniformly lit. From IDLE state:

`dark`	enter DARK state, power on CAMAC crate, move TR to dark patch, take data (same method as STARE). After about 15 s, record data in current log file. DARK transits automatically back to IDLE state. DARK data files are `*.drk`.
`flat`	flat is the same as dark, except it rotates the TR optic to a clear patch. FLAT files are `*.flt`.

With DARK and FLAT data, that TUI can subtract a background dark count, and also take care of pixel-to-pixel gain variations, to make the APD grid more accurately depict illumination. TUI can then treat hitgrids as (new - dark)/(flat - dark) before displaying.

Using housctl To Take FIDLUN Data

FIDLUN state is mostly meant to detect fiducials, at a fixed pulse rate. First, be in WARMUP or STANDBY. The "lunar" gates are just sandwiched in between the laser shots. From IDLE state:

`warmup`	enter warmup state, closes any open data file, power on CAMAC crate, increase temperature rate to every 10s, start & synchronize TR & diffuser motors.
`fidlun`	enter FIDLUN state, create data file `yymmdd-hhmmss.fid`, power on CAMAC crate. FIDLUN continues until 'nruns' shots, or commanded to another state. Returns to STANDBY or IDLE (whichever it came from).

Using housctl to Calibrate the TDC While Flashing Laser (LASERCAL)

LASERCAL state measures the effects of the laser fire (EMI) on the clock/Booster signal. Tom says it requires a special cable be put in place on the system, but I don't know why that's needed. Lasercal works as follows:

- The laser fires, triggering the fast photodiode
- FPD alerts the ACM
- ACM produces a gate of width parameter W
- As a by-product (LUN_START enabled), a START request goes to the Booster
- The normal STOP request accompanies the end of the gate
- W is cycled between 1, 2, 3, 4, 5
- one minute of this --> 240 events for each gate width

Thus we obtain a TDC calibration very much like the normal calibration, but triggered by the laser and thus operating during a very noisy time. Thus we can investigate the effect of the laser fire on the clock pulses. This is important to understanding our fiducial timing performance.

Processing is a mix of FIDLUN and CALTDC. The APD is disabled. The internal laser shutter remains closed. LASERCAL also does the FIDLUN trick of producing fake "lunar" gates, so we get an in-situ calibration of the TDC in a quiescent state for direct comparison. LUN_START enabled is the chief difference to FIDLUN.

Using housctl to Emulate a Real Run (FAKERUN)

The 'fakerun' command runs the same code as 'run', except it keeps the laser firing disabled. 'fakerun' can test most functions of a real run other than the laser. It can be used for testing housctl, TUI, and other subsystems.

To use 'fakerun', you must use a test reflector of '-1', provide a valid prediction file, and must release the laser (in the Space Command sense). For the prediction, I keep a file houston:/home/apollo/daily/polytest with a simple straight-line polynomial in it. Since the actual prediction is irrelevant, this file predicts a time near 2.5s, increasing by ~10 ns per shot. The file is valid 24 hours a day, and sets the day=0, which tells housctl that it is valid every day. This polytest file is this:

```
0
0.000000000000000
0.999
0.000215925
0.0004956036
185.000000
5
2.616835312435575
25.140172
2
2.5
0.01728
```

To release the laser, even though it won't fire, use 'unblock 120' (2 hours). A complete sequence might be this:

```
info
:
exc0 2007-11-09T19:48:01 severity=3; alarms=0x0; alarms_unack=0x10000;
text="Shot time out of range: 313.825021 not in [309.472222, 309.625000]"
0 i dayofyear=313.8250214467593; rtt=2.5
:
refl -1                    disable automatic search for poly file
exc0 2007-11-09T19:51:38 severity=3; alarms=0x0; alarms_unack=0x10000;
text="No valid prediction file"
0 f g="Can't find polyfile"
readpoly daily/polytest          read the prediction parameters
info                       verify no "shot time out of range" exception
warmup                     power up systems, excluding the laser. You may leave the laser
                           powered off.
set nruns=5000
unblock 120                allow laser firing
fakerun                    enter RUN state, but keep laser disabled
cooldown                   shut down normally
```

Using housctl to Generate Real-time Fake Data (FAKEDATA)

The 'fakedata' command generates fake data real-time (20 shots/second). There is no laser firing involved, and no need to unblock the laser. Set 'topt' as a 2 digit number: the 10's digit is the number of fiducial APD events per shot, the units digit is the number of lunar APD events per shot. E.g.

```
set topt=12                one fiducial + 2 lunar APD events per shot.
set nruns=5000
warmup
set state=13               Soon to be fixed with a 'fakedata' command
```

FAKEDATA always generates the fiducial fast photo-diode (FPD) event in channel 15. The fake FPD is always TDC value 2000, and each channel's TDC value is (2000+channel#). Operators may change 'topt' on the fly.

Shutting Down housctl

This is *not* shutting down for the night; use 'cooldown' for that. Shutting down housctl is meant to precede serious maintenance, when major systems will be disrupted. Shutting down housctl cleanly stops *all* functions, including temperature regulation, and powers everything off (except those things we can't remotely power back on, such as the bolometer). It is generally advisable to have housctl shutdown only for short times, so the system maintains proper temperature. To shut down housctl:

`set state=0` Stop all functions and hardware (e.g., TR motor), and power down all devices.

Handy housctl Commands

`rem` The 'rem' command simply adds a comment with a timestamp to the log file, and any open data files. You can put whatever text you want on it, e.g.,

```
rem At this point, the laser rack caught fire
```

Note Whenever something unexpected happens, it's a good idea to enter a 'rem' command describing the anomaly. This allows later analysis of all the logs to help find the problem.

`get` The 'get' command accepts wildcards, which helps if you're not exactly sure of the name of a parameter. You can guess a part of it, and surround it with wildcards. E.g.,

```
get *err*
0 icc0 2006-06-05T17:02:16 (1) get *err*
0 i ts_errorstr="";    comment="term serv error string"
0 i wti_errorstr="";    comment="IP switch error string"
0 i tcp_errorstr="";    comment="TCP lib error string"
0 i las_errorstr="";    comment="laser lib error string"
0 i chil_errorstr="";    comment="chiller lib error string"
0 i pwr_errorstr="";    comment="power lib error string"
0 i pr_errorstr="";    comment="Programmable resistor error string"
0:
```

`set` besides the use of 'set' to set parameters, there are several "memory parameters," which housctl ignores. Setting them makes an entry in the log/data files, and you can 'get' them. Our conventions dictate that operators/TUI set these to document important parameters. They are:
`airtemp, pressure, humidity, guideOff, boreOff, axePos.`

`runnocal` for testing: enters run state, but does *not* run the CALTDC and the beginning and end of the run.

Forcing errors for testing: Here are some ways:

1. Issue an invalid ICC command:

```
432 xyz
```

2. Issue an 'refl' without valid polynomial files. There won't be any files valid for those days we aren't shooting:

```
refl 1
```

Disconnecting a hung ICC session:

Sometimes the hub or network disconnects an ICC session, without closing it properly. Housctl cannot detect this for a few hours (see "Open Issues": it's due to the Linux IP stack implementation). You can use the other ICC session to force housctl to disconnect a dead session. For example, if session 1 is hung, connect TUI or Telnet as ICC, and you will be session 2. Then:

```
disconnect 1
```

This will kill session 1, and free it up.

STV Commands

STV commands give you access to the STV virtual keyboard and display. Each STV command has two forms: a word form that may be easier to remember, and a 1-character form that matches the 'stvadam' 1-character commands. For example
`stv focus` and **`stv f`** are equivalent.

Note that the 1-character abbreviations are *case sensitive*: 's' and 'S' are different.

The STV LED display is 2 x 24 characters. All STV commands reply with the stv_display parameter, e.g.

```
    974 : stv_display="Filter=Open   Please set:+12.8°   70%   Date/Time"
```

stv_display is 49 bytes long, with the two lines separated by a colon (similar to las_display).

`stv`	show STV status, and help

`stv focus`
`stv f`
	enter `[focus]` mode: fast-frame low-resolution video

The following up down left right abbreviations follow the keyboard pattern of W/S=up/down, A/D=left/right. Each takes an optional repeat count, e.g. '**`stv right 7`**' is equivalent to pushing the 'right' button 7 times.

`stv left [n]`
`stv a`
	pushes the `[left-arrow]` button for setting parameters

`stv right [n]`
`stv d`
	pushes the `[right-arrow]` button for setting parameters
	Note that 'd' is NOT down

`stv down [n]`
`stv s`
	pushes the `[down-arrow]` button for setting parameters
	Note that down is 's', NOT 'd'

`stv up [n]`
`stv w`
	pushes the `[up-arrow]` button for setting parameters

`stv file`
`stv o`
	pushes the `[File Ops]` button for image files

`stv image`
`stv i`
	pushes the `[image]` button for image files

`stv parm`
`stv p`
	pushes the `[parameter]` button for setting parameters

`stv value`
`stv v`
	pushes the `[value]` button for setting parameters

`stv setup`
`stv S`
	pushes the `[setup]` button for setting parameters

`stv hairs`
`stv h`
	pushes the `[cross-hairs/display]` button to toggle cross-hairs

`stv int`
`stv I`
	pushes the `[interrupt]` button to reset the STV

Laser Commands

Laser commands give you access to the laser virtual keyboard and display. Operators can control the laser, when houston is physically plugged into the CU-601 laser control unit, with the 'laser' command.

'laser' commands also give you access to the laser virtual keyboard, but normal operation should not require any manual button pushes. Higher level commands are preferred.

The 'laser' command has several subcommands. Every laser command ends with an update of 'las_display', the string which mimics the control-box display. A typical sequence of operations for shooting the moon is:

`warmup`	enter warmup state, closes any open data file, power on CAMAC crate, increase temperature rate to every 10s, start & synchronize TR & diffuser motors.
`laser powerup`	turn on laser rack, cycle keyswitch
`laser warmup`	close cavity shutter, start flashlamps (PGM 1)

Wait ~20 min

`laser preprun`	switch to PGM 2, external trigger
`run`	enter RUN state and take data

Detailed descriptions of laser commands:

`laser` show laser status, and help:

```
0 i help="laser [powerup, warmup, preprun, stop, cw, ccw, reset, keycycle,
  keyoff, keyon, start, act, shutterclose, shutteropen,
  ampdelay, ampvolt, oscvolt]"
0 i las_display="...................:.....................:...."
0 i ampdelay=62;  ampvolt=-1;  oscvolt=101
```

`laser powerup`	turn on laser rack, and cycles the keyswitch. Waits for "SHOT COUNT" on display (gives error if doesn't come up).
`laser warmup`	activates PGM 1 (closes cavity shutter), and flashes flashlamps.
`laser preprun`	activates PGM 2, and opens cavity shutter. Laser is now ready for external triggering.

The following commands are diagnostics, and you don't normally need to use them:

`laser keyoff`	turns off keyswitch
`laser keyon`	turns on keyswitch
`laser keycycle`	turns off keyswitch for 3 sec, then turns on.
`laser stop`	stops a running program (just like STOP button).
`laser start`	starts current program (just like START button).
`laser shutterclose`	closes the cavity shutter.
`laser shutteropen`	opens the cavity shutter.
`laser cw`	rotates second harmonic generator (SHG) clockwise
`laser ccw`	rotates second harmonic generator (SHG) counter-clockwise
`laser code` *xx*	sends the hex code *xx* to the laser as a button push
`laser activate` *n*	activates program *n*
`laser power 0`	switch amplifier delay to low power
`laser power 1`	switch amplifier delay to full power

You can see all the digipot settings with

```
get osc* amp*
0 i oscvolt_r=101;    comment="Laser oscillator Digipot ohms"
0 i ampdelay_low=12696;    comment="Laser low power amp delay"
0 i ampdelay_high=62;    comment="Laser full power amp delay"
```

You can manually set the amplifier delay digipot resistance to any ohmage with this:

```
    set ampdelay_low=10000
    laser power 0
or

    set ampdelay_high=100
    laser power 1
```

You can then nudge around from there. Note that 'set'ting the variable by itself won't change the digipot setting.

laser ampdelay up	amplifier delay up
laser ampdelay down	amplifier delay down
laser ampdelay *n*	set amplifier delay resistor to *n* ohms
laser oscvolt up	oscillator voltage up ~5 volts
laser oscvolt up *v*	oscillator voltage up ~*v* volts
laser oscvolt up30	oscillator voltage up ~30 volts
laser oscvolt up40	oscillator voltage up ~40 volts
laser oscvolt down	oscillator voltage down ~5 volts
laser oscvolt down *v*	oscillator voltage down ~*v* volts
laser oscvolt *n*	set oscillator voltage resistor to *n* ohms
laser ampvolt up	amplifier voltage up
laser ampvolt down	amplifier voltage down

All the commands in the above group return the resistance value in ohms.

bolo 1	move bolometer in, sets 'bolopos=1'
bolo 0	move bolometer out, sets 'bolopos=0'
unblock *n*	unblocks the laser for *n* minutes. When the time is up, housctl rereads the current laser block file from the beginning, thus resetting back to the proper state determined by the file. 'n' is clamped at 240 min (4 hours). You may issue overriding 'unblock' commands at any time (see 'unblock 0' below). Note that 'unblock' *cannot* be used to read a *new* laser block file; you must use 'readblock' for that.
unblock 0	terminates any active 'unblock' command, and immediately reverts to the currently open block file. Note that 'unblock' *cannot* be used to read a *new* laser block file; you must use 'readblock' for that.
las_display	is a string variable that contains the entire laserbox display information: 2 lines of 20 chars, plus the 4 LEDs. It has this format:

```
    las_display="....................:....................:...."
    las_display="This is line 1.  OK?:This is line 2 below:rgG-"
```

It is fixed length: 20 chars of the first line, a colon (which need not be displayed in TUI), 20 chars of the 2nd line, a colon, and 4 chars for the 4 LEDs as seen on the laser box (L-R): charging, end-of-charge, shutter-open, Q-sw. active. The dots indicate the location has not been set by the laser, so it's state is unknown.

The LEDs are individually coded: when all on, they would be 'RGGO' (for red, green, green, and On) [I have never seen the Q-sw. active LED illuminated, so I don't know what color it is]. The LED codes are as follows:

`.`	(dot) the character is unknown, because the laser has never written it
`-`	the LED is stable off
`o`	the LED is off but has toggled in the last second
`r or g`	the LED is on (red or green) but has toggled in the last second
`R or G`	the LED is stable on (red or green)

m33 and m75 Chiller Commands

Operators can control the m33 and m75 chillers from the ICC command line. There are two identical commands, ‘m33’ and ‘m75’ that control the chillers. Each command takes a subcommand, and possibly an argument. The commands are shown below for the m33, but are identical for the m75:

`m33`	displays chiller status: protocol_version, status (0x100 = on), temperature, setpoint, alarm lowlimit, alarm highlimit
`m33 off`	turns off chiller
`m33 on`	turns on chiller
`m33 setpoint` ***`t`***	set setpoint temperature in deg C, ‘t’ forced integer by unit. Unit seems to enforce a lower limit of 5 C.
`m33 lowlimit` ***`t`***	set alarm lowlimit temperature in deg C, ‘t’ forced integer by unit
`m33 highlimit` ***`t`***	set alarm highlimit temperature in deg C, ‘t’ forced integer by unit

TR Motor Commands

Operators can control the TR motor from the ICC command line. Each command takes a subcommand, and possibly an argument:

`tr`	return motor status, including position measured twice, to see if it’s moving
`tr dark`	move motor to opaque patch on TR glass
`tr clear`	move motor to clear patch on TR glass
`tr stop`	stop motor
`tr` *`p`*	move motor to encoder position *p*
`tr sync`	synchronize diffuser motor to TR motor and dphase_target
`tr speed` *`s`*	set motor speed to *s*

Diffuser tracking occurs, presently, only during RUN, and once at WARMUP. It appears that some motor actions seem to accumulate diffuser phase errors We don’t know why this would happen, but it might be due to the TR motor hunting (oscillating); our diffuser logic moves the diffuser forward for all TR encoder changes, forward or backward, so any TR oscillation would drive the diffuser out of phase. If you accumulate enough diffuser phase error, then the TR clear space won't work because the diffuser may block light. You can force a re-sync with ‘**`tr sync`**’. Note that issuing WARMUP when you are already in WARMUP has no effect, and does *not* re-synchronize the diffuser.

housctl "parks" the TR motor in a clear spot (position 2000) at WARMUP, and after all non-IDLE states. housctl carefully enables the diffuser motor to keep the diffuser phase valid between runs, and so will park the diffuser in a clear space, as well.

Velocity Offset (Rx) Mirror Control

housctl has several commands & parameters for controlling the velocity offset mirror (aka Rx mirror). These commands return '>' (queued) status immediately, and a final ':' status when the command completes. After the '>' status, you can issue any other ICC commands while the velocity mirror move is in progress, except you can't issue another velocity mirror move until the pending one completes. housctl returns 'f' immediately for any conflicting mirror move request. All commands final response is something similar to

```
27 i rxxpcum=n; rxxncum=n; rxypcum=n; rxyncum=n
par0 vposx=vx; vposy=vy
27 : text="Mirror command done"
```

As always, housctl puts any “par0” record in log (& data) files.

`rxx` *`offset`*	units are float-pt arcseconds, frame is Rx mirror frame (NOT az-el). Moves the mirror by ‘offset’ arcseconds.
`rxy` *`offset`*	same as ‘rxx’, but in y-direction.
`vnudge` *`dx dy`*	moves mirror by dx, dy, same units as ‘rxx’. Equivalent to ‘`rxx dx`’ followed by ‘`rxy dy`’
`vmove` *`vx vy`*	move to mirror position vx, vy, same units as 'vtarget'.
`vtarget` *`vx vy`*	units are float-pt arcseconds, frame is Rx mirror frame (NOT az-el). For now, this command does NOT move the mirror, but just records the target offsets.
`vcalibrate`	set the current position as (vtargetx, vtargety). Note that by definition, after a "vcalibrate" command, (vposx, vposy) = (vtargetx, vtargety).

"Getable" parameters are:

```
vtargetx  vtargety
vposx     vposy
rxxpcum   rxxncum      rxypcum      rxyncum
```

DAQ Commands

There is a simple DAQ diagnostic, ‘daq’. It has two forms:

`daq`	display all 64 DAQ voltages, read in single-ended mode. The output cannot be seen from TUI; you can see it from a human telnet session.
`daq` ***`n`***	display DAQ channel *n* voltage, read in single-ended mode. This *can* be seen from TUI.

Power Commands and Power Override

`power` ***`dev value`***	set power of device ‘dev’ to 0=off, 1=on, 4=force-off, or 5=force-on. E.g.
`power 0 1`	turns on the CAMAC crate

The power codes are:

```
// Comments below include plug ID (1a, 2d, ...)
UTAH_HEAT        0      // 1a Utah heater Power
CAMAC_PWR        1      // 1b CAMAC & 2d Booster Power
```

```
    APD_FAN          2      // 1c APD cooling fan
    NOREN_FANS       3      // 1d Noren Heat Exchanger Power
    APD_PWR          4      // 2a APD and 2b photodiode Power
    MOTOR_PWR        5      // 2c diffuser, 3a TR motor, & 3b New Focus
    STV_PWR          6      // 3c STV camera

    DI_PUMP_PWR      9      // Auxiliary DI pump
    ILE_EXHAUST      10     // Large exhaust fan for ILE
    PG_PASS_COOL     11     // Heat exchanger and pump for PG loop
    IR_CAMERA_PWR    12     // Infra-red airplane spotter
    LASER_PWR        13     // Laser 3-phase power

    M33_PWR          17     // M33 (Utah) chiller
    M75_PWR          18     // M75 chiller

    HOUSTON_PWR      21     // Houston power on IP switch
    FLOW_MTR_PWR     22     // Flowmeter power on IP switch
    RTD_BOX          23     // Resistive Temperature Detector box
    GPS_CLOCK_PWR    24     // GPS clock
```

Dealing With RTD Failures

housctl specifically anticipated failures of one or more RTDs, and the power control commands allow for overriding automated power controls which are harmful due to invalid RTD data.

Operators can force power outlets on or off, indefinitely, even while normal housctl operations otherwise continue. This allows for such things as environmental control experiments, and overrides to work around hardware failures. The ‘powerstatus’ values range from 0 to 7:

0	off
1	on
2	should be off, but can't tell (hardware failure to respond)
3	should be on, but can't tell
4	operator forced off
5	operator forced on
6	operator forced off, but can't tell
7	operator forced on, but can't tell

As bits:

bit 0	off/on
bit 1	can't tell state (hardware failure)
bit 2	operator forced state in effect

CAMAC NAF Commands

These commands send arbitrary data to/from the CAMAC. The actual functions depend on the CAMAC hardware. You must consult those manuals for specifics.

naf *slot adrs func* [*data*]	send a 16-bit NAF command. *data* is optional, and only meaningful for “write” operations.
nafl *slot adrs func* [*data*]	send a 24-bit NAF command. *data* is optional, and only meaningful for “write” operations.

Detailed Functions of the States

IDLE Just background processing. Temperatures recorded ~1/minute, when dome_air is cold, maintains laser DI flow to avoid freezing. (Other states assume the laser is on, and so don’t maintain flow, as it conflicts with laser rack power.)

WARMUP Power on CAMAC crate, & laser rack. Increase temperature rate to every 10s, power on the motors, synchronize the diffuser phase, park the TR motor in a clear spot. STV to focus mode. After 90 min with no ICC command, automatically goes to COOLDOWN.

RUN Creates a data file named `yymmdd-hhmmss.run`, and writes to it a timestamp and a housctl build timestamp. Copies housctl.hed into the data file. Starts TR motor, tracks diffuser phase, turns off STV camera, choose and read polynomial file. Fire laser 'nruns' times, take fiducial and lunar data. When done, stops TR motor, parks it in a clear spot, turns on STV camera, closes data file, & goes to STANDBY state. Temperatures ~1 per 10 s. RUN uses the actual lunar range, which changes from run to run, to spin the TR motor at a speed that exactly interleaves the lunar return photons with the outgoing laser shots. It can be anywhere from ~19.8 to ~20.2 shots/sec.

Operators can change the length of a run, while running, by setting 'nruns' to a different value, e.g.

```
set nruns=5000
```

STANDBY STANDBY is similar to WARMUP, except it times out to COOLDOWN. Keeps equipment powered and "ready." Can transit to RUN. Continues temperatures ~1 per 10s. STV to focus mode. After 90 min with no ICC command, automatically goes to COOLDOWN. Usually, you will command a state change sooner than that. Keeps the diffuser phase in place.

COOLDOWN shut down active operation, power off CAMAC, APDs, laser, TR motor. APD fan, laser rack, and M75 (laser head cooling) stay on for 4 min. 30 min timeout back to IDLE state, data collection is same as WARMUP.

FIDLUN Turns off STV camera, move TR to dark patch. Runs FIDLUN test; when done, resets TR to clear, & returns to IDLE or STANDBY (whichever it came from). FIDLUN is the only other state (besides RUN) to fire the laser, and is used to for in-dome testing, mostly of receiving fiducials. The "lunar" gates are opened a fixed ~10 ms after the fiducial, and generally do nothing. FIDLUN uses the ACM internal timer to generate 20 shots/sec, regardless of the lunar range.

DARK performs a dark count calibration, and IDLEs. Moves the TR motor to block telescope light. Records the DRK record in the log file and data file, and sends it to ICC/TUI. Operators must make sure the APDs are really dark from other sources before running this test. When done, returns TR motor to clear. DARK is similar to STARE, with different data format.

FLAT performs a light count calibration, and IDLEs. Moves the TR motor to allow telescope light. Records the FLT record in the log file, and sends it to ICC/TUI. Operators must make sure the APDs are illuminated with a calibration reference before running this test. FLAT is identical to DARK, with light allowed in and a different data record format.

STARE similar to DARK, but leaves TR at clear space. Simply opens gates at a (default) rate of 1000 per second, keeps a count of detections, and reports them every (default) 500 gates.

LPOWER flips in the dichroic to direct the beam to the bolometer, starts the laser firing, and reports laser power twice per second, and the beam location as read on the bolometer. Continues until timeout or operator commands a state change. TEMPORARY for now: you must 'set topt=1' to make the laser actually fire. Otherwise, with no laser, the bolometer reads near zero. The laser power and position will come out as

```
cmdid i bolopower=2.29; bolox=1.12; boloy=0.54
```

where the x, y positions are in mm, and the number of decimal places is subject to change.

CALTDC makes 1000 start/stop pairs to the TDC at each of 5 different spacings: 20, 40, 60, 80, & 100 ns. Records all measurements in the *.cal file (and updates cal.last symbolic link). Takes about 6 seconds.

LASERCAL Similar to a combination of FIDLUN and CALTDC: makes ‘nruns’ laser shots, with start/stop pairs to the TDC for both fiducials and “lunars” at each of 5 different spacings: 20, 40, 60, 80, & 100 ns. Records all measurements in the *.las file (and updates las.last symbolic link). Tom says it requires a special cable be put in place on the system, but I don’t know why that’s needed.

FAKEDATA Generates fake data real time. Parameter ‘topt’ specifies how many Fid APD events per shot in its 10’s digit, and how man Lun APD events per shot, in its units digit. For example, ‘topt=12’ generates 1 Fid + 2 Lun APD events per shot. There is always a Fid FPD event per shot. Operators may change ‘topt’ on the fly.

Background processing All states perform background processing: record temperatures, control temperature of ILE and Utah box. Record GPS DAC ~1 per 10 s. Maintain laser head flow, as needed. Process ICC commands.

Known Housctl Problems

housctl crashes when laser log file too big

Problem: housctl crashes when powering up the laser if the laser log file (async.log) is > 2 GB.

Workaround: delete or rename the (async.log) file. housctl will start a new one. See “Files used by housctl” for exact filepath.

The failure of housctl on files > 2GB is likely due to the old C runtime library that we use. It can't handle big files. It is almost certainly fixed in the newer C libraries, but we can only switch to them if we upgrade our drivers and retest.

We can turn off the default logging now, since we aren't using it. We might see if there is a workaround that housctl can implement.

3 Advanced Housctl Information

This information is for advanced users of housctl, and/or the data files it produces.

Laser Blocking

Space Command can issue block times when we must not fire the laser into the sky. New format files give times to 1s precision; old-format gives to millisecond precision. Blockages have been as short as ~20 s, and as long as ~10 minutes. housctl reads a laser blocking file in the format given by Space Command, and allows laser firing only during release times. Since we're only given release during times we request, the laser is blocked most of the time we're not running. No file means no release. housctl logs all block and release events.

The only states affected are RUN, FIDLUN, and LASERCAL, since (as of 10/2007) they are the only states that fire the laser.

If the block time is < 2 min, housctl disables the laser, but continues in the same state. When the block is released, housctl resumes laser firing automatically. If the block time is >= 2 min, housctl moves to STANDBY, thus aborting the operation.

