KPIC Phase II – Troubleshooting

Edit Log		
Name	Date (mm/dd/yyyy)	Comment
Daniel Echeverri	05/24/2024	Initial release (to Keck)
Daniel Echeverri	05/31/2024	Added: BMC DM HIL Not Connecting
Daniel Echeverri	06/07/2024	Added: A Conex Stage (SAM X) Not Connecting
Daniel Echeverri	08/05/2024	Added: Visualizer Remains Frozen
Daniel Echeverri	08/06/2024	Updated: A Conex Stage (SAM X) Not Connecting
Daniel Echeverri	09/24/2024	Added: BMC DM Not responding due to RTC Issue
Daniel Echeverri	11/21/2024	Updated: general doc cleanup to improve readability
Daniel Echeverri	12/18/2024	Added: Sam rotator encoder not found

(NOTE: When viewed online, figures may be in odd places. Recommend opening locally)

Table of Contents

Overview (TRY THIS FIRST IN ALL CASES):

This document lists potential issues in KPIC, along with instructions to recover from them. Typically, restarting the KPIC software, by running the shutdown and init scripts, will fix most problems.

Syntax Notes:

- Orange machine font is for commands to be sent in a **normal terminal**
- Purple machine font is for commands to be sent in a **kpython3 instance**

To shut down KPIC software:

```
KPIC_shutdown -rmsems -V
```

The –rmsems flag closes and cleans the shared memory and semaphore files. The –V flag makes the script extra verbose to help debug in case it has trouble with something.
NOTE: If you use -rmsems (as recommended above), it is also recommended to restart any local python instances with connections to the KPIC shared memories.

To start up KPIC software:

```
KPIC_init
```

We invoke this with the default flags (by omission), which are –DLfCSHMKV :
-D starts the dispatcher, -L starts the low-level control scripts, -f allows the script to "forcefully" restart control scripts even in the presence of stale shared memories, -C tells the low-level control scripts to connect to the stages, -S tells them to close the loops (servos) on any relevant stages, -H starts the high-level control scripts, -M starts the monitoring script, -K starts the HIL (kill) switch for the BMC DM, and