On startup, housctl read the file 'housctl.blk'. At any time, the command 'readblock' reads the current file of that name.

```
    On Thu, 16 Mar 2006, Eric L. Michelsen wrote:
    > What is the accuracy requirement for Space Command?

    From: Tom MurphySent: Thursday, March 16, 2006 10:42
    To: Eric L. Michelsen
    Cc: 'APOLLO Core'
    Subject: RE: [Apollo_core] houston clock
```

I don't rightly know. There are three relevant timescales here:

- the one ms resolution of their request to us
- the 50 ms resolution imposed by 20 Hz firing
- a relevant angular intrusion: if our avoidance has a half-angle of 0.5 degrees, and it typically takes 15 seconds for a satellite to cross the full degree, then it seems 0.5 sec accuracy makes the edge of our avoidance acceptably fuzzy.

But they've never hit us with a strict requirement. I would protest if they did (so many things are arbitrary here, like 0.5000 degree cone angle).

Auto-Detection of Block File Format

Housctl auto detects each line of the file separately, because this is simpler (it's stateless). If the line begins with a number between 2008 and 2100 (inclusive), it's a new-style permission line. If it begins with a number > 1000000, it's an old-style permission line.

Comments in Block File Format

housctl logs any line containing "`TARGET`" or "`AZ:`" or "`Report Date`" as a 'rem' in the log file.

All other lines are ignored.

New Format (Since ~2/2008)

Here is their sample new-format file. This one has only one target, but they also have a file with 2 targets.

```
                Classification: UNCLASSIFIED
```

```
UNITED STATES STATEGIC COMMAND LASER CLEARINGHOUSE (LCH) TIME WINDOWS REPORT

Date:    2008 Jan 11 19:19:26
From:    JFCC-SPACE/J95 (LCH)
To:      APACHE POINT
Subject: LCH Authorized Shoot (Open) Windows


1. The attached information contains the coordinated and approved
   spatial parameters

      (a) Authorized Shoot (Open) Windows

   During Authorized Shoot Windows, the laser owner-operator (O/O) is authorized
   to operate the approved system laser in accordance with the Source/Target
   geometry definitions contained in this report.

2. The laser O/O may perform Hybrid Predictive Avoidance (HPA) during Authorized
   Shoot Windows, if previously certified in writing by USSTRATCOM to do so.

3. Any deviation from this authorization must be immediately reported
   to the Laser Clearinghouse at: Commercial 805-605-6565,6578. DSN=275-(xxxx).

4. See below for comments specific to this mission.

5. If you have any questions, please don't hesitate to contact LCH at
   the above listed phone numbers.

JFCC-SPACE/J95 (LCH)
747 NEBRASKA AVE RM B209
VAFB, CA 93437



Mission ID:                       Apache Point_08011191926_P
Laser Owner/Operator:             APACHE POINT
Report Date/Time (GMT):           2008 Jan 11 19:19:26
Mission Name:                     Apache Point
Mission Start Date/Time (GMT):    2008 Jan 13 00:09:51
Mission Stop  Date/Time (GMT):    2008 Jan 13 01:48:00
Mission Duration   (HH:MM:SS):    01:38:08
Type of Windows in this report:   Authorized Shoot (Open) Windows
Comment:                          None
Number of Targets:                1

YYYY MMM dd (DDD) HHMM SS    YYYY MMM dd (DDD) HHMM SS      MM:SS
-------------------------    -------------------------    -------
2008 Jan 13 (013) 0009 51    2008 Jan 13 (013) 0148 00    0098:09

Percent = 100.00%

Source Geometry: (WGS-84)
---------------
Method: Fixed Point
Latitude:  32.7803 degrees N
Longitude: 105.8203 degrees W
Altitude:  2.788 km

Target Geometry: (WGS-84)
---------------
Method: Right Ascension And Declination
Catalog Date:    J2000
Right Ascension: 346.723 degrees
Declination:     -4.27 degrees
```

Here's the 2-target file:

```
           Classification: UNCLASSIFIED

UNITED STATES STATEGIC COMMAND LASER CLEARINGHOUSE (LCH) TIME WINDOWS REPORT

Date:    2008 Jan 11 19:22:00
From:    JFCC-SPACE/J95 (LCH)
To:      APACHE POINT
Subject: LCH Authorized Shoot (Open) Windows
```

```
1. The attached information contains the coordinated and approved
   spatial parameters

      (a) Authorized Shoot (Open) Windows

   During Authorized Shoot Windows, the laser owner-operator (O/O) is authorized
   to operate the approved system laser in accordance with the Source/Target
   geometry definitions contained in this report.

2. The laser O/O may perform Hybrid Predictive Avoidance (HPA) during Authorized
   Shoot Windows, if previously certified in writing by USSTRATCOM to do so.

3. Any deviation from this authorization must be immediately reported
   to the Laser Clearinghouse at: Commercial 805-605-6565,6578. DSN=275-(xxxx).

4. See below for comments specific to this mission.

5. If you have any questions, please don't hesitate to contact LCH at
   the above listed phone numbers.

JFCC-SPACE/J95 (LCH)
747 NEBRASKA AVE RM B209
VAFB, CA 93437



Mission ID:                       Apache Point_08011192200_P
Laser Owner/Operator:             APACHE POINT
Report Date/Time (GMT):           2008 Jan 11 19:22:00
Mission Name:                     Apache Point
Mission Start Date/Time (GMT):    2008 Jan 13 00:09:51
Mission Stop  Date/Time (GMT):    2008 Jan 13 01:48:00
Mission Duration   (HH:MM:SS):    01:38:08
Type of Windows in this report:   Authorized Shoot (Open) Windows
Comment:                          None
Number of Targets:                2

YYYY MMM dd (DDD) HHMM SS    YYYY MMM dd (DDD) HHMM SS      MM:SS
-------------------------    -------------------------    -------
2008 Jan 13 (013) 0009 51    2008 Jan 13 (013) 0101 19    0051:28
2008 Jan 13 (013) 0101 23    2008 Jan 13 (013) 0148 00    0046:37

Percent = 99.93%

Source Geometry: (WGS-84)
---------------
Method: Fixed Point
Latitude:  32.7803 degrees N
Longitude: 105.8203 degrees W
Altitude:  2.788 km

Target Geometry: (WGS-84)
---------------
Method: Fixed Azimuth/Elevation
Azimuth:   180.0 degrees
Elevation: 80.0 degrees

YYYY MMM dd (DDD) HHMM SS    YYYY MMM dd (DDD) HHMM SS      MM:SS
-------------------------    -------------------------    -------
2008 Jan 13 (013) 0009 51    2008 Jan 13 (013) 0023 10    0013:19
2008 Jan 13 (013) 0102 21    2008 Jan 13 (013) 0148 00    0045:39

Percent = 60.08%

Source Geometry: (WGS-84)
---------------
Method: Fixed Point
Latitude:  32.7803 degrees N
Longitude: 105.8203 degrees W
Altitude:  2.788 km

Target Geometry: (WGS-84)
---------------
```

```
Method: Fixed Azimuth/Elevation
Azimuth:   180.0 degrees
Elevation: 45.0 degrees
```

The formatting is flexible. Housctl verifies the following:

1. Any line containing "Type of Windows" also contains "open" after the colon.
2. Any line containing "Number of Targets" specifies 1.

For release/block, housctl considers only lines that start with numbers > 2000. The START and STOP times use day of year: `DDD,` and must be in parentheses. DDD=001 for Jan 1. housctl uses only the start and stop times. The first line following the open windows that does not start with a number > 2000 terminates the permissions. All other lines in the file are ignored.

The 'unblock' command largely eliminates the need for hand-made block files. However, a hand-made release/block file might be as simple as:

```
Report Date: Handmade file.  "Report Date" line required. Comment highly
recommended
2008 (013) 0009 51 (013) 0101 19
2008 (013) 0101 23 (013) 0148 00
```

Old Format (Prior to ~2/2008)

Here's a typical old-format block file from Space Command:

```
OAAUZYUW RUWRSPD5255 2921500-UUUU--RUWRCAW.
ZNR UUUUU
O 191500Z OCT 05
FM CMOC SPADOC4 CHEYENNE MOUNTAIN AFS CO//SDD// TO RUWRCAW/1SPCS SCC
CHEYENNE MOUNTAIN AFS CO BT
UNCLAS                  FOUO
SUBJECT:  SPADOC NOTIFICATION  (U)
(U) REAL

1.  (U) MESSAGE TYPE:  SPADOC DE SITE CLEARINGHOUSE REPORT 2.  (U)
PREPARATION DATE TIME:  191458ZOCT05
3.  (U) VALID FOR DE EMITTER SITE/VESSEL/MISSILE/SAT # APACH
4.  (U) DE TARGET TYPE:
          FIXED BORESIGHT:
            AZ:  180.00 DEG  EL:  45.00 DEG
           PERIOD OF INTEREST:  05293040500.000Z - 05293123300.000Z 5.  (U)
SAFE IRRADIATION TIMES FOR EMITTER:
           START TIME  DURATION TIME    STOP TIME      WAIT TIME
        293040500.000Z 030051.804 293070551.804Z 000153.253
        293070745.056Z 025035.089 293095820.145Z 000014.375
        293095834.520Z 005057.132 293104931.652Z 000013.602
        293104945.253Z 003743.236 293112728.490Z 000017.975
        293112746.465Z 010513.535 293123300.000Z
6.  (U) REMARKS:
BT
#5255
```

The formatting is flexible.

The START and STOP times are in day of year: `DDDHHMMSS.SSS`, and must contain a decimal point followed by at least 1 digit. DDD=001 for Jan 1. The Z means Zulu, or UTC, and is optional. The duration and wait times are in HHMMSS.SSS. housctl uses only the start and stop times (it requires numbers in the file for duration, but ignores them). All other lines in the file are ignored.

The 'unblock' command largely eliminates the need for hand-made block files. However, a hand-made release/block file might be as simple as:

```
TARGET: Handmade test file.  A comment is optional, but highly recommended
293040500.0 0 293070551.0
293070745.0 0 293095820.0
```

Files Used By housctl

Goals: We want our file/directory structure to achieve the following:

1. Complete record of both production and experimental activity
2. Allow for software development (with its bugs), while protecting real data
3. Accessible enough with standard Unix commands for humans to find what they need

log files go into a unified 'log' directory. (The development directory [src/, bin/, etc.] is completely separate, and described in the development chapter.)

Housctl uses UTC for filenames with embedded dates/times. It's not yet clear...

- When the daily/ files are copied to the date-specific directory. How do we keep old poly/block/eopc files from being replicated each day to new directories?
- We need to insure that the simple globbing formats eliminate the need for links
- We don't know if NFS supports UNC (Universal Naming Convention filenames, so we avoid them for now). This is a procedural issue, and does not affect the final file structure.

Relative paths below are relative to the working directory when housctl was run. Our directory structure looks like this:

```
/home/apollo
    housctl              the current version of the executable
    housctl.rc           the startup script run when housctl starts
    housctl.cum          the permanently "remembered" values, across startups
    housctl.hed          the file prefixed to every data file, usually comments
    housctl.rtdcal       the current temperature calibration file

/home/apollo/data        this is the default setting of variable 'fileprefix1'
    log.last             symbolic link to the current log file
    run.last, etc.       symbolic links to the most recent run, fid, drk, etc. files

    daily/
        PolysReflrSecn       written by operator for today's polynomials
        eopc(??)             eopc file that created the Polys* files
        housctl.blk          written by operator for today's laser blocking
    2006/
        log/                         common directory for all log files
        yymmdd/                      one directory for each date
                yymmdd.log           symbolic link to currently active (running) log
                (all the files for this date:
                *.fits, *.str, *.drk, *.flt, *.cal, *.fid, *.run)
                mmddyy.cum   (not yet implemented) copy of housctl.cum at
                             end of this date

        yymmdd/              one directory for each date

    2007/  ... as above
```

The directory names are implemented partly by housctl, and partly by the directory prefixes we give housctl. housctl puts a symbolic link to the log file (with same name as log file) in the data directory at data directory creation time, and at cooldown. This may (rarely) make 2 log symlinks, which is what we want.

Housctl does not yet implement data file redundancy: There is only 'fileprefix1', and no 'fileprefix2' as of 4/2009. For redundancy, there would be copy of this tree on `houston`, and another copy on `cocoa`. To achieve the above directory structure, we mount "cocoa:/home" on `houston` as "cocoa", then set the housctl parameters in housctl.rc as follows:

```
fileprefix1="/home/apollo/data"
fileprefix2="/cocoa/apollo/data"
```

and run housctl with a working directory of "/home/apollo/data". Note that if either 'fileprefix1' or 'fileprefix2' is a relative path, it is relative to the working directory when housctl was run. This is useful for testing.

Below, files that housctl writes that start with (prefix) are those that housctl writes 2 copies of, one with each of 'fileprefix1' and 'fileprefix2'. If either of these 2 files cannot be opened, housctl continues operation with the one that can be opened.

`/home/apollo/housctl.cum`
housctl reads and updates this file for cumulative parameter storage, such as laser shots, and the rxxpcum (etc.) totals. This filename can be overridden by command line or housctl.rc (for special testing), but beware that moving the Rx mirror with a different *.cum file screws up the mirror position.

`housctl.rc` housctl reads this file at startup as a list of ICC commands. This can override built-in defaults. In particular, this specifies file path prefixes. housctl reads this file from the current working directory.

`housctl.rtdcal` RTD calibration and name file. housctl reads this file from the current working directory at startup, and on an 'readrtd' command.

(fileprefix1)`/daily/PolysRefl`*`r`*`sec`*`n`* These files contain the prediction polynomials. Housctl searches these files on the 'refl' command, and on each 'run' command. *r* is the reflector number (defined elsewhere), and *n* is the section number, where each section spans a different time during the night.

(fileprefix1)`/daily/housctl.blk` housctl reads this file during operation to automatically block the laser as needed.

(prefix)/*yyyy*`/log/`*yymmdd*`.log` The log files: every day at UTC 20:00 (the **roll time**), or when housctl starts, it opens a log file, named with the current 24-hour interval at the time of opening. If the file exists, housctl appends to it, so no data is lost. Each log file therefore contains 24-hours of logs.

(prefix)/*yyyy*/*yymmdd*/*yymmdd-HHMMss*`.run`	The run data files.
(prefix)/*yyyy*/*yymmdd*/*yymmdd-HHMMss*`.str`	The stare data files.
(prefix)/*yyyy*/*yymmdd*/*yymmdd-HHMMss*`.drk`	The dark data files.
(prefix)/*yyyy*/*yymmdd*/*yymmdd-HHMMss*`.flt`	The flat data files.
(prefix)/*yyyy*/*yymmdd*/*yymmdd-HHMMss*`.fid`	The fidlun data files.

Whenever the laser is powered on, and laser logging is enabled, housctl appends the log to the file:

async.log in the current directory of housctl when it was run (usually /home/apollo/).

housctl.hed The housctl.hed file in use as of 4/2009 consists of a single line:

```
rem TDC RTDs reordered to go from bottom to top.
```

Previous Files Used By housctl

Previously: Housctl uses PST for filenames with embedded dates/times. All files were in the working directory when housctl was run.

/home/apollo/housctl.cum
housctl reads and updates this file for cumulative parameter storage, such as laser shots, and the rxxpcum (etc.) totals.

housctl.rc	housctl reads this file at startup as a list of ICC commands. This can override built-in defaults.
PolysRefl*r*sec*n*	These files contain the prediction polynomials. Housctl searches these files on the 'refl' command, and on each 'run' command. *r* is the reflector number (defined elsewhere), and *n* is the section number, where each section spans a different time during the night.
housctl.blk	housctl reads this file during operation to automatically block the laser as needed.
yymmdd.log	Previously: The log files: every day at local noon, or when housctl starts, it opens a log file, named with the current date at the time of opening. If the file exists, housctl appends to it, so no data is lost. The next noon with a different date, housctl rolls to a new log file. This means a log file could contain as much as 36 hours of data, if you start housctl at 0:00 (midnight), and it rolls at noon the *following* day. Note: If you are doing testing in the morning, and you start housctl, the log file gets the current date, not yesterday's date. I found this to be more natural than 24-hour windows that were offset from the current date.

Using housctl As A Human ICC

I frequently telnet into housctl port 5320 and act as a human ICC (**bold blue** denotes human commands to housctl, `normal` is housctl response):

```
telnet houston 5320
rem  Accepted ICC conn, fd = 10, from 127.0.0.7:38624
```

You can now enter commands exactly as from the TUI "apollo houston" command, and see directly the responses destined for ICC. All ICC commands are lower case. 'help' gives a summary of settable parameters and ICC commands. Remember, this is intended to be a machine-machine interface, so is not the most user friendly. Eventually, common commands will be implemented in TUI in a friendlier way.

Warning At a machine-machine interface, you must be careful to command ONLY VALID operation sequences. For example, from IDLE, entering 'set state=3' (RUN) may produce unexpected results.

Here are annotated examples of selected functions. The annotations provide a summary of the state processing, but full details of each state are given later.

Script Files

Currently, the only housctl script file is housctl.rc. It is simply a list of ICC commands that housctl executes on startup. Lines that start with "#" are script comments, and completely ignored (not written to the log file). Lines that start with "rem" are ICC comments, and are written to the log file just like all other ICC commands. A typical housctl.rc file is this one from 4/3/2009:

```
# This is a startup .rc file for Apollo data files.
# housctl executes it as ICC commands.
set runfidgw=7
set t_utah_push=-.25
set t_ile_low=20
set t_ile_hi=25
set flow_interval=30000
set nruns=5000
set nstares=5
set dskew=-41
# dphase_target is now a 'cum' parameter (non-volatile)
```

housctl Startup Sequence

In more detail, when housctl starts, it performs the following sequence:

- Processes any `debug=`, `doptions=`, or `fileprefix1=` command line arguments. It is helpful to process these before opening the log file.
- opens the log file (based on current date)
- executes housctl.rc (if it exists) as a sequence of ICC commands. Like all ICC commands, they are recorded to the log file. Typically, you don't need any housctl.rc file, however, you can do things such as override defaults, etc. with it.
- processes remaining command line arguments
- Listens on TCP port 5320 for ICC connections
- Enters IDLE state

This order allows most command line arguments to override everything (compiled-in defaults, and housctl.rc settings). However, it means the command line sets take place *after* housctl.rc is executed.

Using Time Offset Hunting

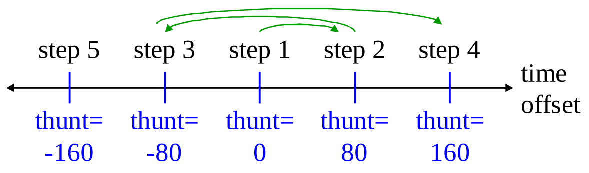
If there is a large uncertainty in the RTT of the target, you may not know where (in time) to look for the return. housctl has a feature to automatically vary the time offset, to make it faster and more reliable to search through time offsets. The time-hunt feature steps through 5 (default) different time-offsets, running for 'nruns' shots *at each time offset*! Therefore, the total shots is 5 × nruns.

The "time-hunt" feature is enabled by using the 'huntrun' command instead of 'run':

```
huntrun offset(ns) [nsteps] [huntdelta (ns)]
```

'offset' is required. The default nsteps is 5, and default huntdelta is 80 ns.

'thunt' is a readable parameter that tells (at any instant) the current time-search offset. housctl sends 'thunt' to the TUI status-line each time 'thunt' is set. The time-offset starts at 'offset', and alternates from side to side in a widening search. E.g.:



To stop a run, you must use 'standby'. Lowering 'nruns' will change the number of shots per time-step, but does not terminate the run.

'thunt' and 'offset' add to 'predskew', so each has the same sign and direction effect as 'predskew'. Therefore, you can experiment with 'huntrun' without "screwing up" predskew. 'offset' skews the hunt to one side; it does not widen the window. For example, if you have a good predskew at A15, and you want to look for Lunakhod 1, you can poke around for a while, and your old predskew is still intact.

Obsolete Time Offset Hunting

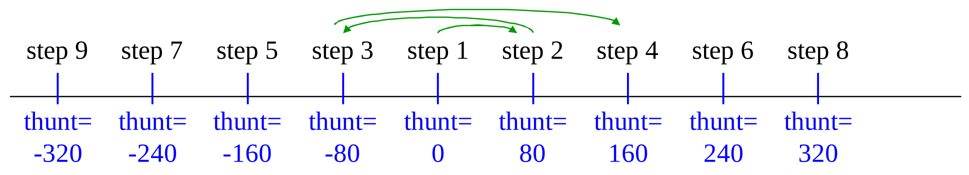
Historical info for older housctl, which we will eventually remove from this document: before we had the 'huntrun' command:

The time-hunt feature steps through 9 different time-offsets, running for 'nruns' shots *at each time offset*! Therefore, the total shots is 9 × nruns.

The "time-hunt" feature is enabled by setting the 'huntdelta' parameter to the time-search offset in ns (a positive number):

```
set huntdelta=80
```

Now, every RUN will invoke the time-search feature. ‘thunt’ is a readable parameter that tells (at any instant) the current time-search offset. The time-offset starts at 0, and alternates from side to side in a widening search. E.g.:



To stop a run, you must use ‘standby’. Lowering ‘nruns’ will change the number of shots per time-step, but does not terminate the run.

‘thunt’ and ‘huntstart’ (see below) add to ‘predskew’, so each has the same sign and direction effect as ‘predskew’. Another parameter, the ‘huntstart’ parameter defines at what time offset to start (in ns). It simply adds to ‘predskew’, so you can experiment without “screwing up” predskew. ‘huntstart’ skews the hunt to one side; it does not widen the window. For example, if you have a good predskew at A15, and you want to look for Lunakhod 1, you can mess with ‘huntstart’, which adds to ‘predskew’, poke around for a while, then set ‘hunstart=0’ and your old predskew is still intact.

‘huntstart’ is still in effect, even when ‘huntdelta’ is zero.

Turn off the time-search feature by setting ‘huntdelta’ and ‘huntstart’ to zero:

```
    set huntdelta=0 huntstart=0
```

4 Operations Support Software

Lunar Prediction

```
From: Tom Murphy
Sent: Monday, June 09, 2008 11:29
To: APOLLO Core
Subject: [Apollo_core] New Predictions
```

The prediction software has a fresh face on it, and should be more robust in the future. The new software is installed and working properly on Houston. I did comparisons between the new predictions and archived predictions from four different nights over the last few months—all checked out perfectly, so I am comfortable with the update.

The prediction code is (at long last) part of the APOLLO SVN repository, providing a central, single origin for updates (revertible).

Specific improvements are:

- no longer sensitive to different "setting" times of the various reflectors, so runs **should** always work for a given start time, provided there are enough data points for the fit. No more tweaking by a minute or two to get all reflectors on the same "page"
- result is that polynomials may span slightly different time ranges for the various reflectors: no longer required to be in lock-step
- some code clean-up, including some changes in naming conventions (raw output times now called pred* instead of fit*)
- more portable, in that path variables are now part of a pred_files.h header file rather than hard-coded in C source
- and, of course, this is all on SVN now

I archived the old prediction code onto cocoa, along with the historical poly files and auxiliary files that have been produced for all our nights of ranging.

Using the Prediction Software

```
From: Tom Murphy
Sent: Tuesday, January 02, 2007 15:25
To: Russet McMillan
Cc: James Battat; Eric Michelsen; C.D. Hoyle
Subject: new poly procedure
```

As of now, there is a new procedure for the polynomial generation: the mkpoly routine does it all, replacing the get_eopc (autoget), eop_now.py, and mkpoly sequence. Now, just:

`mkpoly MM DD YYYY HH MM SS` (or just **`mkpoly`** with no arguments)

The eopc04.YY file is retrieved (YY based on YYYY entry or "now" if no args). The eopc04.YY is propagated to the mkpoly start time (whether based on cmd-line argument or "now"), becoming eopc04.YY_mod (no longer overwrites original file).

The `eop_now.py` command is run with (new) command line arguments specifying the "propagate-to" time: again, the mkpoly start time.

Then mkpoly does all its usual business, creating the polynomial segments.

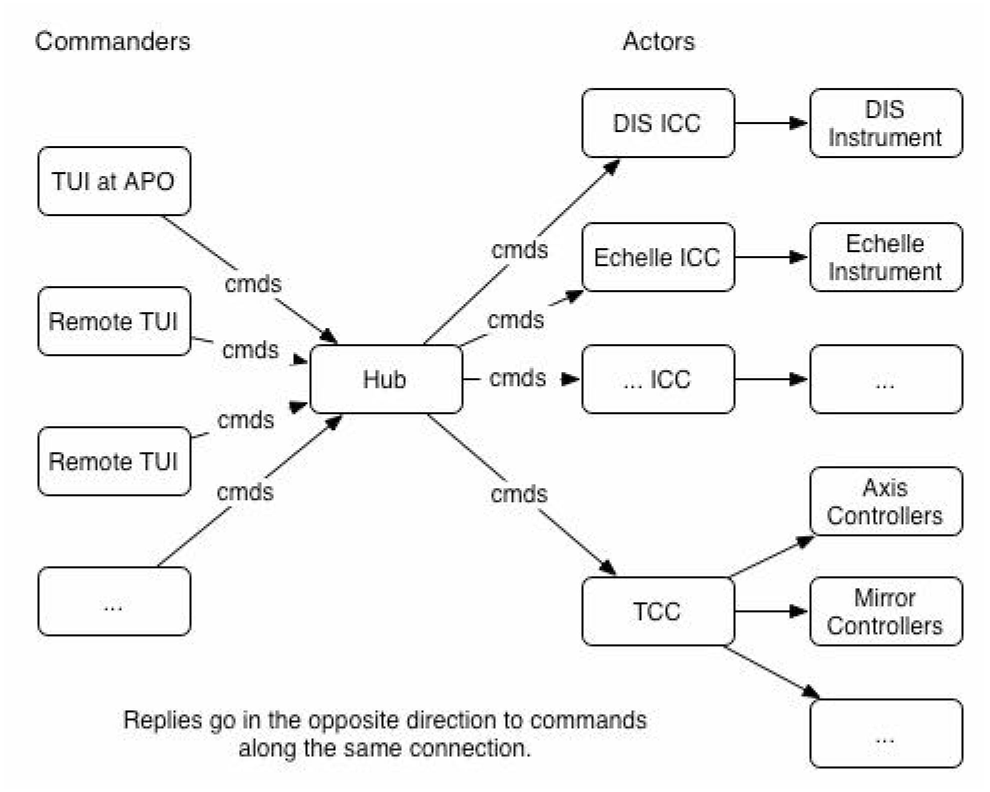
I have a slight preference for using command line arguments explicitly, because the extrapolation of earth orientation parameters will be the most up-to-date possible by doing so, though a few hours should make very little difference.

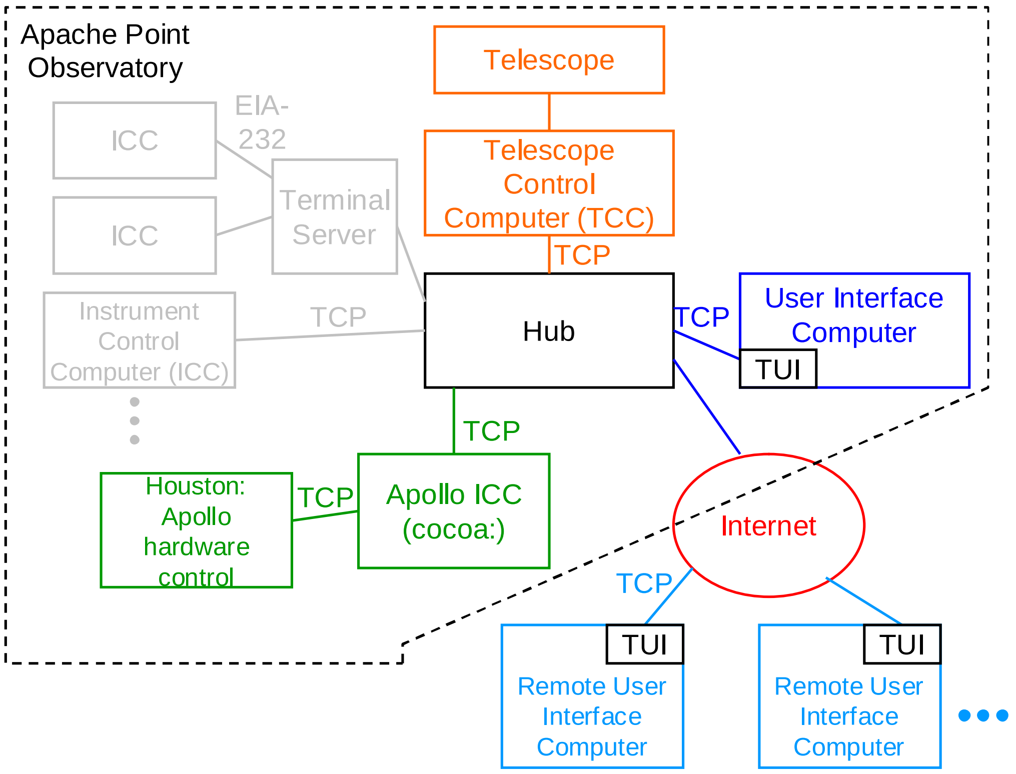
In any case, the three-step dance is now one-step. More improvements to come (EOP's extrapolated for each time step). Let me know if you encounter difficulties.

5 Houston/ICC/Hub/TUI Interfaces

APO TUI/Hub/ICC/Control Architecture

From http://rowen.astro.washington.edu/ICCManual/:




Apache Point Observatory
Telescope
Telescope Control Computer (TCC)
TCP
ICC
ICC
EIA-232
Terminal Server
Instrument Control Computer (ICC)
TCP
Hub
TCP
User Interface Computer
TUI
TCP
Apollo ICC (cocoa:)
Houston: Apollo hardware control
TCP
Internet
TCP
TUI
Remote User Interface Computer
TUI
Remote User Interface Computer

TUI is a program that runs on a user's computer. The Hub is a computer at APO running Hub software. TCC is the Telescope Control Computer, which physically controls the telescope. It is essentially an ICC for the telescope itself.

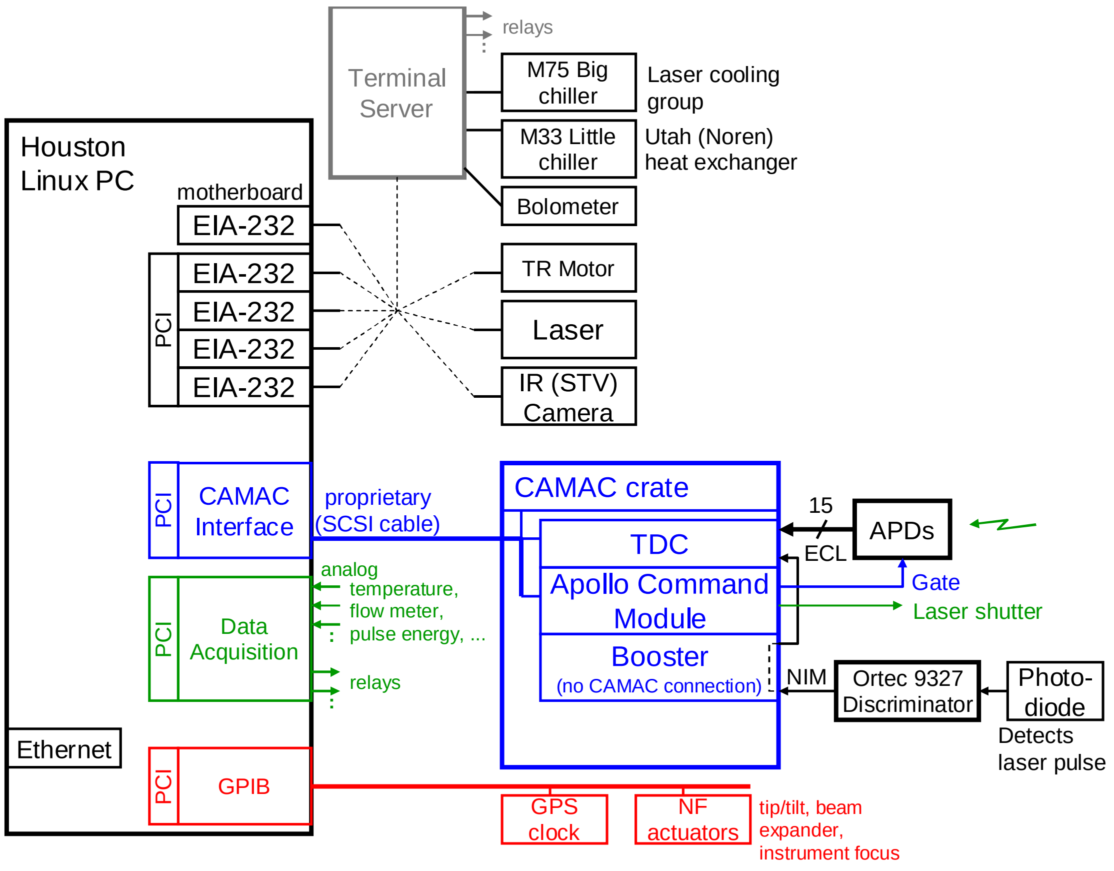
Each experiment has its own customized TUI for its own needs. The APOLLO TUI comprises an APOLLO-specific layer called **APOLLO-TUI** or **ATUI**. ATUI is built on an APO-provided TUI base. In this document, almost all references to TUI actually refer to ATUI, i.e. to the TUI seen by APOLLO operators.

Houston/ICC Overview

Houston is the direct hardware control. ICC (Instrument Control Computer) is a function that interfaces the telescope Hub to Houston. Multiple TUIs can connect through the hub to ICC.

Despite its name, ICC is a function, not necessarily a computer. ICC runs on cocoa:, a separate computer from Houston. For testing, ICC ran on Houston. Later, we needed to raise the priority of housctl to meet its real-time requirements.

Houston Functions



Houston runs one control program continuously.

Overview of operating states: IDLE, WARMUP, RUN, STANDBY, COOLDOWN. All states perform background functions of temperature control, monitoring, etc. Detailed descriptions of states later (Detailed Functions of the States p23).

Houston records to disk two types of files: log files, and data files. Houston writes two copies of *each* type. We will configure housctl to put the two copies on different physical disks, and probably different computers, via NFS.

Houston sends a verbatim copy of all log and data records to ICC. ICC can ignore much of this.

ICC Functions

Read Houston data records, reformat, send to TUI

Read Houston log records, discard some, reformat others, send to TUI

Feed Hub/TUI

Read/process TUI commands, forward to Houston

Read/process Houston's command responses and other ICC records, forward to TUI.

Houston/ICC Logical Interfaces

There is one TCP connection between Houston and ICC:

Port 5320 Command/response (bidirectional)

They are all ASCII, line oriented. Houston requires \n termination, and ignores any \r in the stream (the \r allows a human to play ICC with Telnet). Houston terminates its lines with \n. The maximum line length that Houston will tolerate and generate is 511 bytes, including \n (there is no \0 over ASCII interfaces). Houston tries (unsuccessfully) to keep replies within 80 chars, for easier human inspection.

Houston/ICC Data & Log Files

Data and log files have same high-level format. As currently defined, for any time period that spans both files, the log file is the same as the data file minus fid and lun records. However, we expect the log files to span longer periods, probably 24 hours per file. The data files will be much shorter, typically 5 minutes for a single run. We expect to shoot all 4 retroreflectors within a 1/2 hour window, at 5 minutes per reflector. We'd like to have those in 4 separate data files. The log info is pretty sparse, but some of it is relevant to the photon data (fid and lun), so we have all the log info duplicated in-line right in the photon files, to make connections between log info and photon data easier to analyze.

We also expect Houston (the hardware control computer) to write two copies of all log and data files. Probably one file will be a Houston hard disk; the other file will be an NFS share on another computer (the Apollo ICC?).

The data file is open during WARMUP, RUN, STANDBY, and COOLDOWN states. All log file records are duplicated in any open data file. Photon timing records (fid and lun types) are specific to the data file.

Files are ASCII for 3 reasons, in order of importance: (1) Machine independence of numeric formats (big/little endian, floating point), (2) ability to sort/filter with standard unix programs (grep, sort, etc.), and (3) human inspection for sanity.

File format: Each record is an ASCII line terminated with \n (linefeed). The first 3 characters define a record type, the 4th is a version for that record type. Each type/version pair has its own format beyond that, though most will be simply “keyword=value” sets.

Data File Record Types

A data file starts with date/time information, followed by a snapshot of all the housctl variables. Here’s an excerpt from a real data file:

```
rem
rem Apollo run Sat May 13 07:41:15 2006
rem Houston Control v. May 12 2006 16:37:37
rem TDC RTDs reordered to go from bottom to top.

par0 logfile="060512.log"
par0 datafile="060513-074115.run"
par0 state=3
par0 icc_session=0
par0 debug=0
par0 houstbase=1147478929416
par0 houstime=27146411
par0 logfile_due=71470584
par0 motrps=20.0975119656049
...
par0 pr_errorstr="pr_read.86: status 0"
rem fid0 shot# accsec tws*2  frcount gw bright phase fpd nhit
rem lun0 shot# accsec tws*2   frcount gw ppred tpred nhit
par0 polyname="daily/PolysRefl3sec1"
par0 reflector=3; t0=133.231296296; tf=133.384074074; polyspan=220.000000; ncoef
f=9
par0 a0=2.565978565796585
par0 a1=-0.091589809773129263
par0 a2=0.58279683568352503
par0 a3=0.42266978591166771
par0 a4=-1.8552913261166151
par0 a5=-0.86810426921978268
par0 a6=2.5946125203131092
par0 a7=1.163791703090531
```

```
par0 a8=-2.8191440084312989
...
rem 2006-05-13T07:41:21 Laser enabled
lun0     -10     6 28589054 000000010   8 -1073743368 -1073743368  0
lun0     -10    11 65459966 000000010   8 -1073743368 -1073743368  1  6:0180
gps0 gpstrig="133:07:41:15.918006"; cpserr=0
lun0     -10    17  2330878 000000010   8 -1073743368 -1073743368  0
tmp0 ILE_air=29.43; UTAH=23.19; cabinet_air=16.39; Laser_air=25.48; Laser_Bench=
24.39; AOML=24.44; TDC1=24.71; TDC2=27.42; TDC3=32.71; TDC4=37.54; TDC5=37.00; P
G_in=14.89; Noren_out=19.25; PG_out=16.11; DI_out=28.18; DI_in=28.55; DI_plenum=
16.71; CAB_plenum=15.32; dome_air=15.00; Laser_rack=25.02; ILE_wall=33.46; ILE_i
ntake=17.69; GPS_clock=32.21; PG_shuttle=15.13
pow0 2006-05-13T07:41:32 power11=0; name="Passive_cool"
par0 powerstate=1,0,1,1, 1,1,0,0, 2, 0,1,0,0, 1,0,0,0, 1,1,2,2, 1,1,1,1
pow0 2006-05-13T07:41:32 power17=1; name="M33"
par0 powerstate=1,0,1,1, 1,1,0,0, 2, 0,1,0,0, 1,0,0,0, 1,1,2,2, 1,1,1,1
pow0 2006-05-13T07:41:32 power3=1; name="Noren_fans"
par0 powerstate=1,0,1,1, 1,1,0,0, 2, 0,1,0,0, 1,0,0,0, 1,1,2,2, 1,1,1,1
pow0 2006-05-13T07:41:32 power1=0; name="UTAH_heat"
par0 powerstate=1,0,1,1, 1,1,0,0, 2, 0,1,0,0, 1,0,0,0, 1,1,2,2, 1,1,1,1
fid0       1    18 19374890 058522016   6 -2147483648 1100 2307  4  9:2235 13:3082
14:1363 15:2307
fid0       2    18 24341420 061005281   6 3076 1100 1458  7  2:2956  5:4066  8:1638
10:3976 11:1720 13:1693 15:1458
fid0       3    18 29333490 063501316   6 3060 1100 1389  7  1:1623  4:1641  5:3992
7:0452  9:1376 14:1631 15:1389
fid0       4    18 34293444 065981293   6 3222 1100  927  6  2:1187  5:3535  9:3232
14:1184 15:0927 16:0121
...
fid0      51    20 68174734 182921938   6 3261 1099 1137  7  5:3738  6:1406 10:3651
11:1399 12:3146 13:1389 15:1137
fid0      52    20 73140868 185405005   6 3196 1099 1294  3  5:3903 10:3798 15:1294
lun0       1    20 75637062 186653102   8 2250 2250  0
fid0      53    20 78133100 187901121   6 3034 1100 1148  4  5:3755 10:3671 14:0828
15:1148
lun0       2    20 80603594 189136368   8 1814 1814  3  1:1332  9:1188 11:2898
fid0      54    20 83092972 190381057   6 3209 1100 1328  4  1:1571  5:3944 10:3737
15:1328
lun0       3    20 85595666 191632404   8 2157 2157  0
fid0      55    20 88078888 192874015   6 3136 1100 1604  4  7:1012 11:1856 15:1604
16:1872
lun0       4    20 90555622 194112382   8 2109 2109  0
fid0      56    20 93054612 195361877   6 3233 1100 1135  3  5:3738 12:1422 15:1135
lun0       5    20 95541580 196605361   8 1940 1940  1  9:2217
fid0      57    20 98020832 197844987   6 3210 1099 1137  3  5:3744  8:0798 15:1137
gps0 gpstrig="133:07:41:36.005172"; cpserr=0
lun0       6    21   517216 199093179   8 1878 1878  0
fid0      58    21  3013074 200341108   6 2956 1099  986  5  4:1248  5:3590 10:3478
12:1203 15:0986
lun0       7    21  5483996 201576569   8 2373 2373  0
fid0      59    21  7973056 202821099   6 3224 1100 2326  2 14:3859 15:2326
lun0       8    21 10475358 204072250   8 2083 2083  2 11:2334 12:3611
```

fid0

The `fid` and `lun` records are described in 'rem' in the data file, so we can track changes more easily. `fid` are fiducial records. There are some fixed columns, followed by variable number of TDC values. E.g.,

```
rem fid0 shot# accsec tws*2  frcount gw bright phase fpd nhit
fid0       2    18 24341420 061005281   6 3076 1100 1458  7  2:0956  5:4066  8:1638
10:3976 11:1720 13:1693 15:1458
```

Early fiducials, before a valid GPS time stamp for example, are flagged with shot # -1, to indicate they are not used for anything. Housctl cannot know when a fiducial shot is missed by the FPD (fast photodiode), so fiducials always increment by 1, even when some shots were lost by the hardware.

The FPD brightness is measured by a circuit that takes time to settle; housctl repeatedly reads the ADC waiting for convergence. If the reading does not converge in a reasonable time, housctl reports the last value, but negated, to flag that it is unreliable.

The FPD entry is just a copy of the channel 15 TDC measurement (which is also retained in the hit list). If a fiducial record has no Fast PhotoDiode (FPD) measurement in the TDC, housctl uses a dummy value of 4999 TDC counts, which is outside the valid range of 0 - 4095. This causes the lunar gate to open at a meaningless time, but the record is still taken.

The hitlist uses 4-digit, leading zero TDC values, so that each TDC channel:value pair has no spaces, for easier parsing. Channels are numbered 1-16.

Note that housctl cannot know which fiducials are odd or even, so there is no indicator for this in the data.

lun0

The fid and lun records are described in 'rem' in the data file, so we can track changes more easily. lun are lunar return records. There are some fixed columns, followed by variable number of TDC values . E.g.,

```
    rem lun0 shot# accsec tws*2   frcount gw ppred tpred nhit
    lun0       2    20 80603594 189136368   8 1814 1814  3  1:0932  9:1188 11:2898
```

If the ACM gets a "lunar" gate with no corresponding fiducial gate in the queue, the ACM can't know any shot number, so fills in shot # = –10. This usually means the FRC hit the target leftover in hardware before a valid target got written.

The hitlist uses 4-digit, leading zero TDC values, so that each TDC channel:value pair has no spaces, for easier parsing. Channels are numbered 1-16.

str0

Stare data files (*.str) have stare records which look like ?? They are normalized how??

Log File Record Types

Each record begins in column 1 with a 3 char record type. All types except "rem" have a 1 char version code suffix attached to the record type. The following record types are defined:

rem	A comment of arbitrary text
`tmp0` *sensor=temp; ...*	Uncalibrated temperatures. Sensors are names, temps are in deg C.
`tmp1` *sensor=temp; ...*	Calibrated temperatures. Sensors are names, temps are in deg C.
`icc0` *ansitime command*	verbatim copy of received ICC command
`drk0` *chan1 chan2 ... chan16*	see below
`flt0` *chan1 chan2 ... chan16*	see below

All keywords have the same meaning across all record types, by design, and you can count on it always being so.

Each record type is further described below.

par0

a list of arbitrary parameters, such as time-to-moon estimation polynomial. E.g.

```
    par0 reflector=3; t0=263.166667; tf=263.319444; polyspan=220.000000;
    ncoeff=10
    par0 polyname="housctl.poly"
    par0 a0=2.4465802250443649
    par0 a1=-0.16897226866481169
    par0 motrps=19.938; t=263.273675; rtt=2.432493
```

Keywords: can be any of the parameters, including these:

day	float	current time in days (including fraction), where 1/1 = day 1
polyname	string	name of polynomial file
reflector	int	0=Apollo 11, 1=Lunakod 1 (never seen), 2=14, 3=15, 4=Lunakod 2
t0	float	start time of polynomial, as defined elsewhere in this document
tf	float	end time of polynomial
polyspan	int	span of polynomial, in minutes
ncoeff	int	# coefficients = order + 1
a1..a50	float	polynomial coefficients
t	float	time of polynomial evaluation
rtt	float	round trip time from polynomial at time t, in sec
motrps	float	motor speed, in rps
thunt	float	time hunt offset, in ns, for automated time searching
atime	string	generic ANSI time stamp
gatewidth	int	current gatewidth parameter for DARK, STARE, etc.
apdtoffs	int list of 16	APD time offsets (from fast photodiode to fiducials, in TDC units) as

```
   apdtoffs=636.207,688.602,0,654.450,0,671.388,675.043,0,
      687.673,673.837,662.598,614.974,648.818,629.876,0,668.248
```

exc0

`exc0` denotes an exception (unexpected event) with related parameters, e.g.

```
   exc0 2007-01-04T17:36:54 severity=1; bits=0x20000;
                                    text="Unknown 'set' parameters: refl=3"
```

See “Events and Alarms” for description.

gps0

GPS time, DAC and other status. E.g.,

```
   gps0 gpstrig="263:06:34:14.222062"; cpserr=0
   gps0 gpstime="F03  UTC  09/20/05 06:44:58"; gpsdac="F71 phase= 1.049E-07 s
   offset= 2.487E-11  drift= 3.649E-09/DAY  DAC=  -187"
```

Keywords:

`gpstrig`	string	trigger time of ACM’s 1 pps, as "263:06:34:14.222062"
`cpserr`	int	counts per second discrepancy from 1 second.
`gpstime`	string	asynchronous gps time, as "F03 UTC 09/20/05 06:44:58"
`gpsdac`	string	DAC statistics, directly from clock, as "F71 phase= 1.049E-07 s offset= 2.487E-11 drift= 3.649E-09/DAY DAC= -187"

tmp1

Sensors are names, temperatures are type float, in deg C. Sensor names are subject to change. E.g.,

```
   tmp0  ILE=29.09; UTAH=31.21; cabinet_mid=16.55; TDC1=34.62; TDC2=45.21;
   TDC3=47.06; TDC4=43.09; TDC5=29.34; Noren_in=23.54; Noren_out=20.18;
   dome_air=15.90;
```

Sensor names above are as of 9/21/2005.

icc0

verbatim copy of received ICC command, with timestamp, e.g.
```
    icc0 2005-09-19T23:34:13 775 get houstime
```

drk0/flt0

APD dark or flat count calibration, e.g.
```
    drk0 2005-09-19T21:47:14 195 10000   0 318 10000 142 258 302 253 152 405
    319 368 448   0 322
    flt0 2005-09-19T21:47:14 923 10000   0 899 10000 952 688 823 953 952 805
    919 768 948   0 822
```

The 16 columns are the dark counts, normalized to 10,000 gates, for channels 1 through 16.

flw0

Flow rates of the laser head loop, and the Noren propylene-glycol loop:
```
    flw0 laser_flow=0.350  noren_flow=2.123
```

The old flow rates in early code had the propylene-glycol loop and the dionized water loop (laser head) reversed. Thus “pg_flow” was really laser head, and “di_flow” was really Noren.
```
    flw0 pg_flow=0.350  di_flow=2.123
```

pow0

Power control records. For example,
```
    pow0 2005-11-26T10:38:39 power13=1; name="Laser"
    pow0 2005-11-26T10:38:42 power10=0; name="ILE_exhaust"
```

Every time a log file is opened, housctl queries the hardware for the full current power state, and records it in the log file.

chl0

Chiller status:
```
    chl0 m33_inttemp=17.0; m75_inttemp=20.0
```

Russell’s Comments On Data Formats

- Consider always putting the time stamp in the same column (1 or 2) for easier parsing. Right now it will be in column 2 or 3, depending on what is in the first column. Time in column 1 means your data will sort naturally, for what that is worth.

- Consider adding the date to the time string to eliminate ambiguity, e.g.
```
       2004-12-10T12:17:05.243000
```

This is ANSI standard format, but many folks use a space instead of a "T". It uses a few more characters than just time, but is standard, clear and sorts easily.

- Sparsely encoded data would be safer to parse with a separator, since there is no possibility of mixing channel #s and values, e.g.:

  chan#=value chan#=value

rather than

  chan# value chan# value...

- Consider keyword=value format. It's widely used at APO (ICCs must use it for replies) so parsers are readily available. To avoid a time keyword, you'd have to put the time first and parse it separately. Thus:

  TIME cmdx="command string"

TIME lunx=tdc01, tdc02, ... (all channels format)

or if you prefer sparse format:

TIME lunx; 1=tdc02; 14=tdc14

Houston/ICC Data Formats

ICC sends commands as simple text lines, of format ([ ] indicate optional items):

```
    [id] command [keyword=value ...]
```

where id is a number chosen by ICC, which Houston returns in all responses to this command. For debugging, and hand typing commands, note that 'id' is optional, and defaults to 0.

Example: turn on debug mode:

```
    483 set debug=1
```

The "set" command is commonly used.

Note that this interface is primarily designed as a machine-machine interface, so is not especially human-friendly. However, in practice, while the TUI-Apollo and the ICC are under development, humans must frequently connect to Houston Control (housctl), and send ICC commands directly, and read the responses. Hence, there are some human-friendly features in housctl.

WARNING Consistent with a machine-machine interface, error checking is minimal. For example, mistyping a number in a command usually results in a value of zero. For example, typing "power q 0" will turn off the CAMAC power, because 'q' is not a valid number, and defaults to a value of 0.

housctl Parameter Types

Note: I think housctl's parameter model is quite different from TUI, where call-back functions make setting a parameter an *active* operation that can have any immediate effect.

housctl has 3 parameter types: getables, setables, and reportables.

Getables are parameters that ICC can request on demand, e.g.

```
    get state
```

Setables are parameters that ICC can set:

```
    set nruns=500
```

All setables are also getables. Getables and setables can be either integer, floating-point, or string. Each parameter has a fixed size and precision, which normally the operator needn't know about. Getables and setables are all single-valued, i.e. they cannot have a list of values (which the APO protocol allows).

Setting a parameter usually has no immediate effect; it simply sets a value that will be used later in the execution of housctl commands. There are some exceptions to this, because housctl is constantly using some parameters, such as 'state'.

Reportables are parameters that housctl sends to ICC/TUI, but cannot be "gotten" or set. Reportables can be either single or multi-valued. E.g., 'powerstate' is multi-valued:

```
    par0 powerstate=1,0,1,1, 1,1,0,0, 2, 0,1,0,0, 1,0,0,0, 1,1,2,2, 1,1,1,1
```

housctl sends reportables either unsolicited, or as responses to some commands.

Command Summary

In general, commands with question marks are queries for operator help, rather than for system status.

```
    [ ] => optional parameters;  { } => exactly one value is allowed
    get     keyword [...]           get parameters
    set     keyword=value [...]     set various arbitrary parameters
    status                          returns all getable parameters
```

```
power                    show current stored power status
  power check            query each device, and show current status
  power dev {0, 1}       turn AC power off/on for device
camac                    return fun CAMAC facts
camacz                   reset CAMAC
bolo {0, 1}              put bolometer (laser power meter) out/in laser path
rxx angle                angle is float, ~arc-sec.  Nudge rx path x-mirror
rxy angle                ditto, but in y-direction
vtarget vtargetx vtargety        record rx-mirror target offsets
vmove vx vy              move rx-mirror to (vx, vy), arc-sec in mirror frame
vcalibrate               define current rx-mirror pos as (vtargetx,
vtargety)
idle                     enter idle state; equiv to 'set state=1'
warmup                   enter warmup state; equiv to 'set state=2'
run [runcomment]         enter running state; equiv to 'set state=3',
                         but adds the comment to the data file.
standby         enter standby state
cooldown                 enter cooldown state, closes data file
stare [comment] enter stare state, with 'comment' -> data file.
dark [comment]           enter DARK state, with 'comment' -> log file
fidlun                   enter fidlun state, no comment allowed
readpoly [filename]      read polynomial from filename,
                         Defaults to current value of 'polyname'
readblock                read laser block times from housctl.blk
time                     reports the fractional day and RTT to moon
cums                     write cumulative parameters to disk
disconnect               houston disconnects ICC TCP connection
                         This is MUCH safer than dropping from the ICC side
laser {powerup, keyoff, keyon, keycycle, warmup, stop, start,
   preprun, shutterclose, shutteropen, display} Laser control sequencing
m33 [{off, on, setpoint, lowlimit, highlimit}]  chiller ctl, blank for
status
m75 [{off, on, setpoint, lowlimit, highlimit}]  chiller ctl, blank for
status
help                     print this help
```

To exit Houston Control, send “set state=0”. This causes a careful shutdown of all systems, including temperature and other regulating systems, and housctl exits.

Examples (**bold** denotes ICC->housctl, normal is housctl->ICC):

```
get state gatewidth xyz
0 i state=1
0 i gatewidth=180
exc0 Unknown get parameter: xyz
0 w text=”Unknown get parameter: xyz”
0 :

34 set predskew=2
34 i predskew=2
34 :
```

Device Power Codes

The “`power`” command encodes AC Power plugs, and the M33 and M75 chillers.

```
From: Tom Murphy
Sent: Monday, May 23, 2005 19:16
To: apollollr@u.washington.edu
Cc: dwoods@apo.nmsu.edu; Mark Klaene; bketzeba@apo.nmsu.edu
Subject: [LLR] APOLLO Power Scheme
```

As arrangements for switching power become more well defined, I am trying to keep track of the scheme. Here is what I have at present. Power will be controlled from a variety of devices: Some will be switched via the UW relay boxes, receiving digital controls (7 total) from the National Instruments DAQ card on houston. Some will be controlled via the UW relay box receiving digital input from the terminal server (via port control lines). Some will be switched via the 4-port IP-accessed power switch in the phone-booth/cabinet.

The UW relay boxes are each 4-outlet units, and there are 4 such boxes. Labeling these boxes 1, 2, 3, 4 and the outlets on each a, b, c, d, we can lay out a map designating who is where, and which signals control each.

```
Box 1 is in the Utah laser enclosure
Box 2 is in the phone-booth/cabinet
Box 3 is in the phone-booth/cabinet
Box 4 is in the intermediate level enclosure (ILE)
```

On the terminal server (located in the ILE), ports 1-8 are reserved for communications. Ports 9-16 are usable for relay switching (this division is arbitrary).

The IP power switch is plugged into the UPS.

The devices controlled by the NI-DAQ on houston are as follows:

bit #	box/outlet	equipment	max pwr
0	2a	APD Power supply	
0	2b	Photodiodes/9327 supply	
1	1a	CAMAC crate	175
1	2c	Booster power supply	35
2	3a	T/R motor power supply	65
2	3b	diffuser motor pwr sup.	
3	3c	STV CCD camera ctrl	25
3	3d	NF optics actuators	35
4	1b	Utah box heater	110
5	1c	Noren Lower fan	
6	1d	Noren Upper fan	

Updated by Eric Michelsen, 10/29/2008: The devices controlled by the parallel port are:

port #	box/outlet	equipment	max pwr
9	4a	Plenum fan banks	
10	4b	ILE exhaust fan	
11	4c	Aux. cooling pump & fans	
12	4d	Cabinet circulation fan	
13	--	Laser keyswitch relay	<1
14	--	Laser 208 V 3-phase (3 relays)	2500

The devices supported on the UPS-fed IP switch are:

port#	equipment	max pwr
0	houston	150
1	flowmeter box	
2	RTD box	
3	GPS clock	

Then there are some devices left always powered:

type	equipment	max pwr
UPS	XL-DC clock	25
UPS	Interlock box & shutter	
utility	Laser Power Meter	3
utility	ILE heater (own thermostat)	420
220V	M33 Chiller (RS-232 activated)	1500
220V	M75 Chiller (RS-232 activated)	2000

Gettable/Settable Parameter Summary

Following is a response to an ICC **`help`** command. It lists all the *gettable* and *settable* parameters and their current values. If no "set" commands have been issued, the parameters all have their default values (below are as of ??). *Note that default values are subject to change as needed.* Always use the help command in the version of housctl that you actually use to find its default values.

```
          logfile ="060512.log"      log file name
         datafile =""                data file name
            state = 1                0=EXIT, 1=IDLE, 2=WARMUP, 3=RUN,
4=COOLDOWN, 5=STARE, 6=FIDLUN, 7=STANDBY, 8=DARK, 9=FLAT, 10=LPOWER,
11=CALTDC,  30=TEST, 31=TEST2
      icc_session = 1                ICC session: 0, 1, or 2
            debug = 0                debug level
        houstbase = 1147466408986    epoch time base
         houstime = 3753294          execution time
      logfile_due = 83991014         time to next logfile
           motrps = 0.               T/R speed setpt, rps
    t_utah_center = 20.25            Utah center temp
      t_utah_push = -0.25            Utah push temp offset
     t_utah_limit = 0.75             Utah limit temp offset
        t_ile_low = 20.              ILE low temp
         t_ile_hi = 25.              ILE hi temp
  t_ile_alarm_low = 8.               ILE alarm low temp
        reflector = -1               Lunar reflector: 0=Ap11, 2=Ap14,
3=Ap15, 4=Lunakod
         polyname ="housctl.poly"    polynomial file name
            nruns = 5000             RUN, FIDLUN # of shots to make
        flashrate = 20.              FIDLUN flash rate
      mirrorphase = 170              encoder to fire laser
    dphase_target = 850              quadrant diffuser target phase, -1
disables
           dphase = -1               quadrant diffuser phase
            dskew = -41.             RUN delay skew (fudge)
        huntstart = 0.               Time-hunt starting offset, ns
        huntdelta = 0.               Time-hunt increment, ns
            thunt = 0.               Time-hunt current offset, ns
         predskew = 0                tpred = ppred + predskew
        gatewidth = 8                RUN lunar, STARE, FIDLUN width, ACM reg
         runfidgw = 6                RUN fiducial gatewidth, ACM register
        starerate = 1000             STARE, DARK gate rate
          nstares = 20               # STARE records to make
          binning = 500              STARE binning
           ndarks = 10000            DARK/FLAT # gates to take
         vtargetx = 0.               Velocity offset mirror x-target
         vtargety = 0.               Velocity offset mirror y-target
         flashcum = 147701           Cumulative flash count
          rxxpcum = 0.               Cumulative optics x+ offset
          rxxncum = 0.               Cumulative optics x- offset
          rxypcum = 0.               Cumulative optics y+ offset
          rxyncum = 0.               Cumulative optics y- offset
            vposx = 0.               Velocity offset mirror x-position
            vposy = 1.5              Velocity offset mirror y-position
     ampdelay_low = 7500             Laser low power amp delay
    ampdelay_high = 0                Laser full power amp delay
          bolopos = -1               bolometer: 0/1 = out/in
       laserpower = 1                ampdelay: 0/1 = low/high power
          fakertt = 0.               Forced round-trip time
             topt = 0                test options
         doptions = 0                diag options: 1=No Noren off, 2=no term
serv, 4=Noren on heat
     energy_count = -1               pulse energy convergence iteration
          airtemp = -99.             weather air temperature, deg C
         pressure = -99.             weather pressure, mbar
         humidity = -99.             weather humidity, %
         guideOff =""                guide offset
          boreOff =""                boresight offset
           axePos =""                axes position
       slewtarget =""                telescope pointing target
      las_display =".................:....................:...." laser
control box display
    circulate_temp = 2               Temperature to enable laser DI
circulation
        laser_log = 0                Turn laser display log off/on=-1,1
```

```
   flow_interval = 30000           Flow measurement interval, ms
          alarms = 0               :4..0: utah_temp, ile_temp, write_cum,
read_cum, di-low-flow
    alarms_unack = 0               bits 18..16: laser_block, ICC_cmd,
nonspecific
     ts_errorstr =""               term serv error string
    wti_errorstr =""               IP switch error string
    tcp_errorstr =""               TCP lib error string
    las_errorstr =""               laser lib error string
   chil_errorstr =""               chiller lib error string
    pwr_errorstr =""               power lib error string
     pr_errorstr =""               Programmable resistor error string
```

The '`status`' command also returns

- the APD time offsets (from fast photodiode to fiducials, in TDC units) as `apdtofpd=636.207,688.602,0,654.450,0,671.388,675.043,0, 687.673,673.837,662.598,614.974,648.818,629.876,0,668.248`
- the latest drk0 record, if any. If there has been no DARK done since housctl was started, housctl sends no drk0 record. This allows TUI to retain DARK counts across housctl restarts.

Response Summary

Houston sends 3 kinds of records to the ICC:

(1) ICC records: keyword=value responses, that start with a cmdid and status character;

(2) log records: measurements and other data written to the log files, that start with a 4-character record identifier (no cmdid or status char). All log records (measurements except for fid and lun) also use keyword=value format for their data.

(3) fid & lun records: high volume time measurements of fiducial and lunar events.

ICC Records

Houston sends ICC records both in response to ICC commands, and unsolicited. Keyword=value responses follow the ICC/hub format:

`id status keyword=value [;...]`   convey values of parameters

Houston sets id=0 for unsolicited data. Status is one character, per the ICC/Hub interface:

```
: command finished successfully
i information
w Warning: something is unusual, but the command will continue to execute.
> command queued; final response will come later
f command failed
```

Houston does not currently send the "!" (fatal error requiring reboot) status char.

The keywords can be any keyword of any type; i.e., any of the settable parameters, plus any of the measurement parameters from any record type.

Log Records

Houston sends log records verbatim to ICC, because some if it is interesting to TUI and the operators. See section "Log File Record Types" for log-record descriptions.

fid and lun Records

The fid and lun records compose over 99% of both the data recorded on disk, and the traffic Houston sends to ICC. Its volume is noticeable, and hence fid/lun records are "compactified" by omitting keywords. Individual fields are identified by parsing the known record format.

Errors

There are two kinds of errors: ICC command errors, and exceptions. housctl responds to ICC errors with the APO standard “f” and “w” status characters described above. housctl responds to exceptions with “exc” exception records (described in “Events and Alarms”).

TUI Functions and housctl/TUI Interfaces

APOLLO TUI must perform the functions described here, in addition to others documented in TUI documentation.

housctl uses the concept of a “record” of information from housctl to TUI. Many parameters are interpreted within the context of the record in which they appear. For example, the “i=...” augments the information from other parameters in the same record. TUI has the ability to keep the entire text of each record available for any parameters later processed in that record. For example, when housctl replies with an “f” (failure) status, TUI displays the whole reply, and then processes each parameter in the record.

General TUI Processing of Housctl Messages

TUI maintains 3 display boxes (possible on different tabs): (1) a housctl reply box, (2) a hub log box, and (3) an alarm text box. Each box is a scrolling window of text.

(1) **housctl reply box:** displays all command replies (i.e., solicited), and all “i=...” parameter text. All “text=...” should display in red. Note that “f” status and “text=...” also go in the alarm text box.

(2) **hub log box:** displays all traffic from the hub. Operators can filter it to only show APOLLO messages ??, or to have those from APOLLO ICC highlighted..

(3) **alarm text box:** display all failed command replies (“f” status), and “text=...” parameters. Note that “f” status also goes in the housctl reply box. See also “Events and Alarms,” below.

TUI should display all solicited messages from housctl, in the housctl reply box. Messages with “i” status (or “>”) display in normal text, “w” status in blue, and “f” status in red. It also display here additional information as described elsewhere. Note that “f” status also goes in the alarms box.

Keyword Parameters Requiring Additional TUI Processing

TUI should process all keywords the same, whether solicited or unsolicited. Note that there is a command reply status “i”, and a parameter “i”, but they are two different things.

powerstate	TUI should use the ‘powerstate’ variable for its Power tab. This keeps the power state up-to-date. Note that the states are 0-7, not just 0 or 1. A powerstate of -1 means that number is not connected. (As of 10/5/2007, TUI seems to take the power state from the ‘powerstatus’ strings.)
polyname	TUI should prominently display the poly file name, and set polyname window red if blank. Note that we hope to soon replace “polyname” with “predictname”, since we hope to eliminate the polynomial fits altogether.
blockremaining	Seconds remaining to block laser. During a space-command blockage, housctl sends an unsolicited “blockremaining=...” message occasionally. housctl sends the messages every few seconds near the end of the blockage. TUI should display this information, possible color coded. During a blockage, the main tab should display blue.
releaseremaining	Seconds remaining with laser released. During a space-command release time (allowed to lase), housctl occasionally sends an unsolicited “releaseremaining=...” message. Near the end of the release time, housctl sends them every few seconds.
statusline	TUI should display this text on the status line of the entire window, visible on any tab.

others? there are probably several parameters missing from this list

housctl debug parameters: TUI simply ignores any parameter it does not understand. There are many parameters that are just for human interpretation.

See also the list under “Events and Alarms."

Other Keyword Parameters

See the list under “Events and Alarms."

Events and Alarms

The event/alarm model is of global importance: every piece of the system must use it consistently.

It appears that there is no pre-existing event scheme in APO-standard parts of TUI, so housctl chose a simple, consistent model. Here’s some background information:

Typical industry event models define **event** as an unsolicited happening *with data associated with it*. The data are often transient in nature, and cannot be read after the fact (such as how far off the diffuser phase was before the system corrected it). All events would have some common data fields: timestamp, event-type, severity. Each type of event then has type-specific fields; e.g., the “diffuse phase error too big” event would include the phase error that triggered the event.

An **alarm** is just a high-severity event. However, usually, “alarmed” is also a state of the system. The system has a model for how alarms get cleared after they are declared. There are typically 3 kinds of alarm, defining how long they last and how they are cleared: (1) transient, (2) persistent-auto-clearing, and (3) persistent-operator-cleared.

Transient alarms are by nature transient, and do not persist for any length of time; e.g., “photon count too low” during the last measurement interval. The system never actually enters “alarmed” state from a transient alarm.

Persistent-auto-clearing alarms persist in “alarmed” state as long as the alarming condition persists; e.g., “cable unplugged” persists until the cable is plugged in, when the system clears the alarmed state by itself.

Persistent-operator-cleared alarms remain “alarmed” until the operator manually clears it; e.g., “safety door open”: even if the door swings shut, you might want to be sure the operator checks that it is secured (later we’ll see that the “unacknowledged alarms” record might also serve this purpose).

The choice of which kind of alarm a given condition will be is subjective, and driven by needs.

System-management systems (TUI, log-files, APOLLO system manager) typically log all events, and provide some way for operators to scan/cull/filter/sort through the list.

If we regularly get an alarm, and we're not alarmed by it, then it shouldn't be an alarm. When ignoring an alarm is regular practice, it encourages people to ignore alarms, which completely defeats their purpose.

Question Does APO maintain an IP Syslog server that is useful to APOLLO? If so, would it help if our exceptions, or environmental failures, were logged there?

housctl Alarms

housctl currently (7/2008) implements only transient alarms and persistent-auto-clearing alarms. However, each alarm sets one of the bits in the ‘alarms_unack’ parameter; when an operator connects, ‘alarms_unack’ provides a summary of what’s gone wrong before the operator connected. Operators must review exception logs to find details of any such alarms; usually the APOLLO System Monitor web page is current enough. Operators clear 'alarms_unack' when they choose, typically when they have understood the alarms.

housctl alarms generate 'exc' records in the log & data files. housctl generates both *persistent auto-clearing* and *transient* alarms. Persistent alarms put housctl in an "alarmed" state, and continue until the cause of the alarm is removed (self clearing). For example, if the laser DI flow is low, housctl remains alarmed until the flow is restored. Per APO's original ICC recommendations, housctl maintains current persistent alarm state in bits of the getable "**alarms**" parameter. Like all getable parameters, "alarms" is included in the 'status' command response.

In addition, any event that sets a bit in the "alarms" parameter, also sets a bit in the "**alarms_unack**" parameter. When alarms clear, housctl clears the "alarms" bit, but does not clear the "alarms_unack" bit. Operators "acknowledge" past alarms simply with 'set alarms_unack=0' (or any other value, if operators want to acknowledge only some of the bits set). In this way, operators can easily see if any alarms have occurred since the last acknowledgement.

Transient alarms are caused by one-time events, for which there is no corrective action, and therefore no "alarmed" state. For example, a communication error (checksum error) is a transient alarm. It generates an 'exc' (exception) record, but no further action. It sets a bit in the "alarms_unack" parameter, but not in "alarms", because by definition, there is no such thing as a "current transient alarm."

There are currently (7/2008) these alarms defined. These are likely to change over time:

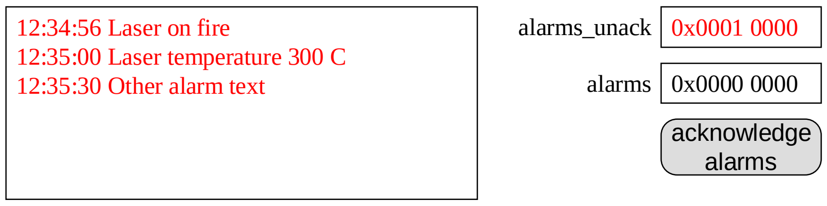
```
// Persistent alarm bits:
#define   AL_DILOWFLOW 1      // laser head DI flow low
#define   AL_READCUM   2      // can't read cumulative value file
#define   AL_WRITECUM  4      // can't write cumulative value file
#define    AL_ILETEMP  8      // ILE temperature out of bounds
#define   AL_UTAHTEMP  0x10   // Utah temperature out of bounds
#define   AL_LASERBLOCK 0x100 // invalid laser blocking file

// Transient alarm bits:
#define   AL_NONSPECEXC 0x10000      // nonspecific exception
#define AL_QOVERFLOW   0x40000      // lunar queue overflow
```

The nonspecific-exception is any exception that does not have a specific alarm bit assigned to it. We can define more alarms (and their bits) as needed.

TUI Processing of Alarms and Events

See also the "General TUI Processing of Housctl Messages," above.

TUI maintains an "Alarm" tab. It includes a "acknowledge alarms" button, which sends a "set alarms_unack=0" command. The alarms part of the screen might look something like this:


12:34:56 Laser on fire
12:35:00 Laser temperature 300 C
12:35:30 Other alarm text

alarms_unack 0x0001 0000

alarms 0x0000 0000

The 'alarms_unack' box is writeable by the operator (like many parameters are). With this, operators can selectively clear unacknowledged alarm bits.

TUI sets the alarm tab to one of 3 colors:

normal when alarms=0, and alarms_unack=0

pink	when alarms not 0, and alarms_unack=0
red	when alarms_unack not 0

The APO TUI standard "warning" color is blue, which TUI uses for "w" responses.

Alarm and Event Keywords from Housctl

text	This should be called "error_text": Displays in red in the housctl reply box, and the alarm text box.
i	This should be displayed in the housctl reply box, which allows housctl to get unsolicited information into the housctl reply box.
h	help text. Someday, TUI could accumulate all help text into a tab, which operators could switch to for online text help.
w	(not implemented as of 10/2007) This could be like 'i', only displayed in blue.
alarms_unack	TUI displays alarms tab in red if this is non-zero. If zero, tab color depends on 'alarms'. See "Events and Alarms". This is state, and housctl sometimes sends it redundantly, so TUI *should not log anything* in any text windows on receiving this parameter. TUI displays 'alarms_unack' in hex in a box beside the Alarm text box: red if non-zero, black if zero.
alarms	TUI displays alarms tab in pink if this is non-zero, and alarms_unack=0. See "Events and Alarms". This is state, and housctl sometimes sends it redundantly, so TUI *should not log anything* in any text windows on receiving this parameter. TUI displays 'alarms' in hex in a box beside the Alarm text box: red if non-zero, black if zero.
g	arbitrary text for humans. TUI should not do anything special with it, and let it be displayed normally with the command reply in the housctl reply text box.

Exception Records from Housctl

When an unexpected event occurs, housctl sends an exception record, with related parameters, e.g.

```
    exc0 2007-01-04T17:36:54 severity=1; bits=0x20000;
                                 text="Unknown 'set' parameters: refl=3"
```

Keywords:

`severity`	int	0=information, 1=warning, 2=severe
`bits`	int	alarm bit that was set
`text`	string	human readable text

housctl clears an alarm with an exception record like this:

```
    exc0 2008-07-08T02:59:14 clear; bits=0x8; alarms=0x0; alarms_unack=0x10008;
    i="ILE temperature 24.95"
```

The "clear" parameter indicates this is an alarm clearing. The 'alarms' & 'alarms_unack' parameters give the current alarm state after the cleared alarm. Note there is no 'text' parameter, but there may be an 'i' parameter.

Possible Event/Alarm Future Enhancements

*Add a warning for moderately high temperatures, that are not yet alarming. Things like "a little high" temperature are exactly what "warnings" are designed for. I think a warning would work well for moderately high temperatures.

We must then define a housctl/TUI interface for the passage of these warnings. One possibility is that TUI interpret the 'text' keyword in the context of its accompanying 'severity' keyword (this is a *little* stateful). Another possibility is that housctl use a 'warn_text' keyword instead of 'text' for warnings. The APO standard for warnings in TUI is to display them in blue. We might want the alarms tab color to then have some blue indication of warnings, but that's not trivial. There is no warning state, so there could only be "unacknowledged warning" status. Maybe just blue log lines without tab color is enough.

* housctl could maintain a list of the error text for each bit set in the 'alarms' parameter. When TUI connects, housctl could upload this list as part of 'status'. TUI could use this information to augment the Alarms tab.

* We could provide more information with status variables: for each event type, housctl would maintain a set of variables:

1. event-type
2. # times event happened since last reset
3. timestamp of last occurrence
4. timestamp of last reset
5. - n: snapshot of event-specific data associated with the last event of this type

* Interlock enhancement

```
   From:  James Battat
   Sent: Monday, November 13, 2006 13:57
   To: Eric Adelberger
   Cc: apollo_core@u.washington.edu
   Subject: Re: [Apollo_core] (no subject)
```

In thinking about Eric Adelberger's comment on the APOLLO interlock/warning system, I've come up with a few potential failures that we should certainly be alerted to when they occur. By no means is this list complete.

- Houston down
- gps antenna lost lock (prevents detection of lunar photons)
- cocoa down (icc)
- flow meters stopped (perhaps combined with temperature readings)
- (danger of freezing)
- ILE temp too high/low
- Utah temp too high/low

Of course, we have to determine what is meant by "alerted" (email?, to whom?). As far as I know all the alerts/alarms that we have implemented require the user to be looking at the right website or screen and can therefore easily go unnoticed. To do this right will likely require a significant amount of system (re)design.

Let's at least come up with a more complete list of failures that we must be alerted to, to help inform our interlock/warning system design. Please send your suggestions. I am happy to compile them for the group.

In some instances, we may have to build in some flexibility/tolerance into the system in the event that our sensors (e.g. RTDs) are not doing their job correctly.

```
   On Mon, 13 Nov 2006, Eric Adelberger wrote:
```

We need to have a more sophisticated interlock system that looks at all recorded parameters and checks if they are within their proper ranges. If any one is out of range, it announces this in a prominent way (forcing the observer to respond) which parameter(s) are out of range.

TUI/Hub/ICC Information

Hub capacity:

```
From: Craig Loomis [mailto:cloomis@apo.nmsu.edu]
Sent: Friday, December 03, 2004 15:34
To: Tom Murphy
Cc: emichels@physics.ucsd.edu; hoyle@npl.washington.edu;
owen@astro.washington.edu
Subject: Re: APOLLO architecture and dataflow
```

It appears that you can run ~100 replies/sec through the hub with three listening TUIs. The hub itself barely noticed that traffic, for what that's worth. The size of the replies is not important, btw.

Hardware & Low Level Interface:

```
From: Craig Loomis [mailto:cloomis@apo.nmsu.edu]
Sent: Monday, August 02, 2004 19:09
To: Eric L. Michelsen
Cc: 'Russell E Owen'; 'Tom Murphy'
Subject: Re: architecture
```

We have a 100Base-T switch in the intermediate level. We run copper to that from all but the _most_ essential/expensive devices. But if you want fiber, TX/FX converter boxes are cheap & easy. SC connectors preferred; ST connectors accepted.

```
> For our ASCII command interface, are there any Telnet/NVT (Network
> Virtual Terminal) requirements?  It doesn't seem like it to me.  Can we
> just open a TCP connection, and read/write ASCII to it?
```

Opening a TCP socket and chatting is the preferred method. If you will be sending commands, I will assign you a pseudo-userID that the obs-specs can use to disable your commands.

ICC

The current ICC reference is `http://rowen.astro.washington.edu/ICCManual/`.

Here are a few details from Russell that may be of interest:

- The communication channel from the mountain to the rest of the world is rather slow. If this is a problem for Apollo then it may have to be run on site.
- Hub/ICC communication is usually an ASCII command/reply stream, plus perhaps data to a shared disk.

However, there is also a binary version of the hub/ICC protocol which some older instruments use to send back data. This is discouraged because it ties up the command/reply stream while data is being read out.

- TUI/Hub communication is only ASCII. TUI also can automatically fetch images via ftp. In theory TUI can open additional lines of communication if necessary, but it makes for a messy architecture.
- The Hub supports ICCs that use an RS-232 interface for the command/reply stream (via a terminal server). If you prefer direct TCP, that's fine.

The "hub" is a computer that lives at APO in the computer room. The hub routes commands from "commanders", such as TUI users, to "actors" such as:

- the instruments
- the Telescope Control Computer (TCC)
- various other scripts, including perms (a permission system)

The hub tries to do very little other than authorize users and route commands. Authentication and some of the scripts are documented at <http://tycho.apo.nmsu.edu:81/MC2/>.

The hub passes commands "unmolested" to the various Instrument Control Computers (ICCs). However, it expects replies to be in standard keyword-value format (more on that in the P.S.).

The TCC is the Telescope Control Computer. It also lives at APO in the computer room. See <http://www.apo.nmsu.edu/Telescopes/TCC/TCC.html> for info on the commands and expected replies.

There is no standard name for a client machine running TUI. The person running TUI is a "user".

TUI: <http://www.astro.washington.edu/owen/TUIHelp> (this is a copy of the html help built into TUI).

Hub: <http://tycho.apo.nmsu.edu:81/MC2/>, and in particular, the link Authentication.

The syntax for command/reply format for your instrument control computer (ICC):

- All commands and replies are ASCII
- Each command and each reply should take only one line. However, there is no line length limit. Also, one command can generate more than one reply.

The format for commands to your ICC is up to you (within the rules given above). However, if you can execute more than one command at the same time, or if you permit unsolicited output, then you must allow an initial command ID integer (which will then be included with all replies for that command). Even if you don't meet the requirements you may want to accept a command ID integer; most of our ICCs do. I suggest making the command ID optional (defaulting to 0). This makes it easier to test by hand.

Here are some example command sets that are simple and possibly worth emulating:

- DIS <http://tycho.apo.nmsu.edu:81/DIS/DIS_Commands.html>
- expose and tlamps: see links from <http://tycho.apo.nmsu.edu:81/MC2/>

Replies from your instrument controller MUST be in APO-standard keyword-value format. This is, by example:

```
    cmdIDInt cmdStateChar keyword1=val11, val12,...; keyword2; keyword3=value3
```

- cmdIDInt is the command ID integer for the command which triggered the reply. If your output is not in response to any command (i.e. unsolicited), then use 0. If you don't allow command ID numbers then always use 0.
- cmdStateChar is one of the following characters:

`:`	command finished successfully
`>`	command queued (this is only relevant for systems that can execute more than one command at a time, e.g. the TCC)
`i`	information
`w`	warning
`f`	command failed
`!`	system failure (e.g. the command interpreter or a subprocess died); init or reboot required to proceed

- Solicited replies (triggered by a command) must end with exactly one ":", "f", or "!" reply. Before that, you may issue as many "i" or "w" replies as you like. (You may also issue more than one ">" reply, but it violates the intent of that message code.)
- Unsolicited replies (not triggered by a command, thus CmdID = 0) do not have any such restrictions.
- keywords are words. I suggest you stick to legal C identifiers. Case is ignored.

Typically the user interface or hub will set up a callback to be triggered by a particular keyword, so its state display can be kept up-to-date. For these to work well, keywords should:

- Always be output when the relevant state changes. Remember that we can have more than one user at the same time. Other users should not have to parse commands to know what is going on. For example, if one user says "focus=50", you should output a keyword indicating the new focus.
- Self-contained and non-modal. The keyword data is much easier to use if all one has to know is the values for that one keyword, rather than looking at the rest of the reply, or worse, knowing what the ICC was doing at the time.

Examples:

```
      Bad: Duration=... (duration of what?)
      Good: PulseDuration=...
      Bad: MirrorID=1; MirrorData=data1, data2...
           (the MirrorData user also has to look at MirrorID)
      Good:
        if you only have a few mirrors:
          SecData=data1, data2...
          PrimData=data1, data2....
        if you have a lot of mirrors:
          MirrorData=id, data1, data2...
```

You will probably also want a keyword for command warnings and error messages (data for the user to read, but not for the user interface to parse). I suggest simply using the keyword Text for all these.

Keyword values can be:

- Quote (")-delimited strings (escape any contained " or \ with \)
- numbers (int or float or nan; ints can be in any base using C notation)
- dates and times (yyyy-mm-dd hh:mm:ss.sss or yyyy-mm-ddThh:mm:ss.s) Please stick to UTC (or TAI if leap seconds are an issue); avoid local time if at all possible
- keywords (same rules as keywords)

Telescope Control Computer (TCC)

The TCC is essentially an ICC for the telescope itself.

```
   From: Russell E Owen
   Sent: Monday, July 11, 2005 15:28
   To: apollollr@u.washington.edu
   Subject: [LLR] TCC keywords for offset
```

Here is the output of the TCC for a sample guide offset.

```
   10944 5 > Started; Cmd="offset guide 0.000139,0.000042"
   10944 5 I MoveItems="NNNNNNNYN"; Moved
   10944 5 I GuideOff = 0.000139, 0.000000, 4627497670.43000, 0.000042,
   0.000000, 4627497670.43000, 0.000000, 0.000000, 4627497670.43000
   10944 5 :
```

The keywords are described in the TCC Message Keywords Dictionary: <http://www.apo.nmsu.edu/Telescopes/TCC/MessageKeywords.html>

Typical keywords for a given command are listed with each command in the TCC Commands Manual: <http://www.apo.nmsu.edu/Telescopes/TCC/Commands.html>

P.S. This output is for an uncomputed offset, as usual for a guide offset. Uncomputed offsets are small offsets which happen quickly but with known end time and no jerk limiting. The other option is a computed offset. Computed offsets are used for large moves or when you really need to know when the move is finished. Computed offsets are basically slews; they take longer and generate much more output, but are much gentler on the drives and report an accurate (if conservative) end time.

6 APOLLO Instrument Control Computer (ICC)

ICC is a function, not a piece of hardware. For a long time, we have run ICC on Houston itself. We plan to soon migrate the ICC function to the 1U rack-mount computer "cocoa".

The main purposes of ICC are these:

(1) to pass commands from the Hub to Houston, and

(2) to pass APO standard responses from Houston to the Hub, and to reformat log/data records from Houston into APO standard format, and send to the Hub.

ICC Processing Details

Houston to Hub Direction

ICC should perform default processing on all unrecognized record/version types, so Houston and TUI can easily change without changing the ICC. Such transparency is standard encapsulated software design.

I'd like to have ICC changed to strip timestamp-like things: that on all record types, it checks the 2nd "word" (delimited by spaces). If this "word" starts with a number of digits and a hyphen, ICC discards that word. For example:

```
    tmp0 2007-01-01T12:12:12 dome_air=5
```

would be equivalent, from ICC's point of view, to:

```
    tmp0 dome_air=5
```

Note that ICC must preserve all the numeric fields in data records such as fid0, lun0, str0, etc, which start with numbers (but no hyphen).

This would allow me to remove the long-standing kludge for DRK and FLT records, as well as allow us to add timestamps to any record type without needing to change ICC again.

Log records:

First, for all record types, whether recognized by ICC or not, check for and remove any 2nd word which starts with a digit. For example:

```
    tmp0 2007-01-01T12:12:12 dome_air=5
```

would be equivalent, from ICC's point of view, to:

```
    tmp0 dome_air=5
```

Then, other record-type-specific processing as follows:

default: ICC should strip any time stamp-like thing from records (described above). E.g.,

```
    exc0 ansitime a=b; ... becomes        0 i exc; a=b; ...
```

rem and **icc0**: ICC discards rem and icc* records (i.e., icc records of any version). Should we change ICC to pass them on so all TUI users can see what others are doing??

par0: ICC changes par0 records as

```
    par0 key1=val1; key2=val2 ...        becomes
    0 i par; key1=val1; key2=val2 ...
```

drk0 and **flt0**: ICC changes dark records as

```
    drk0 ansitime 10 5 ... 22     becomes
    0 i drk; Dark=123, 190, 0, 225, 10000, 126, 165, 225, 212, 146, 317, 233,
    285, 336, 0, 274
```

Note that the timestamp is handled by the general timestamp removal function.

str0: ICC changes stare records as

```
str0 ansitime 10 5 ... 22     becomes
0 i str; Stare=175, 193, 0, 261, 10000, 163, 190, 263, 175, 155, 348, 269,
294, 359, 0, 254
```

Note that the timestamp is handled by the general timestamp removal function.

Note: ICC need not know all the record types, as it should apply the default processing to any unknown record type.

Data (fid/lun) records: require the most processing. Note that ICC discards TWS and FRC, leaving:

```
Fiducial=1, 0, 1500, 1912, 3, "5:3970, 13:2755, 15:1912"
```

The numbers are shot #, Accumulated Sec, pulse energy, photodiode TDC, "chan:time..."

```
Lunar=1, 0, 1912, 1912, 2, "5:1590, 14:0572"
```

The numbers are shot #, Accumulated Sec, ppred, tpred, "chan:time..."

In the TUI log, fid and lun records look like this:

```
21:53:27 .apollo.0 0 apollo i Fiducial=1,0,-438,0,2000,2,"1:2001, 15:2000"
21:53:27 .apollo.0 0 apollo i Lunar=1,0,2000,2000,1,"1:2001"
```

Hub to Houston Direction

Hub commands can be intended for either ICC itself, or for Houston. ICC recognizes and intercepts the very few ICC commands. They are the "built-in" Apollo ICC commands:

```
connDev  [dev1 [dev2...]]
```

Connects ICC to houston

```
disconnDev [dev1 [dev2...]]
```

Disconnects ICC from houston. Does this really work??

```
exit      Log me off (leaving everything else running)

quit      Same as exit

users     Show info about users

houston [cmd...]        Send cmd... to houston ("unmolested")
```

This last ICC command, "houston" tells ICC to pass the remaining text of that command as-is to Houston, as a Houston command. For example, if ICC receives from Hub

```
cmdid houston set blah=3.14
```

then ICC sends to Houston:

```
cmdid set blah=3.14
```

Currently ICC also recognizes several Houston commands explicitly, and passes those to Houston even without the "houston" command. TUI does not currently (7/2008) use this feature of ICC. Ideally, since 99+% of all traffic is for Houston, if we want to economize to remove the "houston" word on Houston commands, the ICC should pass all unrecognized (i.e., non-ICC) commands to Houston. Again, this allows Houston and TUI to change without requiring ICC changes, and is standard encapsulated software design. ICC need not know explicitly about Houston commands. Nonetheless, the known houston commands that are sent directly through to Houston without need for a preliminary "houston" command, as of 9/26/2005, are:

```
HoustonCmds = (
    "status",
    "set",
    "get",
    "camac",
    "bolo",
    "rxx",
    "rxy",
    "stare",
    "idle",
    "warm",
    "run",
    "readpoly",
    "disconnect",
```

We are not adding to this list, because we don’t need or use this feature.

ICC Design Notes

```
    Subject: Some thoughts on the Apollo ICC
    From:    "Russell E Owen" <rowen@u.washington.edu>
    Date:    Thu, July 14, 2005 7:28 am
    To:      "Tom Murphy" <tmurphy@physics.ucsd.edu>
             "Craig Loomis" <cloomis@apo.nmsu.edu>
```

I was thinking about how the Apollo ICC could command the hub.

The "RO" package (a utility package distributed as part of TUI) includes code to parse hub messages and call functions based on code that registers interest in a particular keyword (RO.KeyVariable and RO.KeyDispatcher are the pieces that are most directly relevant). In particular, the key dispatcher just needs to be fed a hub message to the "dispatch" method do its magic.

You can use all this within the confines of the "select" loop, though it may require very minor changes to deal with the lack of Tkinter (the code in question doesn't care at all about tk, but I think there is at least one indirect references in imported code that would be easy to get rid of).

This would be completely adequate for code that runs as a result communications with Houston or the hub, specifically this means: data from Houston, messages from the hub or commands sent to Apollo via the hub.

If that suffices, I think it's a great path to take. I'm willing to clean out the Tkinter references in RO.KeyVariable and RO.KeyDispatcher so you can use standard off-the-shelf code.

However, if you want to execute anything based on time (e.g. wait 3 seconds and then...) I think it is not so easy with the select loop. Craig can correct me here if I'm wrong, but I think your choices with select are:

- Threads. You really should avoid these if possible because it makes for very fragile code that can have extremely subtle bugs.

- Separate processes. These are perfectly reasonable if the amount of data communicated is limited (i.e. the process if fairly standalone). They get messy if tighter integration is needed.

Another option is to use the tk event loop for everything. Then socket-based communication and time-based communication are both trivial. If you go this route, you can use the whole RO package. RO.Comm.TkSocket for talking to Houston and the hub. RO.ScriptRunner to execute code like "wait for this and then do that but don't block anything else".

A potential third option is to use the Twisted Framework for communication. Frankly if I was writing TUI from scratch today I would do this. It has an excellent reputation and integrates with all the standard loops (select, tk, wx...). It may also give you time-based execution but I don't know whether that is so.

I strongly suggest trying to make this decision now, so that your early code can be built upon rather than restarting from scratch or, worse, spending way too much time trying to make a square peg fit into a round hole. This would be a good discussion to have while the three of us are all in the same place.

The Hardware (Cocoa)

Cocoa is a 1U rack-mount computer (speed, RAM, etc?) that runs ICC, and serves web functions such as streaming video from the STV, and Houston monitor data.

```
[root@cocoa ~]# uname -a
Linux cocoa.apo.nmsu.edu 2.6.14-1.1644_FC4 #1 Sun Nov 27 03:25:11 EST 2005
i686 i686 i386 GNU/Linux
```

STV Video

```
From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Thursday, January 26, 2006 15:50
To: 'Adam Orin'; Eric L. Michelsen; Tom Murphy
Subject: streaming video
```

WinTV	PCI card that does the hardware encoding of an input video stream the company that makes the card is Hauppauge and the model of the card is the WinTV-XXX (where XXX is either 150 or 250, I'm not sure if tom got the former or the latter).
IVTV	the linux driver for the WinTV card.
VLC	Video Lan Client which is a media player and streamer. We can use VLC to stream video from cocoa and to view it on another computer.

ok, after much ado, we are now able to stream video via HTTP at a user-defined bit rate. We also have a much better understanding of the WinTV+IVTV+VLC system than before.

1. VLC can change the driver settings for the WinTV card so when you adjust any WinTV parameter from within VLC the WinTV the card continues to encode using the previously set parameters even after you close VLC.

2. This suggests that it is cleaner to set the WinTV parameters using IVTVCTL from the command line _before_ launching VLC because you can get into many confusing states otherwise.

3. The software transcoding in VLC seems to produce a much nicer image at a lower bit rate (by a factor of >2) than an equivalent bit rate chosen on the WinTV card itself (Adam can explain more about that one).

4. In aorin's home directory on cocoa there is a file called streaming.txt in which I describe two functional setups. One in which there is no software transcoding and one in which VLC provides the software transcoding (at a CPU consumption rate of 15%).

There is a printout of this document on the cocoa keyboard.

```
From: Adam Orin [mailto:aorin@physics.ucsd.edu]
Sent: Thursday, January 26, 2006 18:39
To: emichels@physics.ucsd.edu; jbattat@cfa.harvard.edu
Subject: ivtv codec parameters
```

I think when we thought we were changing the hardware mpeg encoder settings with ivtv and vlc we were not. Evidence:

(1) Capture card's manual only states several discrete encoding modes, while we were able to encode at a lower bitrate.

(2) I think the capture card's hardware encoder is an MPEG-2 encoder, yet we were able to select MPEG-1 as the encoding method, and other more exotic codecs were also available.

(3) Many (cheap) capture cards do not have hardware encoders, yet the bitrate and encoding options in ivtvctl seem to be card independent.

Think the settings we thought were capture card settings are actually ivtv settings, not capture card settings. I'm just guessing here.

It all doesn't matter, since it's prettier and lower filesize to let VLC transcode the stream.

```
From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Thursday, January 26, 2006 20:03
To: Adam Orin
Cc: emichels@physics.ucsd.edu
Subject: Re: ivtv codec parameters
```

ok, but if that's the case then we should see the CPU being eaten up by some sort of IVTV software transcoding even if VLC is not running. perhaps you can look for this tomorrow.

but you're right that, in the end if the ivtv settings somehow do not consume cpu and the quality is acceptable we are in good shape. it would, of course, be nice to understand the system fully though...

7 Design and Implementation of housctl

SVN

housctl svn: svn://svn.apo.nmsu.edu/Apollo/Houston/trunk/

The first housctl checkin is Rev 497 from 2005-10-20 tar file.

497	first housctl checkin from 2005-10-20 tar file.
498	from 2007-09-29 tar file.
499	from 2007-10-06 tar file.
500	from 2007-10-19 tar file.
501	from 2007-10-28 tar file. (compressed tars from here on.)
502	from 2007-11-15 tar file.
503	from 2007-10-22 tar file.
504	from 2007-11-29 tar file.
505	from 2007-12-04 tar file.
506	from 2008-02-21 tar file.
507	from 2008-04-09 tar file.
508	from 2008-04-17 tar file.
509	from 2008-04-30 tar file.

Overview

Warning Mutual exclusion is critical! You must understand the mutex information below before changing housctl code. Deadlocks can cause housctl to hang, and stop environmental control.

housctl runs continuously. The main thread is responsible for all real-time work, and in RUN state, blocks only to wait for either fiducial or lunar photons. When not in RUN, it blocks for lots of things, because there are no time-critical functions outside RUN.

Common Maintenance Tasks

When testing new housctl code, it's nice to send email to apollo_core notifying team members of the test.

makefile

The makefile does not currently have a proper set of dependencies (for any of the targets). All the *.o files use the default dependency on only the *.c file, so none of the *.h file dependencies are recognized. This means that after changing a *.h file, you usually need to do:

```
make clean
make housctl (or 'make whatever')
```

Changing the Names of Log Parameters

mapping.h defines the names and numbers of the powered items, and other things such as CAMAC slots, etc.

Changing the names of log parameters requires coordination between housctl, the html activity monitor, and the summary plot scripts that make the web graphs, so that they all use/recognize the same names.

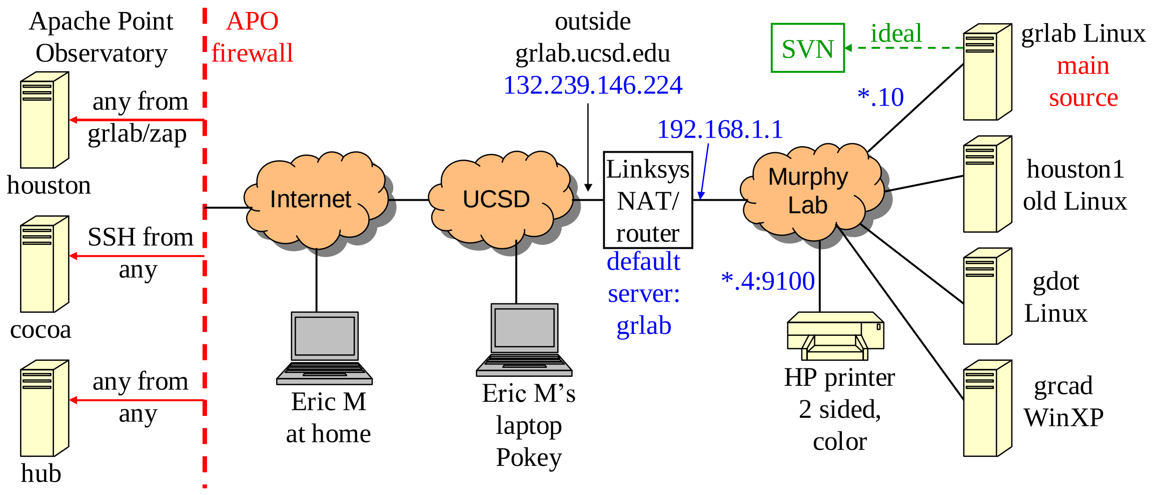
In housctl:

To change the names of powered items, edit mapping.h, macro POWERNAMES.

To change the names of RTDs, edit housctl.rtdcal. Then issue a 'readrtd' command to housctl. To change critical RTD numbers, edit the names, and update 'mapping.h' to match the T_xxx definitions.

Changing housctl does *not* change the names of the html monitor. Adam, what about the graphs??

Current Development Environment



Note that housctl source is now in SVN!

Here's what Eric M. does now (4/2008) to make a new version of housctl, but many other variants are possible:

1. Edit housctl source locally on Pokey.
2. SSH to grlab shell.
3. grlab> `timesync` to fix grlab's time of day clock
4. SSH file transfer source to grlab (with SSH client 3.2.9 build 283), using the file window.
5. grlab> `ssh houston1`
6. houston1> `timesync` to fix houston1's time of day clock.
7. houston1> `cd /home/apollo/src`
8. houston1> `make housctl`
9. Fix compile errors, repeat. Proceed when build is successful
10. Send mail to apollo_core informing of impending testing of housctl
11. houston1> `ssh houston` from grlab, SSH to houston:
12. houston> kill housctl (described elsewhere)
13. houston> `mv housctl bin/housctl.description`
14. houston1> `scp -p housctl houston:../apollo/`

15. From a new (2nd) grlab shell window: `grlab> ssh houston`
16. houston> start housctl (described elsewhere)

Privileges

Though we run housctl with root privilege, housctl does not, in general, *require* root privilege. However, it does attempt to improve its performance by locking all it's virtual memory into real memory, with `mlockall()`. It's not clear if this improves performance noticeably or not, but it doesn't hurt. mlockall( ) requires root privilege, but housctl continues to run even if the call fails.

Threads and Mutual Exclusion

In RUN state, dealing with slow serial (and even GPIB devices) is a problem because the main loop cannot block for them.

housctl blocks only to wait for either fiducial or lunar photons; all other blocking functions are relegated to child threads.

Some operations must block for unknown times exceeding ~10 ms. Houston Control real-time processing cannot stop for so long, so the main thread launches threads to handle blocking tasks. Most threads perform one function, and terminate, allowing for simple communication to the main thread. An exception is the ICC read thread, which continues forever. Threads that run and die include fetching GPS information, picomotor adjustments, and laser controls.

Mutual exclusion semaphores are needed for some shared resources, such as any file I/O, and the GPIB. We use "recursive" semaphores, which must be unlocked as many times as locked to be free. This is necessary for insuring continuity of multi-statement data/log-file lines. Of course, a child thread hogging a mutex can block the main thread. This is especially true of the I/O mutex.

Warning Taking only one mutex at a time insures no deadlock. Avoid locking two mutexes simultaneously. Because of mutex lock recursion, even if you think you're locking two mutexes in "proper" order, it's hard to insure that a higher level lock is not already in place. As of 6/1/2006, there is no place in housctl that locks two mutexes simultaneously.

When not IDLE, any operation that can block must be done by a child thread of the main thread. To insure proper synchronization, all data/log file I/O are synchronized with a mutex. ICC commands are read by the ICC thread, and written by the main thread. Many threads return results and busy status in a structure, which other threads (usually the main thread) can poll. This leads to potential problems with optimization, because the compiler may squeeze out a load of a volatile memory location.

housctl is *not* compiled with optimization, to avoid the usual problems of optimization and multi-threading.

We could probably work around this by declaring the status variables **volatile**, but this is tedious and error-prone, so (as of 10/2007) *we have not done this*. We should probably declare the things we know to be volatile, but that shouldn't give a false sense of security that everything that needs to be volatile is so declared.

Threads:

Main	owns data socket (TCP port 5321). Executes ICC commands
ICC read	owns ICC input (TCP port 5320). Passes ICC commands to Main thread.
GPS	locks GPIB resource

Picomotor	locks GPIB resource. This is a slow machine that requires special processing. See APOLLO's Picomotor library documentation elsewhere.
Temperatures	

Maybe we also need:

Laser	owns async I/O handle
STV camera	owns async I/O handle
TR Motor	owns async I/O to motor

No threads are needed for the Data Acquisition (DAQ), or CAMAC operations, because they don't block.

Mutual exclusion:

Our mutexes MUST be recursive, because high level functions often need to lock a resource, then call a low-level function which must lock the same resource.

Warning Stunningly, "man pthread_mutex_unlock" states "On ``error checking'' mutexes, pthread_mutex_unlock actually checks at run-time that the mutex is locked on entrance, and that it was locked by the same thread that is now calling pthread_mutex_unlock. If these conditions are not met, an error code is returned and the mutex remains unchanged. ``Fast'' and ``recursive'' mutexes perform no such checks, thus allowing a locked mutex to be unlocked by a thread other than its owner."

Thus, all code MUST unlock a mutex ONLY if it is sure it owns it! This is stunningly bad behavior of the library. Further, so far as I know, there is no workaround, i.e., no way for a thread to know if it does or does not own a mutex. The upside is that it is not hard to follow this, but very dangerous if you fail.

There are currently (3/2006) 3 mutexes:

mutexIO	for log/data/icc I/O
mutexGPIB	for GPS and Picomotor GPIB devices
mutexLaser	for laser control/display access

We probably need one for DAQ.

Properly written drivers would not require mutual exclusion between threads, but I cannot find any documentation on the drivers to confirm this. As a result, DAQ and GPIB use mutual exclusion semaphores, because DAQ is accessed concurrently for power control and temperature data. GPIB covers the Picomotor and GPS clock. It's too hard to schedule around these conflicts, so we use the native "recursive mutex" functions in the pthread library.

Note that we do not use the (awkward) "semaphore library," which is just layered on top of the native pthread library.

We use shared-memory variables for limited communication between threads, mostly to indicate thread completion. For this, we assume 32-bit writes are atomic, and in-order (between threads, at least). This is almost certainly valid on a single processor machine, due to the Linux process/thread context-switch time.

I'm not sure what would happen on a multi-processor machine.

Linux pthread Issues I found out the hard way

When starting a child thread, the Linux pthread library creates an additional thread. I guess that this is for thread control, but I don't know for sure. So, if you do a ps after creating a child thread, you see 3 threads (not 2): parent, system, and child. There is only one system thread no matter how many concurrent child threads run.

Also, see the nice() notes below in "Real Time Issues."

Real Time Issues

Priority Scheduling: `renice` requires root privilege. `renice` affects the dynamic scheduling of "interactive" processes, but does not give a constant ("static" in Linux-talk) higher priority. Testing has shown that '`renice` *`pid`* `-20`' , along with efficient coding of the temperature acquisition loop, eliminates most of the missed lunar returns.

If `renice` doesn't work, or we don't trust it, we *can* go to static scheduling. However, then you always have to have a high-priority shell available, in case the high-priority process dies in an infinite loop. When that happens, all normal processes stop, including all the normal shells. Without a high priority shell to kill the errant high-priority process, you must reboot the machine. The need for this high-priority "emergency-stop-shell" is somewhat problematic for our remote access. We may be able to do it through a serial port on houston, and the terminal server. I wonder if ssh or etc/passwd has an option to make some users high priority when they log in. Will ssh even work with a high priority cpu hog?

In sum, renice may be all it takes, or it could be much more involved. We have known all along that "winging it" may fail at some level of loading, and we have now reached that level. As of 8/2006, '`renice`' seems to work.

Note the nice( ) call from within housctl is no good, because all child threads inherit the high priority. However, using renice from the shell can change the main thread, but child threads still get launched with nice = 5.

All child threads get a nice( )=5 from the system (I don't do it), which is lower priority than the default of nice = 0. I don't know why this happens, but it seems to be OK, so I let it go.

From: swanson: I think this url at the Oreilly site does a bit better job than the man page in describing scheduling. http://www.oreilly.com/catalog/linuxkernel/chapter/ch10.html

Memory locking: Beside priorities, page faults (not seg faults; page faults are normal in virtual memory) can cause delays. Other processes, even lower priority ones, can cause page faults in housctl, so we may have to lock memory with mlockall(), and do one or two other tricks to insure the needed memory is really there.

mlockall( ) requires root privilege, but housctl continues to run, even if the call fails.

From 'man mlockall':

```
    If  MCL_FUTURE has been specified and the number of locked pages exceeds
    the upper limit  of  allowed  locked  pages, then  the  system  call  which
    caused the new mapping will fail with ENOMEM.  If these new pages have been
    mapped  by the growing stack, then the kernel will deny stack expansion and
    send a SIGSEGV.
```

Optimization: see *Multi-threading*, elsewhere, for a discussion of optimization.

Consistency Checks

In RUN state, housctl checks the FRC on each fiducial shot, and if we passed an expected lunar gate, moves on to the next lunar gate. housctl skips ahead as far as needed until the next lunar gate is in the future, so housctl recovers as best it can from any number of missed gates.

TCP Timeouts

A long-standing problem is that of TCP timeouts. The Linux system defaults are hours, which means a network problem can hang housctl for hours (i.e., "forever"). The digipots are on the terminal server (TS), connected with TCP. Eric M. doesn't know how to change these timeouts on a per-connection basis. As of 10/2007, some of the environmental controls are also fed by the TS, and can suffer a similar fate. We need to move these onto the parallel port.

Quadrant Diffuser Tracking

Ideally, once the quadrant diffuser phase is set, it should never need adjusting. However, for unknown reasons (see outstanding issues section), the diffuser phase sometimes drifts off. The diffuser phase is measured in TR motor steps of 1000 per revolution (divided down from maximum of 4000 steps/rev). Operators should set the 'diffuser_target' parameter to the desired phase. On entry to RUN, housctl sets the diffuser phase to 'diffuser_target'. During shooting, housctl checks the phase each second, and corrects it. Each time housctl checks the phase, it sets 'diffuser_phase', and sends it to the ICC:

```
    0 i diffuser_phase=976
```

Seems like we should log these with 'par' records??

Eric M has seen the TR motor go into persistent oscillations when it was supposed to be stopped, which supports the theory that diffuser phase drifts due to overshoot and settling in the TR motor. Note that our diffuser divide chain counts TR encoder pulses, without regard to what direction the TR motor is moving. It's only supposed to move forward, but if it oscillates, it will step the diffuser forward without any net gain in TR position.

Environmental Control

The Intermediate Level Enclosure (ILE) and Utah box environments are temperature controlled.

ILE Temperature Control

The ILE temperature is not critical; the goals are to keep it from overheating, and avoid freezing. Housctl controls an exhaust fan than blows out ILE air, and takes in ambient intermediate level air. If either ILE or cabinet temperature is above t_ile_hi, housctl turns the fan on. If both are below t_ile_low, the fan is turned off. This provides some hysteresis, to avoid excessive cycling of the exhaust fan. As of 3/16/2006, the defaults are 20 and 25 C.

ILE temperature >1 C above set point, or less than t_ile_alarm_low, generates an alarm.

In additoin, the ILE has a self-contained thermostatically controlled heater (what wattage??), currently (3/16/2006) set for ~10 C.

Utah Box Temperature Control

There is a small temperature range in which the Utah box is stable, and does not require active temperature control. housctl then issues no control commands.

Utah box temperature is more critical than the ILE, because the laser and lots of electronics are in the Utah box. Utah box temperature is controlled by 3 parameters:

t_utah_center	desired set point, typically 20.25 C
t_utah_push	how far heat/cool will go past (above/below) center point, typically -0.25 C
t_utah_limit	how far below/above center point triggers heat/cool
Warning	Before the Norens were turned on during heating, if t_utah_limit < t_utah_push + ~.75, heat and cool will sometimes compete. We don't know if that's true now.

Cooling: When the temperature exceeds t_utah_center + t_utah_limit, housctl cools it to t_utah_center - t_utah_push. With typical values of 20.25 and -0.25, housctl cools down to 20.5 C.

There are 3 components to Utah box temperature control: passive cooling, M33 chiller cooling, and heating.

Passive cooling: When the dome_air is more than ~7 C cooler than t_utah_center – t_utah_limit, and the CAMAC is off, housctl turns on the Noren heat exchangers in the Utah box, and cycles the propylene-glycol through a small heat exchanger in the intermediate level. This provides adequate cooling. If the dome_air warms to less than ~6 C cooler than t_utah_center – t_utah_limit, housctl switches to active refrigeration with the M33 chiller. Again, hysteresis avoids excessive switching between passive and active cooling.

M33 chiller cooling: When the intermediate level air is too warm for passive cooling of Utah, housctl uses the M33 chiller. housctl turns off the passive cooling pump, and turns on the M33, which takes over pumping and chilling the coolant.

Utah heating: When the Utah box is too cold, housctl turns on the internal heater. This heater has its own thermostat set to ~32 C, which we use as an over-temperature safety stop. housctl provides the actual heating regulation by turning off the heater at a lower temperature. When the temperature goes below t_utah_center – t_utah_limit, housctl heats it to t_utah_center – t_utah_push. With typical values of 20.25 and -0.25, housctl heats up to 20.5 C. As of 10/2007, housctl turns on the Noren fans when heating. This makes the heating cycle much longer, as the heat is uniform throughout Utah, which takes longer during heating, and stays warm longer when idling.

Utah temperatures >1 C outside set points generates an alarm.

Background:

```
    From: Adam Orin [mailto:aorin@physics.ucsd.edu]
    Sent: Wednesday, November 23, 2005 14:16
    To: Tom Murphy
    Cc: Eric L. Michelsen
    Subject: Re: proposal for utah temp control algorithm
```

It looks to me like right now there is about a 1.5 C window in the allowed Utah temp. If you decrease the window, the heater is going to come on a lot more often at night (the Utah RTD heats and cools rapidly). Right now when it's cold, the heater comes on once every 15-30 mins, for about 4 mins at a time.

```
    From: Tom Murphy
    Sent: Sunday, November 06, 2005 20:58
    To: Eric L. Michelsen
    Subject: RE: Why is Utah so hot?
```

I think a few things are going on here:

1) the Norens are more effective than the external heat exchanger, so the fluid temperature is closer to the Utah than the external.

2) the Noren airflow is not yet baffled fully so that some of the output can get right back to the intake, not fully circulating in the box

3) the hot air collects in the top of Utah, so the Noren temperature readings are often lower than the temperature in the top of the box.

The net effect is that the passive cooling can't handle a small delta-T.

Dealing With RTD Failures

housctl specifically anticipated failures of one or more RTDs, and the power control commands allow for overriding automated power controls which are harmful due to invalid RTD data. See the “Handy housctl Commands” section for more information.

On the Prospect of Smoother Utah Temperature Control

```
From: Tom Murphy
Sent: Saturday, July 15, 2006 12:30
To: Michelsen Eric
Cc: Swanson Erik; Battat James; Orin Adam; Murphy Tom
Subject: temperature control scheme
```

It's worth taking a look at the thermal data plots from last night. The sequence went like:

Run laser flow for an hour with no flashing to see warmup rate in water: conclude that heat addition does not come from flashlamps because DI at laser (called Laser Air, AOML) shows no differential on either side of the flashlamps.

After short pause, flash for 30 minutes to warm up, using M33 setpoint of 10

Start regular schedule of flash for 10 min, off for 5. for next 30 minutes, stay at M33=10.

```
Go to M33=11 for 30 min
Go to M33=10 for 30 min
Go to M33=9 for 30 min
Go to M33=8 for 30 min
```

These periods are well-defined by looking at the PG temps--you can see the steps clearly.

Utah temperature stabilized quite well in initial M33=10 stage. Went up in M33=11, back down (slightly) at M33=10. Think there was longer term upward trend as large therrmal masses contribute to long time constant. M33=9 brought down further. M33=8 drove down to 20.50 lower level and shut off cooling, after which there is a bounce (the thing we're trying to eliminate).

So the lesson is: if we want to control temperature smoothly (without introducing spikes in the all-important TDC), we need to keep the fans on and the M33 on, but vary the setpoint for control. Last night's exercise was a proof of concept and also a decent caracterization of response.

It looks like initially, a higher M33 setting does the job, but we will want to increment downward as time goes on over long runs as the thermal mass catches up. It doesn't have to be too smart. A possible rule set is:

- If CAMAC is on, don't shut off M33 or fans if the temperature falls below lower limit: bump up M33 setpoint in this case
- If upper limit is hit, turn on M33 and fans if not on yet, with M33=10. If already on, bump M33 temp down a notch
- If above upper limit, and reduction of M33 by a notch is not effective after some interval (10-15 minutes?) then down another notch

The last is the trickiest. We do, after all have to deal with the non-instantaneous response in some smart-ish way. We may also want to set limits on the allowed M33 setpoints (e.g., 7 to 13) just so we don't run away from a sensible position. If Utah RTD fails right now, it could drive the system batty if not for this protection.

Laser Coolant Circulation

Houston is now (11/3/2005) running a version with a preliminary flow-sustaining function. It works only in IDLE state (for reasons given below), thus:

If the dome_air temp < circulate_temp (default 2) deg C, housctl circulates the laser head loop, to avoid freezing. Set ‘circulate_temp’ high (say, 40 deg) to make it circulate all the time.

If, when circulating, the flow < 0.3 for 2 flow measurements and the laser rack is off, housctl declares an alarm, turns on the laser rack (and hence pump), and turns off the passive pump.

If the flow is good for 2 measurements, housctl turn on the passive pump, and turns off the laser rack.

This function includes at least 3 "firsts" for housctl:

First #1: There is a conflict over laser rack power. What to do if the operator powers on the rack during an alarm? When the alarm clears, currently housctl will turn off the rack, likely surprising the operator. Also, the operator can command the laser off at any time, temporarily defeating circulation, though housctl will resume it after 2 low measurements. Fixing this requires separate "requests" for laser power, and an internal priority scheme. During non-IDLE state, the laser is expected to be on, so the "circulate_laser" function is disabled, again to avoid conflicts.

First #2: "Low laser-head flow" is the first persistent alarm, and it is self-clearing.

First #3: This is housctl's first use of stored temperature for internal operation. Until now, temperatures were reported, and immediately acted on for environmental control, but not stored for future use. This means we better keep the dome_air RTD functioning properly (or manually [and riskily] enable "circulate_laser"), or we risk freezing.

Laser Oscillator Voltage

housctl controls the laser cavity flashlamp voltage through a digipot (see James B's http://www.cfa.harvard.edu/~jbattat/apollo/docs/oscillatorVoltage/ for calibration info).

The digipots have internal non-volatile RAM, and so housctl does not include their settings in the "cums" file.

housctl has hard-coded the volts/ohm calibration to implement voltage control commands. Note that the PR (programmable resistor aka digipot) library converts ohms to DAC steps. We derive the volts/ohm calibration from James' plots as follows:

$$V_- = 2.49V\left(\frac{R_{23}+3k+2.4k}{R_D+10k+3k+2.4k}\right) \qquad \textit{where} \quad R_D = \text{digipot resistance in } \Omega$$

$$R_{23} = 7\text{k}\Omega \text{ for our typical threshold setting}$$

$$\Delta V_{dial} = \frac{1000\left(\Delta V_-\right)}{1.337934}$$

Evaluating V_- at R_D = 100 and 200 ohms, we find ΔV_{dial} = 0.09544 V/Ω.

Weather Data

housctl provides 3 settable parameters for recording weather data, so all TUIs can be kept up to date.

```
    set airtemp=degc pressure=mbar humidity=percent
```

housctl does not use these for anything, but they are recorded in the log/data files for data analysis. Of course, you can send these as separate 'set' commands if you want. housctl has no hard limit, but expects these to be updated ~once/minute, or less.

From Russell: Weather as reported by the TCC:
```
    sho weath
    0 4 I AirTemp=   13.30; SecTrussTemp=   12.17
    0 4 I PrimF_BFTemp=   10.54,    0.06; SecF_BFTemp=   10.00,    0.00
    0 4 I Pressure=   73097.0; Humidity=  0.22; TLapse=  6.50
    0 4 I WindSpeed=     6.3; WindDir=   297.0
    0 4 I TimeStamp=  4638110846.58
    0 4 : Cmd="sho weath"
```

See <http://www.apo.nmsu.edu/Telescopes/TCC/MessageKeywords.html> for the units (which are all metric, I believe).

Automatic Fetching in Houston

Houston can be modified to fetch weather data automatically:
```
    From: Eric L. Michelsen
    Sent: Sunday, March 11, 2007 19:28
    To: Fritz Stauffer (fstauffer@apo.nmsu.edu)
    Cc: 'Tom Murphy'; 'jbattat@cfa.harvard.edu';
    'hoyle@npl.washington.edu'
```

```
Subject: Automated barometric pressure
```

Hi, Fritz. In Apollo, we are running at the limit of accuracy on barometric pressure, and could really benefit from having more frequent barometer readings in our data file. Is there a simple way for a C program to fetch barometer readings whenever it wants? Ideally, some simple UDP/TCP session? What I hope to avoid is complex authentication. Or is there any documentation or other people to which you could direct me?

```
From: Fritz Stauffer
Sent: Sunday, March 11, 2007 19:49
To: Eric L. Michelsen
Cc: 'Tom Murphy'; jbattat@cfa.harvard.edu; hoyle@npl.washington.edu
Subject: Re: Automated barometric pressure
```

Eric, Here's a python script which does what you want. If it is run at APO, then there is no authentication. If it run outside of APO, then I need a fixed IP to open up the firewall.

I checked, and currently the pressure is read every minute. I can decrease the interval, but, I don't know offhand what is the time needed to make a pressure measurement. Let me know what you need.

```
#!/usr/bin/env python
'''
Example of a custom script to process weather server messages.

Weather server returns a string like this:

Received: 0 -3 i timeStamp=1143747047 pressure=21.372 tempout=4.67
tempin=14 humidout=29 windd=237.8 winds=13.6 gusts=27.8 gustd=254.1 temp=6
dpTemp=44.1 dewPoint=14.0 dpErr=P dusta=16363 dustb=1364 dustc=4569
dustd=142 airtemp=6.7 dewpoint=-10.0 humidity=29.1 dperr=P structtemp=4.67
sectemp=6.34 end

Clever fiddling lets one exec() the string and bring the variables into
the program name space.
'''
from socket import *    # import *, but we'll avoid name conflict

sock = socket(AF_INET, SOCK_DGRAM)
messout = "all"
sock.sendto(messout, ('weather.apo.nmsu.edu', 6251))
messin, server = sock.recvfrom(512)
sock.close()

# print received string
#print messin

# strip off beginning.
# replace = with =', and replace space with space'
# for example dewPoint=14.0 becomes dewPoint='14.0'
start = messin.find('timeStamp')
stop = messin.find('end')
stuff = messin[start:stop].replace ("=","='").replace (" ","'; ")

# exec - causes the pieces to become global variables
exec(stuff)

print 'pressure=%s, airtemp=%s, humidity=%s' % (pressure,airtemp,humidity)
```

Here’s a tutorial on sockets with Python examples: http://gnosis.cx/publish/programming/sockets2.html. It has this note:

> In Python, the socket object keeps track of the temporary socket number over which the message actually passes. We will see later that in C you will need to use this number from a variable returned by `sendto()`

His knowledge of C sockets is limited, though, because this claim is not true:

> The [Python] client gets three arguments: server address, string to echo, and the port. Being that Python wraps up more in its standard modules than do roughly equivalent C libraries, you can specify a named address just as well as an IP address. In C you would need to perform a lookup yourself, perhaps first testing whether the argument looked like a dotted quad or a domain name.

In fact, the C-library routine `gethostbyname()`handles both DNS names and IP addresses transparently.

8 Houston Device Library Software

The philosophy is to name the devices and functions somewhat generically, so that if we replace equipment, we can hopefully just replace the library, while the master program uses the same high level commands (e.g., get_time()).

'as_util' is a superset combined-serial/telnet library, where the caller need not know whether the interface is hardware serial or telnet.

The libraries are mostly reentrant, with the exception of static error strings, and the laser library. This is too hard to fix right now.

Library Name	**Dependencies**	**Blocking?**	**Comments**
`ser_util`		read( ) blocks	Generic serial package
`tcp_util`		read( ) blocks	Simple way to open TCP connections. Tightly coupled to serial
`as_util`	`tcp_util, ser_util`	read( ) blocks	General async package supporting both TCP and serial ports, transparent to client
`ts_util`	`tcp_util`	read( ) blocks	Terminal server set/clear control leads.
`camac`		Non-blocking	
`gpib`		?	
`daq`		Non-blocking	National Instruments DAQ card
`tdc`	`camac`	Non-blocking	
`acm`	`camac`	Non-blocking	
`gps_clock`	`gpib`	?	XL-DC Model 151-602-949
`optics`	`gpib`	move blocks	New Focus optical actuators
`tr_motor`	`ser_util`	read( ) blocks	
`chiller`	`ts_util`	only on comm failure	Thermo NESLAB chillers
`laser`	`ser_util`	button pushes	Continuum laser
`stv_lib`	`ser_util`	?	STV video camera
`pwr_meter`	`ser_util`	only on comm failure	laser power meter
`wti_util`	`tcp_util`	blocks	WTI IPS IP Power Switch
`pr_util`	`as_util (tcp_util, ser_util)`	blocks	Programmable resistor

Serial Library

Provides open, close, read, write, poll, and EIA-232 lead get/set functions.

TCP Library

Duplicates most functions of serial library.

Asynchronous Library

Provides all the functions of the TCP and serial libraries, transparently between hardware serial and TCP session. If the device name starts with a "/", as_open( ) assumes it is a hardware serial port; otherwise, as_open( ) opens it as a TCP connection. Other as_xxx( ) functions work transparently (poll, flush, etc.).

Soon it will have diagnostic features such as time-stamped logging of bidirectional traffic to a file.

Terminal Server Library

Provides EIA-232 lead get/set functions. There seems to be a high rate of failure of terminal server lead controls (~1 in 10). Clients should retry terminal server lead control commands, if they fail the first time.

Power Control: houspower.c

Terminal Server Power Not Used Anymore

In the past, houston used the terminal server EIA-232 control leads for some power outlet controls. There seems to be a high rate of failure of terminal server power controls (~1 in 10). housctl now retries terminal server power commands, if they fail the first time. The cause of the failure is some kind of TCP failure or delay of several seconds. One retry is probably not enough, because there may be still a 1 in 100 chance of failure. Eric M has seen it fail both times, to make an actual failure.

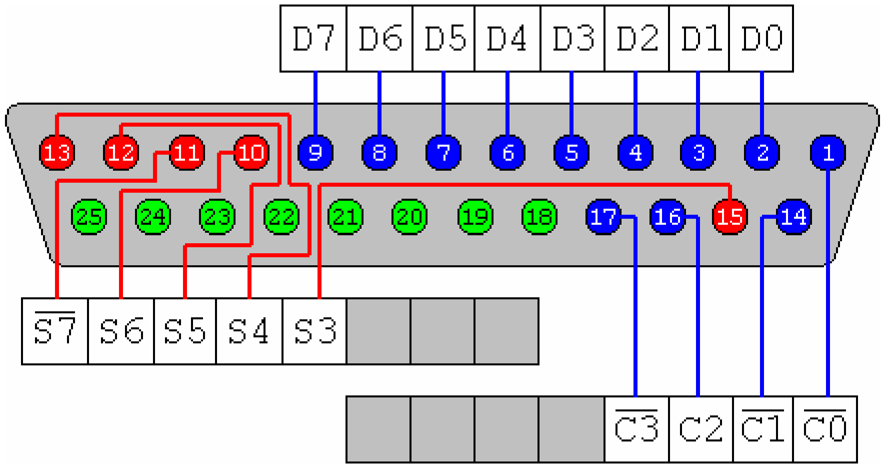
Parallel Port

We have switched from the TS for critical power control (e.g., environmental control), to the parallel port.

UCSD made a power-distribution box specially for the parallel port, with a D25 connector wired for the parallel port. The cable from the parallel port to the new power distribution box is a straight-thru DB25 cable, currently (10/2007) a 15 foot commercial cable.

Eric M prefers to use the pseudo-file approach, because it is clean and reliable, requiring neither root privilege, nor special compilation options. See people.redhat.com/twaugh/parport/html/x623.html, or web search for "parport0", "lp0", or "ppdev". Eric M has verified that we can open '/dev/parport0', and use the PPCLAIM, PPWDATA, and PPRDATA ioctl( ) to take exclusive control, write to data lines, and read back data lines. Reading back the data is critical for a restarted housctl to learn the current system state, in this case, the power states of all parallel port devices.

The parallel port is a female D25 connector, with 8 data lines [www.geocities.com/nozomsite/parallel.htm]:



D25 Female

Pin No (DB25)	Signal name	Direction	Register - bit	Inverted
1	nStrobe/	Out	Control-0	Yes
2	Data0	In/Out	Data-0	No
3	Data1	In/Out	Data-1	No
4	Data2	In/Out	Data-2	No
5	Data3	In/Out	Data-3	No
6	Data4	In/Out	Data-4	No
7	Data5	In/Out	Data-5	No
8	Data6	In/Out	Data-6	No
9	Data7	In/Out	Data-7	No
10	nAck	In	Status-6	No
11	Busy/	In	Status-7	Yes
12	Paper-Out	In	Status-5	No
13	Select	In	Status-4	No
14	Linefeed/	Out	Control-1	Yes
15	nError	In	Status-3	No
16	nInitialize	Out	Control-2	No
17	nSelect-Printer/	Out	Control-3	Yes
18-25	Ground	-	-	-

There is also a crude, low-level way to control the port, which requires both root privileges and compiling with optimization [as6edriver.sourceforge.net/Parallel-Port-Programming-HOWTO/accessing.html]. Eric M prefers to avoid these restrictive methods. The optimization is required to replace the "library" call with in-line code, because there actually is no library function. It will show up as unresolved in the linker if compiled without optimization. Recall that housctl is *not* compiled with optimization, to avoid the usual problems of optimization and multi-threading (see elsewhere in this doc). We could, however, compile only the parallel port utility library with optimization, but not other modules. Note that ioperm( ) is inherited by exec( ) processes, even if they are not root. fork( ) does *not* pass on the ioperm( ). See ioperm(2) in man pages.

CAMAC Library

The CAMAC hardware and driver are barely documented, so some things are vague. We found out the hard way that a machine (or a process) can open multiple handles to the CAMAC, but it is senseless to do so, because closing the first handle invalidates all the others. This is probably because closing the CAMAC handle resets the hardware in some way.

We are at the mercy of the crummy CAMAC driver, which does not allow timeouts on waiting for CAMAC interrupts. If the CAMAC does not interrupt, housctl will hang waiting for it, and there is nothing we can do about it. A better CAMAC driver would allow housctl to set a timeout on the wait for interrupts (say, 40 ms), which would allow housctl to keep functioning even on CAMAC failure, or other hardware or software errors that lead to no interrupt. Perhaps a newer version of the driver has this feature, which is usually standard on any driver.

TR Motor Library

The TR motor is a large, intelligent motor, driven by an asynchronous serial EIA-232 interface. The motor takes cryptic commands, and provides cryptic responses.

The TR motor library uses a serial port file handle as a its handle.

Motor positions use the 0-3999 encoder, for 4000 steps/revolution.

Note that the diffuser motor is controlled from the ACM library. Resetting the CAMAC or ACM will disable the diffuser.

ACM Library

Note that the diffuser motor is controlled from the ACM library. Resetting the CAMAC or ACM will disable the diffuser.

TDC Library

The upper and lower thresholds do not work, and we don't know why. The TDC is supposed to count as "no measurement" any count less than (or <=??) the lower threshold, or greater than (>= ??) the upper threshold. It should not set such channels in the hit mask, and should not return values from them in sparse reads. Currently, it's as if the thresholds aren't enabled, and we get measurements from all channels that had 'start' pulses.

Prepulses (where the APD fires before the TDC gate opens) cause the maximum time (4095 bins) on the TDC. This results in a large number of bogus "hits", and this even bogs down TUI. We should change housctl (as of 3/3/2008) to filter these outliers in software, and not write them to the data file.

Bolometer (pwr_meter) Library

The bolometer is plugged into the UPS power strip directly, because it requires a manual button push after power up, so we don't want it to lose power.

Chiller Library

The female DB9 plugs directly into a PC (DTE = Data Terminal Equipment), with a straight through cable. (In other words, the female DB9 is a DCE [Data Communications Equipment], no crossovers, null-modem, etc.). Functions include:

- turn on/off
- get operating status
- get temperature, limits, parameters
- set temperature, limits, parameters

GPS Clock Library

Hardware:	XL-DC Model 151-602-949, with low-phase-noise 10 MHz reference option, and GPIB interface. Operating temperature: -40 to +70 C. Storage: -55 to +85 C.

Page 1-11 of the manual describes the pinout. It says explicitly that it is a DTE, with a male DB9, so that pin 3 is data out of the XL-DC and pin 2 is data in. This means we need a null modem between the clock and PC. The XL-DC is on /dev/ttyM3 (3/31/2006). It is a 7E1 9600 interface (changeable with keypad function 04, but we don't know if it survives power cycles, so we leave it at the default).

Closed up in the XL-DC box, we expect temperature to be 40 C or so. The clock is on the WTI IP switch, plug 4. This way if the clock overheats, we can shut it down.

Programmable Resistor Library

The new resister boards use Analog Devices AD5262 (branded AD5262B20), with no NVRAM (http://www.analog.com/UploadedFiles/Data_Sheets/618372110AD5260_2_0.pdf). The NVRAM is now in the controller, but that is transparent to the serial interface. Microchip's spec sheet says the EEPROM is rated for 100,000 cycles. These resistors are true 3-terminol potentiometers, but we use them as 2-terminal variable resistors.

Library resistor 0 <-> RES1; Library resistor 1 <-> RES2.

```
NDAC steps=256
min         ~60 ohms (DAC=0)
max          20 kohm (DAC=255)
```

James Battat designed an RDAC board, with this interface (as written by the man who wrote the microprocessor code, with updated instructions by ELM):

```
Date: Wed, 11 Jan 2006 13:36:08 -0500
From: Jim MacArthur <macarthur@physics.harvard.edu>
To: James Battat <jbattat@cfa.harvard.edu>
Subject: RemoteRes protocol
```

The terminal settings are 9600, 8 bits, 1 stop, no parity.

The basic command format is a single alphabetic character, followed by an optional (decimal) numeric field, followed by a delimiter. The Remoteres processor echoes all characters. Lower-case characters control RES1, and upper-case characters control RES2 (if any).

```
Supported commands:
D = Decrement current RDAC value
I = Increment current RDAC value
N = No operation, enter low-power mode
R = Retrieve the contents of the non-volatile memory into the RDAC
S = Store the RDAC setting into the non-volatile memory
Q = Query the RDAC setting
W = Write the following decimal field to the RDAC

Recognized delimiters are:   ,-./:;
```

All characters other than '0123456789DdIiNnRrSsQqWw,-./:;' are ignored. When a processor receives a delimiter, it performs the command specified by the last valid alphabetical character. Thus:

```
'kldfj2 234 wer.'     is interpreted by the processor as     'r.'
```

which retrieves the contents of the non-volatile memory into RDAC 1. The following commands are identical:

```
W100.
W 100;
W   0100,
```

They all write the decimal value "100" into RDAC 2. The "W" command is the only one that requires a following numerical field. All others just need an alpha character and a delimiter.

"Q" is the only command that returns information (before echoing the delimiter): a 4-digit numerical field with the setting of the selected RDAC. Thus, typing: 'Q.' echoes as 'Q0123.' assuming the value of RDAC 2 is 123.

I've attached the assembly code: digipot1.asm

Hardware

The female DB9 plugs directly into a PC (DTE = Data Terminal Equipment), with a straight through cable. (In other words, the female DB9 is a DCE [Data Communications Equipment], no crossovers, null-modem, etc.).

Board dimensions are 2.4" x 3.4". A board layout is viewable at:
http://www.cfa.harvard.edu/~jbattat/apollo/docs/remoteControlOfLaserPower/boardView.PNG

```
From: James Battat
Sent: Wednesday, March 15, 2006 15:14
To: apollo_core@u.washington.edu
Subject: [Apollo_core] DIGIPOT board
```

I've done the schematic and layout for the DIGIPOT board which should work in all three desired locations (the capacitor bank and the power units for the oscillator and amplifier). It is based around a digital-potentiometer chip (DIGIPOT) that can take up to 15V across the pot. In our case we will have no more than 12V across the pot.

See an image of the board and PDFs of the circuit schematic (there are 2 pages) at:
`http://www.cfa.harvard.edu/~jbattat/apollo/docs/digipot/`

```
From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Thursday, May 11, 2006 14:34
To: Eric L. Michelsen
Subject: pot stuff

1. The capacitor bank manual potentiometer is 100k full scale.
2. The PU manual pots are 10k each.
3. Low power mode on the capacitor bank should give ~50 mW power.

I've put the pot mapping that Tom made up at the top of:
http://www.cfa.harvard.edu/~jbattat/apollo/docs/remoteControlOfLaserPower/

From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Thursday, May 11, 2006 18:46
To: Eric L. Michelsen
Subject: RE: pot stuff
```

I did a visual check of the 10k pots in the PU610c and PU620c. They both have the same labeling.

If I turn the pot fully CCW then the reading is 1860 Volts

If I turn the pot fully CW then the reading is 620 Volts

Because the markings on the pot end at 740 Volts, the number 620 is an estimate...

DAQ Library

Our hardware is National Instruments NI 6031-E, 100 kS/s, 16-Bit, 64-Analog-Input Multifunction DAQ: http://sine.ni.com/nips/cds/view/p/lang/en/nid/1055. The 6031 has 64 channels of single ended, 16 bit analog inputs

From http://www.comedi.org/doc/ : Comedi is a free software project to interface *digital acquisition* (DAQ) cards. It is the combination of three complementary software items: (i) a generic, device-independent API, (ii) a collection of Linux kernel modules that implement this API for a wide range of cards, and (iii) a Linux user space library with a developer-oriented programming interface to configure and use the cards.

RTD Library

As of 10/2007, RTDs have their own library, rtd_util, that calls on the DAQ library. The RTDs are

RTD Sensors Rdf Corporation 29230-T10-A-24 Important specs:

Ours are 1000 Ω at 0 C, varying 3.85 Ω/C.

Temperature Range	–200°C to 260°C (–320°F to 500°F)
Time Constant	<0.2 Seconds on metal surfaces
Self-Heating	>15 mW/C mounted
Long Term Stability	Better than 0.05°C (0.2Ω) per 5 years, above –50°C
Maximum Current	5 mA for limited self heating
Recommended Current	1 mA maximum

The RTDs are driven by analog interface boxes made by James Battat. The boxes have offset & gain controls, but we calibrate the RTDs in software. Ideally, the calibration would include the wire resistance, but doesn't yet (as of 12/14/2006). We have two RTD boxes, each with its own calibration. As of 1/2007, we are using Box 2.

The RTD channel map (rtd#, NI DAQ# and name) on the web at: http://www.cfa.harvard.edu/~jbattat/apollo/docs/rtd/

```
From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Sunday, January 15, 2006 16:48
To: Eric L. Michelsen; 'Adam Orin'
Cc: Tom Murphy
Subject: rtd calibration (fwd)
```

Tom and I calibrated all 24 channels of the RTDs. Instead of multiplying the voltage by 10, please do the calibration described below. In the attached file the columns are:

```
 Col 1         = Channel Number
 Col 2 (aka M1) = 1k resistance
 Col 3 (aka M2) = 1.2k resistance
```

If V is the voltage read by the DAC then to properly calculate the temperature, T (deg C), from the voltage reading you should do the following:

```
   T = 51.948*(V-M1)/(M2-M1)   (in deg C)
```

Box 1

```
 0  0.051 5.248
 1  0.046 5.240
 2  0.041 5.239
 3  0.047 5.242
 4  0.047 5.242
 5  0.039 5.235
 6  0.056 5.254
 7  0.034 5.230
 8  0.044 5.241
 9  0.042 5.240
10  0.034 5.230
11  0.041 5.239

12  0.007 5.205
13 -0.014 5.180
14  0.000 5.193
15 -0.042 5.148
16 -0.006 5.199
17 -0.036 5.160
18 -0.036 5.160
19 -0.049 5.142
```

```
20 -0.035 5.158
21 -0.036 5.160
22  0.134 5.322
23 -0.025 5.171
```

Box 2

```
From: Tom Murphy [mailto:tmurphy@physics.ucsd.edu]
Sent: Wednesday, January 24, 2007 9:26
To: James Battat
Cc: Michelsen Eric
Subject: Re: Calibrated temperatures
```

The last calibration was July 13, 2006, on the box (2) currently in place. The calibration can be found on houston in /home/apollo/util. There are two identical files there: one a copy of the other with a useful date-stamp filename (manually accomplished).

```
# Ch  1k         1.2k  Box 2 7/13/2006

0   0.0513465  5.2402535
1   0.03959725 5.23323425
2   0.05935775 5.2473485
3   0.05905225 5.2550545
4   0.0484475  5.2390325
5   0.050889   5.24094
6   0.06218025 5.2542915
7   0.04448    5.2375065
8   0.0420385  5.2298775
9   0.06088325 5.2584115
10  0.0518805  5.23674375
11  0.05592425 5.24895075
12  0.240482   5.43839175
13  0.07286175 5.25993725
14  0.00404375 5.19913025
15 -0.0052645  5.18478675
16  0.01785325 5.19363675
17 -0.01441975 5.169604

18 -0.017319   5.18089575
19 -0.02235425 5.16334775
20 -0.02739    5.1673915
21  0.00701925 5.193103
22 -0.01586925 5.17288475
23 -0.00846875 5.1783015
```

Old Box 2

```
From: Tom Murphy
Sent: Wednesday, January 18, 2006 11:06
To: Eric Michelsen
Cc: James Battat; Adam Orin
Subject: RTD box2 calibration

0       0.039   5.235
1       0.045   5.245
2       0.046   5.231
3       0.037   5.239
4       0.043   5.230
5       0.044   5.239
6       0.045   5.231
7       0.042   5.241
8       0.046   5.240
9       0.047   5.257
10      0.045   5.239
11      0.049   2.226

12      -0.026  5.163
13      -0.011  5.179
14      -0.029  5.163
15      -0.026  5.167
```

```
16      -0.026  5.164
17      -0.026  5.166
18      -0.035  5.157
19      -0.028  5.163
20      -0.027  5.164
21      -0.020  5.170
22      -0.028  5.162
23      -0.024  5.165

From: James Battat [mailto:jbattat@cfa.harvard.edu]
Sent: Tuesday, April 04, 2006 16:46
To: 'Adam Orin'; Eric L. Michelsen; Tom Murphy
Subject: Reordered RTDs in TDC
Adam and I re-ordered the RTDs in the TDC.  Now the mapping is:

RTD_Chan  Name   Description
 7        TDC1   Intake
 8        TDC2   C16
 9        TDC3   C9
10        TDC4   C2
11        TDC5   Exhaust
In the TDC, C2 is near the top of the board, C9 near the middle and C16
near the bottom.
```

Laser Library

Safety

For safety, `las_setshutter()` verifies that the shutter state becomes what is asked, or returns an error. If the state is unknown at first, `las_setshutter()` toggles the shutter, which should cause the CU to send its state, and then toggles again, if needed.

Capabilities

1. activate a program,
2. stop/start
3. open/close shutter
4. arbitrary button pushes (2nd harmonic CW/CCW, etc.)
5. key-switch off/on

Future Enhancements

Maybe? Speed up operations by reading the display, and moving forward when display confirms that the button push has been accepted, instead of always waiting for a blind timeout.

Functions

```
extern      int     las_open(       // returns ts_port fd
  const     char    *device);       // serial device name, e.g. "/dev/ttyS0"

extern      int     las_close(int fd);      // close fd: returns close() status
extern      int     wait_for_display(// ret <0 on error, 0 = no change, 1 = change
    int     fd,             // serial port to Control Unit
    int     ms);            // ms to wait

extern      int     las_poll_display(// ret <0 on error, 0 = no change, 1 = change
    int     fd);            // serial port to Control Unit

extern      int     las_push_button(        // returns write() status
    int     fd,             // fd of open serial port
    char    ch);            // character to send
extern      void    las_keepalive(int fd); // request to send keep alive (if enabled)
extern     int     las_start(int fd);     // start firing laser
extern     int     las_stop(int fd);      // Stop firing the laser

// Stop laser, activate a given program
extern     int     las_activate(  // return <0 on error
    int    fd,            // laser fd
    int    program);      // program # to activate
extern     int     las_get_display(      // return contents of virtual keybox display
    char   *buf);         // Filled with a fixed DISPLAY_SIZE bytes.
extern     int     las_setkeyswitch(     // returns < 0 on error
    int    fd,            // laser fd
    int    offon);        // 0/1 = off/on
extern     int     las_setshutter(       // returns < 0 on error
    int    fd,            // laser fd
    int    offon);        // 0/1 = off/on
```

Timing

The laser is very sensitive to button-push timing, and we've had to learn the timing requirements through many months of trial and error. The current (6/2/2006) delays are

Query response	We use a 50 ms poll cycle; must happen within <100 ms. 100 ms poll cycle missed about 1 in 10 startups
button-push	800 ms
activate	800 + 800 ms
reset	800 + 2000 ms
key-cycle	We recommend applications leave the key-switch off for at least 3 s, which works well in housctl. We have not investigated shorter cycles.

Picomotor Library

The picomotors drive the Rx steering mirror. They are very slow, and can take several seconds to complete an operation. housctl returns the ">" response code for picomotor moves, which tells TUI that the command is queued, and will be completed in the future. It takes too long to hog the GPIB semaphore for these operations, so Picomotor moves are threaded out, and the thread releases the semaphore after starting the operation. The thread then waits for a computed amount of time sufficient for the picomotors to complete. Next, it enters a poll loop (with sleeps in each iteration), to insure the picomotor has indeed completed the operation. Then the thread terminates. The picomotor library therefore provides functions to initiate, and to poll for completion of, motor moves.

The library also provides routines for a complete move in one call, for applications which don't care about the long-time to complete.

STV (Video Camera) Library

From Adam Orin's original 'stv-library-readme.txt':

```
************************************************************
STV LIBRARY DOCUMENTATION     ADAM ORIN   FALL 2005
************************************************************
RELEVANT FILES
************************************************************
stv_lib.h         Header file for stv_lib.c
stv_lib.c         Library with functions to interact with the STV
stvadam.c         Human interface to STV that uses stv_lib

Via 'apollo.h', uses functions defined in ser_util.c to communicate via
serial port.

************************************************************
```

```
STV_LIB
************************************************************

Look at the header file for function definitions and usage. The PUBLIC
functions are intended to be used by outside applications.  The PRIVATE
functions are used only within the library itself.

To use the functions, you open the port, call whichever functions you
want to call, then close the port.

Look at stvadam.c to see example uses of this library.

Public functions:

Open the port to the STV
stv_open() ;

Close the port to the STV
stv_close(int fd) ;

Simulate a keypress on the STV
stv_presskey(int fd, uint16 key) ;

Get back an acknowledge packet from the STV
stv_ack(int fd) ;

Get the text display of the STV
stv_getscreen(int fd, uint8 disp[]) ;

Get info on which image buffers have data
stv_get_buffer_info(int fd, uint16 fullbuffers[]) ;

Download an image from specified buffer to a file
stv_download_image(int fd, uint16 flashbuffer, char imgname[]) ;

Download the image currently displayed on the STV
stv_getdispimg(int fd, char imgname[]) ;

Download all images in the STV
stv_getallimg(int fd, char imgname[]) ;

Quickly put the STV into focus mode
stv_quickvid(int fd, int desired_sens) ;

************************************************************
STVADAM
************************************************************
This is a program that lets you remotely operate the STV.

************************************************************
STV COMMUNICATION INFO -- INFORMATION DUPLICATED IN STV_LIB.H
************************************************************
The stuff written here is what I've learned in conjunction with the STV
Command Protocol documentation. If I were you, I would read through that
before looking at this mess.

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
PACKET DATA FORMAT
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Here is how the data format works
Byte 1 = 0xa5
Byte 2 = Command
Byte 3 = Number of data bytes (least sig byte)
Byte 4 = Number of data bytes (most sig byte)
Byte 5 = Header checksum (least sig byte)
Byte 6 = Header checksum (most sig byte)
Byte 7 = Data byte 1
Byte 8 = Bata byte 2
...
Byte 6+N = Data byte N
Byte 6+N+1 = Data checksum (least sig byte)
Byte 6+N+2 = Data checksum (most sig byte)

If a command takes in no data, it is 6 bytes long. If it takes data it
is 8 bytes plus the number of data bytes long.

To calculate header checksum:  (byte1 + byte2 + byte3 + byte4) % 65536
-Add the 4 header bytes, the result is a 16 bit unsigned int
-Cut the 16 bit checksum into two 8 bit chunks
-Byte 5 = least sig half of the checksum
-Byte 6 = most sig half of the checksum

To calc data checksum: Same algorithm as header checksum, except
performed on the data bytes.

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

COMMANDS, AND WHAT TO EXPECT FROM THE STV IN RETURN
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

!!Note = Command 0x10 Request Acknowledge doesn't work - don't know how
much/what data to send

=-=-=-=-=-=-
SC_PRESS = press button.  Send in 2 bytes of data that are the button to
push.  The STV will return an ACK packet (6 bytes)

=-=-=-=-=-=-
SC_ECHO = echo display.  Send 2 bytes of data, data[0] and data[1], int
hat order. If data={1,0}, the STV will immediately return the display,
and will return the display every time a button is pressed. The header
returned is (0xa5 0x9 0x30 0x0 0xde 0x0), where 0x30 = dec48, the number
of bytes in the data returned.  If data={0,0}, the STV will stop sending
the display and returns seemingly nothing (the documentation says it
returns an ACK, but it hasn't for me)

NOTE!! Although the STV seems to not reply when you send it data={0,0},
I've noticed display data in the computer's input buffer before.  There
are inflush calls to prevent any screen data from being a problem, but I
need to get to the bottom of this.

NOTE!! Sometimes, maybe about 30 percent of the time, the STV does reply
to data={0,0} by sending the entire screen. Except, one of the
characters int he screen that would have been a whitespace (HEX 20) is
replaces by HEX A4, which represents nothing in ASCII, and the computer
displays as a whitespace.

=-=-=-=-=-=-
```

SC_BUFSTAT = request flash buffer status. STV responds with typical header junk, plus four data bytes that tell you which buffers have image data. That would be 12 bytes total that the STV will send you in response.

=-=
DOWNLOADING IMAGES FROM STV
=-=
To download an image stored in one of the STV's 31 flash buffers, it appears that you have to download it row by row.

NOTE!! You must request image info before STV will respond to a request for image data. At least, it seems that way. So, to download an image in buffer X, you send a request image info command. The STV sends

NOTE!! It appears that the STV will not always let you download an image regardless of what state it's in. For example, once you download an image, the display often changes to "DOWNLOAD ALL". If you request image info for a second image, you get nothing back. But simply pressing the FILEOPS key, for example, seems to put the STV in a state that will allow it to send another image.

SC_REQIMGINFO = request image info. You tell the STV which buffer you are interested in, and the STV send you 44 data bytes of info (apparently) plus the usual 8 bytes of header crap. That's 52 bytes total.

SC_REQIMG = request image data. You ask the STV for a specific row of image data in a specific buffer, and the STV gives it to you, along with the normal 8 bytes of header crap. Looks like you have to go one row at a time, from row 0 to 199 (for a 320*200 image). I just set left=0; I think that's if you don't what the entire row. And, if you get the whole row of a 320*200 image, length = 320, not 319, dumass.

=-=
FOCUS MODE SENSITIVITY SETTINGS
=-=
01 = 1 ms exposure, 1x gain
02 = 1 ms, 1x ?? Difference between 1 and 2?
03 = 3 ms, 1x
04 = 10 ms, 1x
05 = 20 ms, 1x
06 = 25 ms, 1x !! This is the setting we often use when ranging
07 = 50 ms, 1x
08 = .10 s, 1x
09 = .25 s, 1x
10 = .25 s, 2x
11 = .25 s, 4x
12 = .25 s, 8x
13 = .25 s, 16x
14 = .50 s, 16x
15 = 1.0 s, 16x
16 = 2.5 s, 16x

From: Adam Orin
Sent: Tuesday, April 24, 2007 19:24

```
To: Eric L. Michelsen
```

I remember an occasional weird character in the display. The STV has some very strange looking characters that it sometimes displays. I cannot remember what they are, but if you play with the physical STV you will see some little characters on the display that are pretty unusual characters. .... I've never seen them before on a PC. So they don't display properly using the STV library because I think they are not standard characters. My memory may be muddled but I think this is the source of the nonsensical characters using the stv library.

The STV puts control characters in the display, which messes up the housctl/ICC/hub/TUI stream. I guess we need housctl to escape them, perhaps like C (\x..) or like URLs (%XX).

WTI Library

Turn outlets on/off. To read status, and confirm successful control, this library has to parse the ASCII-graphics display output from the WTI bar.

Apollo/Houston has 3 different controls for AC outlets: DAQ digital outputs 0-7, Terminal Server ports 9-16, and the WTI power bar outlets (1-4). The DAQ and TermServ are controlled with library functions in the DAQ and TermServ libraries.

9 Various Issues Desiring Resolution

Lost Data From Temperature Taking

```
From: James Battat
Sent: Wednesday, September 27, 2006 7:21
To: Tom Murphy
Cc: Eric L. Michelsen
Subject: missed lunar FRC errors in .run files
```

I looked at the 4 run files from August 12th, a night where you got lunar returns, and see that there are missed FRC errors that appear as rem lines in the .run file. e.g.:

```
rem Lunar 196 missed FRC 97253604
```

It appears as if these errors are often brought about when a tmp0 record is taken, presumably because it takes a long time to read 24 voltages off of the ADC. In each run file there are 51 tmp0 records and 35-41 of them trigger at least one lunar FRC errors.

Also there are often many more errors than tmp0-triggered-errors, however these numbers can be misleading because it is often the case that there are 45 or so consecutive frc error statements (is this how long it takes for the q to clear?). For example, in the first file, there are 151 errors. However, two of these errors came in groups 48 and 47 errors long. So that there were really:

```
 151-48-47+2 = 58 *triggers* of FRC errors.  Of these, 38, or ~2/3 were due
to tmp0 data collection.
```

In these runs, we lost between 1.5% and 6.5% of the 10,000 shots because of these errors. Not terrible, but not negligible either.

```
run filename       #errors  #errsNextTo_tmp0  #tmp0recs dt_run
060812-114722.run  151      38                  51      8m25sec
060812-115759.run  548      35                  51      8m30sec
060812-120912.run  518      36                  51      8m27sec
060812-121809.run  653      41                  51      8m29sec

by the way, to get these stats i did:
to get the total number of errors
  > grep "rem Lunar" file.run | wc
to get the number of errors triggered by a tmp0 read
  > grep -B 1 -A 1 "rem Lunar" file.run | grep tmp0 | wc
```

where "-B 1" gives one line of context before the match and "-A 1" gives one line after the match

Perhaps the issue is that internal housctl processes can create the lun frc problems

Also, in the case that a lun FRC error did happen, I thought that there was a software workaround to ensure that there would not be a whole string of successive errors (until the queue cleared).

?? I don't know off the top of my head. -ELM

ICC Connection Issues

3/7/2007: I (Eric M) made many changes to improve ICC session robustness since these messages, but I can't tell if I fixed these issues specifically.

```
From: Tom Murphy [mailto:tmurphy@physics.ucsd.edu]
Sent: Saturday, September 30, 2006 22:59
To: Eric L. Michelsen
Cc: Hoyle C.D.
Subject: RE: ICC disconnect
```

But for instance, the "ConnDev" button on the TUI window requests that the ICC establish a connection to housctl. Pressing this button results in an entry in the log file like: accepted ICC connection #1.

Hitting the disconnect tears down the ICC-housctl connection, and you see that entered in the log file as well.

The confusing thing is: simpy exiting TUI and restarting TUI then issuing a ConnDev accepts ICC connection #1 again. So its as if housctl knows its the same connection or that the old one went away, even though it is not commanded and there is no entry for disconnnect in the log.

```
From Eric L. Michelsen:
> TUI connects to ICC, not housctl directly.  I thought ICC kept the link
> to housctl up all the time, even after TUI disconnects.  In that case,
> you won't see an ICC disconnect in the log, but you shouldn't see an ICC
> connect in the log either.  Are you seeing an explicit connect in the
> log, or just that the ICC session is still #1?  The latter is what I
> expect.
>
> However, if ICC disconnects from its side, it can kill housctl.  This is
> the bad behavior buried in the C Run Time Library that we have to hunt
> down.  I do not know exactly which RTL routine commits suicide, or how
> to stop it.  Since you're not seeing housctl killed, that further
> suggests that ICC is keeping the link to housctl up.
>
>> -----Original Message-----
>> From: Tom Murphy [mailto:tmurphy@physics.ucsd.edu]
>> Sent: Saturday, September 30, 2006 14:39
>> To: Michelsen Eric
>> Cc: Hoyle C.D.; Battat James; Russet McMillan ; Swanson Erik
>> Subject: ICC disconnect
>>
```

>> I've been careful in the past to issue a "disconnect" from the TUI control window before exiting TUI. My fear was that we would accumulate unterminated ICC sessions if we didn't take this step. C.D. reported that he didn't usually do this, and had no problem. I checked into it this morning, and found that I could indeed ignore this step, and each new session (after restarting TUI) was accepted by housctl as session #1. Yet I never saw a message in the log indicating that session #1 got disconnected. So I like the way it works, but wonder if this is expected behavior. If housctl never acknowledges disconnect of session #1, how is it happy with a second session #1? Maybe I missed the disconnect message in the scrolling log, but I don't think so...

10 Support Software

HTML Environmental Monitor

To start the HTML environmental monitor: On cocoa:

```
cd /home/environment
python htmlmonitor.py &
```

Why doesn't Linux kill this when the user logs out ??

```
From: James Battat
Sent: Tuesday, January 23, 2007 17:09
To: Tom Murphy
Cc: Eric L. Michelsen; 'C.D. Hoyle'
Subject: Re: Calibrated temperatures
```

htmlmonitor is designed to look for tmpN records where N is any number 0-9, so it should be able to find the temperature records.

```
From: James Battat
Sent: Tuesday, November 22, 2005 4:44
To: Tom Murphy; 'Adam Orin'; Eric L. Michelsen
Subject: Web status monitor
```

Based on the phone con yesterday, I made a *very* preliminary structure for a web-based version of my system monitor. I would like to keep things simple (to minimize transfer times) while still providing useful and complete information to the user.

In this first pass, only the RTD Temperatures and the Power Status of 24 devices are included. The values are color coded in a completely arbitrary way now (if T > 19 then box is red), just to show that the feature exists. Thus hi/low alarms on any particular value can be set to alert the eye to values that are out of range. The power status is red if off, green if on and white if unknown (no status change during this logfile).

Because I am running this on my harvard account, you cannot see it update as it would on houston, but rest assured that I have tested the update feature and it does work.
http://www.cfa.harvard.edu/~jbattat/apollo/sysmon/

Environmental Plot Generation

From: Adam Orin
Sent: Tuesday, April 18, 2006 14:50
To: James Battat; Tom Murphy; Eric Michelsen
Subject: Re: environment plot suggestion

I have a script running that will archive the environment plots once a day and makes a barebones html file to view them online. You can see them at

http://cocoa.apo.nmsu.edu/environment/archive/

STV Images

```
From: James Battat
Sent: Thursday, January 26, 2006 16:25
To: 'Adam Orin'; Eric L. Michelsen; Tom Murphy
Subject: FITS images from the STV
```

We now also have the capability of downloading images from the STV camera in FITS format. I've attached an image that we downloaded today, in FITS format, from the stv (the image has been sitting in the STV buffer for a while now, either from the Jan or the Dec trip).

As yet there is no useful information in the FITS header. Adam said that he can parse the binary info that the STV provides and then this info can be directly incorporated into the fits header. If we're really clever, we can even grab info from the TCC (RA/DEC of the current pointing position, etc) and stuff that in as well (if that is deemed useful).

It looks like, for each image, the STV will provide the info that is listed in the IMAGE_INFO structure. This includes:

```
     image size (Height, width)
     image exposure time
     Number of added exposures
     Analog Gain
     Digital Gain
     Date/time of image
     Display background (?)
     Display range
     Pedestal
```

and some fixed values:

```
     Focal Length of telescope
     aperture diameter (scope or camera?)
     Site ID
```

I would like to overlay on all images the boundary (or at least vertices) of the APD array, so in the Feb trip we can get the pixel values of the APD vertices on the STV display.

Lunar Prediction

We need to predict the moon’s range to within about +/30 ns or so, round trip, to catch the photon signal in our gate window. UT1-UTC drift can accumulate 3-4 ns/day of prediction error. So now we extrapolate the UT1-UTC drift from prior days in the EOPC file.

Determining Shot Time for Prediction From FRC & TWS

```
From: Tom Murphy
Sent: Sunday, December 03, 2006 12:51
To: Eric L. Michelsen
Subject: RE: losing time
```

So if we find implementation of this scheme in housctl is feasible, we could remove our sensitivity to GPS interruptions as far as the prediction goes. We still have to establish an early event where the GPS clock is right: we can do this by comparing the gpstrig record to the next TWS, which should match the fractional seconds field.

I can also implement a data reduction scheme to do the same thing. In this case, both prediction and even/odd identification are affected.

I like that we can use a built-in sanity check to know whether there has been any slippage. I check for this in the reduction, and have never seen slippage between the FRC and TWS. A good sign.

```
From Eric L. Michelsen
```

I believe we can directly calculate the time delta from the TWS and FRC values. Assume, for now, that both counters start at 0, and count the same ticks, with no slips or errors.

When the FRC first wraps to 0, the TWS will show 2^28 mod 5e7 = 18,435,456. Each wrap increments the TWS by the same 18,435,456. Eventually, the TWS will again wrap to 0 at the same time as FRC, and we return to our initial state. We find this "maxwrap" period by removing the greatest common factor from 5e7 and 18,435,456, which is 128, leaving 5e7/128 = 390,625. (For future reference, note that 18,435,456/128 = 144,027.)

Since each FRC wrap is about 5 sec, this allows for 5*390,625 = 1,953,125 s ~= 542 hours. So long as our outage during a run is less than 542 hours, we can unambiguously know the time delta from two

TWS/FRC pairs. For any single TWS/FRC pair, we can compute the "wrapcount" by finding what TWS was when FRC was 0: just subtract (TWS - FRC)/128. This parameter will determine the number of FRC wraps since 0/0. The wrap parameter is

```
      wp =(TWS - FRC)/128 mod 390,625.
```

We don't want wp, which counts in a weird order; we want wc, the actual count of FRC wraps. But wp and wc are related by

```
      wp = wc*144,027 mod 390,625
Which means       wc = wp*(144,027^-1) mod 390,625
where             144,027^-1 = the multiplicative inverse mod 390,625 =
280,088,
i.e.              (144,027*280,088) mod 390,625 = 1
```

We now compute the final delta T from (TWS1, FRC1) and (TWS2, FRC2) pairs:

```
wc1 = wc(TWS1, FRC1)    // as above
wc2 = wc(TWS2, FRC2)
dt = FRC2 - FRC1 + (wc2 - wc1)*2^28       // in ticks
```

We have to be careful with computing wc, because the formula exceeds 32-bit integers, but I use 64-bit integers in lots of places in housctl already.

There is a simple way to deal with the likely situation that both counters do not start at 0: we can find the offset from zero from the initial TWS/FRC pair, and use this offset for the final pair as well. In fact, we can verify that the two offsets agree. If they don't, there must have been a slip between the two clocks. Then we won't know which one to believe.

```
From: Tom Murphy
Sent: Friday, December 01, 2006 13:59
To: Eric Michelsen
Subject: losing time
```

I looked briefly at the problem of deducing elapsed time from the TWS/FRC comparison. If you have two events, each with an FRC and a TWS value, and you take the difference between the two (adding one period if negative), then you can define the difference (my lower case d is a Delta):

```
dFRC = dt % T1
dTWS = 2*(dt % T2)
```

where dt is tha actual time (in 20 ns clicks) elapsed between the events, T1 is 2^28, and T2 is 5e7. TWS, remember, is represented with a factor of 2 for easy readability.

It is hard to invert the % function, so this is a pesky problem.

You can, in principle, construct an array of N values:

```
      [0,1,2,3,4,5,...]
```

and for each value make a dt estimate: dt_est = dFRC + N*T1. Then compare to the dTWS by computing 2*(dt_est % T2) - dTWS. There should be one zero somewhere in the array, which tells you how many wraps the FRC had.

I've tried this and it works well, but it's a nuisance that you can't go straight to a closed solution without trying various N values.

Polynomial Fitting

We run the polynomial generator program must each day, because the earth's chaotic wobble is measured and published every day. We could probably tolerate a week or two of missing data and be ok, but this is not tested. The program downloads EOP (earth orientation parameters) measurements from a web site, and computes polynomial fits the give the RTT (round trip time) to the moon, given the laser shot time. There is something majorly wrong with this program, because it cannot fit over more than a few hours, and the fitting routines are not stable, and sometimes fail to converge. The lunar path is simple enough that we should not have these problems.

It produces a file of numbers in this format (text added on the right):

```
2                          reflector: 0=Apollo 11, 2=14, 3=15, 4=Lunakod?
205.333333333488554        t0: start time of fit: days since start of year
205.486111111007631        end time of fit: days since start of year
8.08117e-05                RMS error of fit, ns
0.0003104184               max error of fit, ns
220.000000                 span of fit, minutes
5                          minutes between fit points
2.381101499786292          midpoint RTT, sec
7.917010                   range rate, ns/gate (at 20 shots/sec), i.e. ns/50 ms
9                          number of coefficients = order + 1
2.383526638827937e+00      a0
-7.811391819226714e-02     a1
5.761043343230440e-01      a2
6.732126750138415e-01      a3
-1.795798333981848e+00     a4
-1.455305024451972e+00     a5
2.393287589516413e+00      a6
2.304098432739442e+00      a7
-3.370381157585106e+00     a8
```

The order of the polynomial is *variable*; this example shows 8th order.

The polynomial parameter is t = curtime - t0, in days (t << 1). Then

$$RTT = a_8t^8 + a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$$

It is well established that numerical accuracy is improved by evaluating the polynomial in factored form:

$$RTT = \left(\left(\left(\left(\left(\left((a_8t + a_7)t + a_6\right)t + a_5\right)t + a_4\right)t + a_3\right)t + a_2t\right) + a_1\right)t + a_0$$

Eric M. would like to change the file format to be more self-documenting.

```
From: Tom Murphy
Sent: Saturday, May 06, 2006 13:24
To: Michelsen Eric
Subject: mkpoly zap vs houston vs cocoa
```

One more piece to the puzzle: the prediction numbers are identical between houston and cocoa. And zap to houston was identical. My laptop to zap had occasional last-digit variance. My laptop to grlab has 2-3 digit disagreement. So grlab seems to still be unusual.

```
From: Tom Murphy
Sent: Friday, May 05, 2006 13:34
To: apollo_core@u.washington.edu
Cc: owen@astro.washington.edu; mcmillan@apo.nmsu.edu
Subject: [Apollo_core] (no subject)
```

I have prepared a few software packages for your use/amusement. I attach here a tar file that will unpack to a lunar/ directory that contains some useful Python code. Among these is a version of the old moon.c program that computes position and tracking information for the moon. Now I have it in 100% Python form. The Python programs in this package are:

moon_pos.py: computes lunar position, tracking, and other useful info

moonplan.py: computes start and stop times and other relevant stuff for an observing session

longplan.py: computes observing constraints for a whole quarter, to aid planning lunar_pointer.py: graphical display of lunar illumination, orientation, etc. This is the original code behind the APOLLO TUI pointer tool.

The last three programs rely on the moon_pos.py program. The C version is 10 times faster, and interchangeable--but less portable. So what I provide here is at least fully self-contained. I attach the README file here dealing with the care and feeding of these programs (also unpacks in the tar).

Polynomial Prediction Software Now On houston

From Tom: I have put the polynomial-generating prediction software on houston, so that the observers from now on can expect to use houston to generate polynomials for the night. It's in /home/apollo/ephem/predict/. See the README file in that directory (also attached here: second "README") for detailed instructions on how to create the polynomials.

Latitude and Longitude

```
From: Tom Murphy
Sent: Sunday, August 06, 2006 4:39
To: ~ Buttery ~
Cc: tmurphy@physics.ucsd.edu; emichels@physics.ucsd.edu
```

MCD is the mcdonald obs. early on, we made comparisons to their predictions. But we don't need this any more.

The old lat value was before we knew we needed geocentric rather than geodetic latitude. At this point we expect only small variations around the top two in the lines below.

From Aaron Buttery: Hey Tom, I was going through the ephem.h file and I can across these values:

```
#define   LAT          32.6054942   // geocentric latitude
#define   LON          254.1795718  // deduced from Oct 2005 runs

//#define LAT          32.779561    // matched to 7/24 preds
//#define LON          254.163194   // matched to 7/24 preds

#define   WGSLAT       32.780361    // WGS84 geodetic latitude

#define   LAT_MCD              30.51171089
#define   LON_MCD              255.9848038
```

Rolling Polynomials

It is much simpler, more reliable, more accurate, and faster to have housctl generate rolling 4th order polynomials directly from the 5-minute-spaced ephemeris points, on the fly with each run. Preliminary results:

Recalling that the odd order terms didn't gain much in Aaron's work, I threw in a test for an explicit, closed-form rolling 4th order fit, with dramatically better results than 3rd order: sub-ps over an hour (compared to existing variable-order polynomial). Computational burden is only slightly higher than 3rd order version. In fact, we can compute the new set of 4th order coefficients, *and* evaluate the 4th order polynomial, in about the same computrons as evaluating a single 12th order polynomial once.

Also, the only reason the 4th order result was limited to 1 hour was artificial. In fact, I only used 1/2 of a fit interval, so it's almost certainly good for 2 hours. And even that is an artificial limit. I don't know, with this quick test, how much longer it would go. I'll look into some other fit files, but it didn't look to me like the ephemeris files for the longer (earlier in the night) stretches were saved.

There's nothing magic about 5 min intervals. We could easily do, say, 3 min or others at insignificant cost.

From: Tom Murphy

```
Sent: Thursday, December 07, 2006 16:04
Subject: RE: 5 point, 4th order interpolations
```

I sure like this number. It means we're not giving up our current (Evan's) level of precision--which we would do adopting Aaron's. As we get more into plotting residuals against our prediction, the more I'm interested in a polynomial that does not dominate the error. So ps-level is great. We should try your routine on a number of nights/segments to get a good feeling for it's representative precision.

From Eric M: I don't think residuals against our prediction are the key figure. It's residuals against our fit for the normal point that really matter. I think we need to make sure that our prediction is never a significant factor in any of our data reduction.

I still favor a fit to our data without reference to our prediction at all. I have code for that, already, from my simulator analysis. I will start to update it for real run files, which I need for first photon bias work.

Eric M’s older results for 3rd order interpolations show rapid degradation with given point spacing, as expected:

5 min interval	0.13 ns max difference from current polynomial fit
10 min	2 ns
15 min	10 ns

This is all for a single few-hour polynomial file. Actual results over more diversity would vary some, but I'll bet not very much.

Aaron Buttery’s Work On One Polynomial Per Night

```
    From: AAron Buttery
    Sent: Tuesday, October 03, 2006 14:27
    To: tmurphy@physics.ucsd.edu; emichels@physics.ucsd.edu
    Subject:
```

...I went in at the end of last week to try and leave a readme file, but my computer froze and I lost it all. This is pretty much what it said.

Aaron's last problems:

1) The main one was the discrepancy between the max errors when you would start at different times of the day. i.e. h=0 m=0 s=0 vs 16 20 0.

2) The format of the output files is not quite what I wanted. It currently has the year as a four digit number instead of two.

3) There is still a small glitch when some reflectors rise or set before all the other ones. For some reason the times get all screwed up. Tf<Ti which makes no sense. You can find an example of this happening on Feb 7 2006.

I thought there were four in my original message, but I can't seem to remember the last one. ....

Prediction Code Design

From Aaron’s README file:

From the prediction directory: /home/apollo/predict/, make a directory for this day, and cd into it:

```
    prompt$ mkdir 060505
```

(YYMMDD format to match our data storage convention)

```
    prompt$ cd 060505
```

get the earth orientation parameters:

```
    prompt$ ../bin/autoget
```

Lately (As of ~6/1/06) we've been getting the warning:

```
"WARNING! 267 bare linefeeds received in ASCII mode
File may not have transferred correctly."
```

but everything seems to still work fine.

```
APR  25  53850 0.105999 0.362269 0.2442683   0.0018148   -0.05299 -0.00574
APR  26  53851 0.106224 0.361873 0.2424549   0.0018254   -0.05333 -0.00543
APR  27  53852 0.106531 0.361691 0.2406734   0.0017161   -0.05373 -0.00530
APR  28  53853 0.107324 0.361485 0.2390578   0.0014789   -0.05413 -0.00536
APR  29  53854 0.108535 0.361198 0.2377216   0.0011565   -0.05445 -0.00554
APR  30  53855 0.109473 0.360750 0.2367293   0.0008635   -0.05457 -0.00580
MAY   1  53856 0.109797 0.360148 0.2359569   0.0006926   -0.05438 -0.00612
MAY   2  53857 0.109831 0.359559 0.2352450   0.0006901   -0.05398 -0.00637
MAY   3  53858 0.109956 0.358914 0.2344719   0.0008828   -0.05363 -0.00641
MAY   4  53859 0.109751 0.358103 0.2335858   0.0008140   -0.05348 -0.00627
```

Verify that end date is close to today's date (may lag a few days).

Generate polynomials:

```
prompt$ ../bin/mkpoly [args]
```

There are four options for [args]:

```
blank: uses computer time (now) as start time, uses 20.0 deg elevation
limit
one arg: uses computer time as start time, uses single arg as elev. limit
6 args: MM DD YYYY HH MM SS.SS as start time, 20.0 elevation limit
7 args: MM DD YYYY HH MM SS.SS EL.EL as start and elevation limit
```

Time is in UTC. Examples:

```
prompt$ ../bin/mkpoly
prompt$ ../bin/mkpoly 18.5
prompt$ ../bin/mkpoly 5 5 2006 20 30 0.0
prompt$ ../bin/mkpoly 5 5 2006 20 30 0.0 18.5
```

The third example is the most commonly used.

The program runs 160 predictions at 5 minute intervals, spitting out the following sort of block for each time:

```
(69)    5/ 6/2006  2:10: 0      Julian date = 2453861.5910322224
Center  2.652400030034  73.269903
0       2.641631319332  73.307542
2       2.641510678121  73.251683
3       2.642388993257  73.393747
4       2.643621854104  73.412526
```

This is the 69th point in the sequence. The time (UTC) is May 6, 2006, 2:10:00.00, with the corresponding Julian date (this in TDT). The next five lines refer to the center of mass of the moon, and the four reflectors [0 = A11, 2 = A14, 3 = A15, 4 = L2]. Following each reflector identifier is the round trip travel time and elevation in seconds and degrees, respectively.

At the end, one sees:

```
(130)   5/ 6/2006  7:15: 0      Julian date = 2453861.8028377779
Center  2.681085111968  18.755889
0       2.670374390341  18.707072
2       2.670135250381  18.807497
3       2.671015710180  18.842463
4       2.672324978496  18.773427
(131)   5/ 6/2006  7:20: 0      Julian date = 2453861.8063099999
Moon Below 18.0 degrees elevation
Moon Below 18.0 degrees elevation
Moon Below 18.0 degrees elevation
Moon Below 18.0 degrees elevation
Moon Below 18.0 degrees elevation
Ceasing data prediction because the moon has set
128
Congratulations, Fits created successfully
```

On the 131st point, the moon was below 20 degrees for all five targets, so the process halts. 128 points went into the fit (why not 130? the first two points were also below the elevation limit). The fit was successful.

You can check details of the fit by looking at the generated files:

Date:	has time stamps (UTC) for the data points
elevation:	has elevations for the center of the moon at each time
fit*:	the round trip travel times for each refl at each time
np*:	fake normal point sets for the prediction
Reflector#(Date):	polynomial information. Format is:

```
# mkpoly v. Sep  6 2006
# apollo ?
reflector = ?
Year = 2006
ti = 250.097581018693745      Start time. Julian day.
t0 = 250.279872685360412      Time at mid span.
tf = 250.458692129701376      End time.
timespan_min = 525
fitinterval_min = 5
midspan = 2.346122500871582
rangerate = -6.538189
rmserr_ns = 0.001226
maxerr_ns = 0.003964
nData  = 105
ncoeff = 11
a0 = 2.346122500871582201637e+00
a1 = -1.129799035253190379169e-02
a2 = 6.619814355871829446840e-01
a3 = -5.702782078463447752955e-03
a4 = -2.102100464587396298852e+00
a5 = -1.000691990949274746883e-02
a6 = 3.057039733951018886269e+00
a7 = 3.234121354439520977864e-02
a8 = -3.418348059065643128503e+00
a9 = -4.406308478007229962470e-02
a10 = 4.127021706229873293428e+00
Pressure = 725.000000
TEMP_C = 0.000000
HUMIDITY = 33.000000
P = 6374.692130
LAT = 32.605494
LON = 254.179572
LUNRHOZERO = 1735.472177
LUNLATZERO = 0.693506
LUNLONZERO = 23.455288
eopcyear = 2006
eopcmonth = 9
eopcday = 5
eopcmjd = 53983.000000
xpoledph = 0.077150
ypoledph = 0.254169
ut = 0.169730
lod = 0.000467
dpsi = -0.065920
deps = -0.005520
```

Place the reflector# files into /home/apollo/daily/. This step MUST BE TAKEN in order for housctl to actually use these polynomials.

Problems:

If you get an error at the end like:

```
147
Cannot acheive desired accuracy over interval
```

there are a few things you can try to get it to work:

1. change the elevation limit: higher usually eases the burden
2. change the start time: if not by 5 minute intervals, try even one minute intervals. There is an instability that can sometimes be remedied by a slight change in the start time, without changing the elevation limit.

Evan M's Notes from predict.old directory

Section 1: Creating the binary files needed to run mkpoly.c

Begin by downloading files from ftp://ssd.jpl.nasa.gov/pub/eph/export. You will need ascp1975.403, ascp2000.403, and ascp2025.403 and header.403. These files still exist on GRlab in emillion/ephem/jpl/de405. You will need to change all the exponents from D's to E's to correctly make the binary files since D is a fortran convention and E is a C covention in header.403 and ascp2000.403. You will also need to run asc2eph.c to create the appropriate binary files. To do this you need to attach the name of each binary file to the end of header.403 that is included. You only really need ascp2000.403 since the experiment will be running between 2000 and 2025. Look at the start and end times of ascp2000.403 and you will find the times you should run asc2eph.c between. With the binary file, you must put its file name in the appropriate spot in moon.c. It is on line 214. I am taking you through this since each computer likes to do binary files a different way. If you wish to run ascp1975.403 and ascp2025.403 you must change the name of the file to either one at the bottom of header.403 before running asc2eph.c and should also change the name of the binary file. I've left it ready for you to use for ascp2000.403. Compile and run. It will ask for two Julian Dates. These are them for ascp2000.403 only, Begin: 2451536.5, End: 2460688.5. To merge each binary file you must run merge.c. On lines 168, 169, and 170 are where each file is read and combined with third file being the resultant file at the end. Remember to recompile merge.c each time with new file names. When you have a big binary file you can place it in moon.c for use. Search using vi /FILE *INITIAL and place the binary file in the char *Name="xxxxxx" line several lines below.

Section 2: mkpoly.c

Here is a list of all the files that mkpoly.c requires to run properly with what they do.

Binary file **-** Search for /FILE *INITIAL and put it in the char *Name="Binary File"; line.

eopc04.05 **-** opened in eopc.c and is the earth orientation parameter file.

Configure **-** part of the make support

Makefile **-** Allows you to type make mkpoly and install the program

dist.c **-** takes the vectorial distance at the end, calls for the atmospheric drag, and calculates the time it takes for a photon to travel a leg of the journey.

eopc.c **-** opens the earth orientation parameter file for use in moon.c

ERA.c **-** calculates the earth rotation angle and rotates the earth vectors through it

jde.c **-** calculates the julian date and is split into two parts

julianday.c **-** calculates the julian date out of a given date for greater precision.

lib.c **-** uses the ephemeris to find the libration angles and rotates the lunar vector through them

mjd.c **-** calculates the number of days since Jan 1, 2000

mm3x3.c **-** multiplies 2 matrices together from 2 9 element vectors mkpoly.c - the main program that does everything

mv3x3.c - multiplies a matrix by a column vector, the matrix is stored in a 9 element vector

nut.c - searches through the eop file for nutation corrections

pole.c - searches through the eop file for the daily value for x" and y" polar wobble

prec.c - calculates the precession of the earth

Qrot.c - rotates the earth vector through the precession angles

range.c - calculates the resultant of the 3 vectors

reader.c - puts the downloaded file from iers.org into a more readable form for mkpoly.c

refraction.c - calculates the time delay from light refracting through the atmosphere, both ways

rel.c - calculates the relativistic time delay due to gravitational effects

rone.c - general rotation through the x axis

rtwo.c - general rotation through the y axis

rthree.c - general rotation through the z axis

ut.c - searches the eop file for the ut1-utc correction needed for the earth rotation angle

zenith.c - calculates the elevation needed for the refraction delay.

ephem.h - header file needed for mkpoly.c

The following files are part of the fitting process
```
nrutil.h - header file for numerical recipes
nrutil.c - general functions needed for numerical recipes
```

pfit.c - This program reads in a list of data and generates a polynomial that best fits the data to the certian error requirements defined in "Good_Enough" and "Too_Bad". Then it returns the coefficients to the polynomial as well as some other stats about the run. For more information look at README.??

Section 3: Setup

Put the files that need to be read in a directory called bin. Put files that are associated with mkpoly in a directory called src. Put the header files in a directory called include. Place the Makefile and Configure files in the main directory. In the main directory you will type "make mkpoly" and it will install the program. The execute will be in the directory bin.

Section 4: The program

Executing the program, it will first automatically retreive the file needed for the earth orientation paramaters. Placing it in bin it will be read and then written to another file so that I did not have to overhaul all my reading programs. The program will calculate the time it will take for a laser pulse to reach the moon and back. To see this data you must go through mkpoly.c and find the printf statements that require to be uncommented. In addition, at the end of this process, the program will then create a file for each reflector. To change the atmospheric conditions used in the fit, please see refraction.c and change the temperature (K), humidity (%), and pressure (mbar) as you see fit. Be warned, the atmospheric model being used here could still have bugs. You can turn it off by commenting out the +dr; in dist.c, but it is unwise to do so. Mkpoly.c will evaluate for each reflector and the center of the moon, the time it would take for a laser pulse to get there and back. This is evaluated in 5 minute intervals. If you care to alter this, as well as any other "important" parameter, look in the include file ephem.h.

At the beginning, if given no arguments, the program will run using the current computer's time. If you wish to start at a different time, then enter it when calling mkpoly. You must enter the times in this order: month day year hour minute second.(Look at line 50 in this file to see how this is done.) Mkpoly.c will then run for 160 iterations and stop. You may increase the number of iterations if you wish to see more of the night.

The fitting program pfit.c uses a multiple linear regression precess. A good reference on how this model works may be found in Walpole and Myers' "Probability and Statistics for Engineers and Scientists".

I have added several checks and balances to prevent the program from being misused. First, if data has been taken and then the moon sets, the prediction process is stopped.

Second, if the number of data points is below 35, it asks whether you wish to continue but asks that you consider starting earlier. Finally, if the total number of data points divided by 40 has a remainder that is 10 or less, it fits 33 points at a time instead of 40.

Section 5: Program dependence and functions contained

dist.c dist(double r[],double c[]) - zenith, refraction

eopc.c *eopc() - none

ERA.c ERA(double jde,double hr,double min,double sec,double r[],double a[] - rthree, mv3x3

jde.c julian(int yr,int mth,double day,double hour,double min,double sec,double jde[]) - jd

julianday.c jd(int yr,int mth,double day,double hr,double min,double sec) - none

lib.c lib(double t,double phi,double theta,double psi,double r[],double f[])

- rthree,rone,mm3x3,mv3x3

mjd.c mjd(int yr,int mth,double day,double hour,double min,double sec) - none

mm3x3.c mm3x3(double a[],double b[],double c[]) - none

mkpoly.c main - jd, julian,pole,ut,jde,ERA,nutcorrect,precnut,lib,relcorrect,range,zenith,dist,pfit

mv3x3.c mv3x3(double a[],double b[],double c[]) - none

nut.c nutcorrect(int mth, double day,int yr,double ang[]) - eopc

pole.c pole(double t,int mth,double day,int yr,double hr,double min,double sec,double r[],double g[])

- eopc,rone,rtwo,mm3x3,rthree,mv3x3

prec.c precnut(double t,double psi,double eps,double r[],double b[]) - qrot,mv3x3

Qrot.c qrot(double x,double y,double s,double c[]) - rthree,mm3x3

range.c range(double lun[],double eop[],double d[]) - none

reader.c reader() - none

refraction.c refraction(double E) - none

rel.c rel(double e[],double p[],double q[]) - none

rone.c rone(double angle,double a[]) - none

rtwo.c rtwo(double angle,double a[]) - none

rthree.c rthree(double angle,double a[]) - none

ut.c ut_one(int mth,double day,int yr,double hour,double min,double sec) - eopc

zenith.c zenith(double z[],double c[]) - none

pfit.c - ldgaussj,mmpow,getcoeff - mkpoly.c nrutil.c nrerror,vector,ivector,matrix,free_vector,free_ivector,free_matrix - none

Section 6: Yearly Maintenance

In autoget, you need to change the file you retreive automatically to eopc04.##, where ## is the last two digits of the year in question. Recompile using chmod +x autoget.

In reader.c, you need to change the file that it reads to the file you downloaded. Also, you need to change the year designation in reader.c to that of 20## so that the file is read correctly.

In eopc.c, you need to make sure that the file it opens is the same as the one you wrote from reader.c.

These should be the only changes you need to make year in and year out.

11 Tracking Loops

Note We have not currently implemented any of these loops (10/3/2007).

Our signal rate is 10x - 20x lower than we predicted, so many of the loops won't work with such a low signal-to-noise ratio (SNR).

For system:

Function	**Target Accuracy**	**Measurement Rate**	**Update Rate**	**Notes**
TDC window alignment (prediction): acquisition	1 ns	per shot	once	Need rapid acquisition. Start with most-recent prior result.
TDC window alignment: tracking	1 ns	per shot	minutes	much slower response
Fiducial intensity	10%	per shot	seconds	Start with last value
Telescope pointing	.5 "pixels"	per shot	~10 sec	
Temperature	1 °C	?	minutes	? Thermal gradient changes w/ ambient?

Probably put all but telescope pointing on Houston; pointing on ICC.

General Tracking Loop Theory

Some tracking loops will likely be simple discrete-time, first-order loops. This means they track a slew-rate with fixed, non-zero error. The target accuracy then means "bias + 3σ".

Noise filtering will be optimized for the worst case. Performance will always be better for less noise, but will not be optimum for the lower-noise conditions. Making loops noise-adaptive is *much* harder, and generally reduces the worst-case performance.

Performance optimization requires an estimate of the worst-case slew-rate of the tracked parameter. This essentially tells you how far back you can rely on parameter estimates, and therefore how many estimates you can average for noise-reduction. Each loop design must specify its assumed slew-rate and noise conditions.

TDC Window Alignment: Acquisition

Our Round Trip Time (RTT) estimate is good only to about 10 ns, which is insufficient for aligning the return photons in the same 20 ns TDC window as the fiducial photons. We seek 1 ns accuracy. For a sinusoidal error of period 12 hours (?? actual error is ~28 days), the worst case slew is

$$e = A\sin\left(2\pi t/T\right) \Rightarrow \qquad \max - slew = 2\pi A/T = 1.5\ ps/s$$

After the system starts receiving lunar photons, the worst-case noise of a single estimate (maximum retroreflector tilt) is (from poster)

RMS(300, 60, 50, 45, 20, 7) = 314 ps

This means we can track to bias + 3σ = 1 ns in a single measurement. (3σ is overkill, because PDF of time variation is very flat). However, lunar stray photons, and thermal electrons (each w/ Pr ~= .001) can corrupt 1 measurement by up to 10 ns. Averaging, say, 20 measurements reduces this to .5 ns. Thus, we can easily align the photons in the TDC window in 1 second.

TDC Window Alignment: Tracking

Using the above numbers, and targeting 0.1 ns accuracy, we need only update every minute. However, it costs almost nothing to update this loop; it only affects how we calculate the window position for lunar returns.

Fiducial Intensity Tracking

Do we need an acquisition phase for this??

This loop reduces “first-photon” bias differences between the fiducial photons and the lunar photons. Since the system is fairly stable from day to day, it seems reasonable to start the quadrant diffuser at the last known position. However, any change to the mechanics (timing belt) destroys that number, and the system must track from scratch. Initial acquisition is unknown at this time.

Telescope Pointing Tracking

TBS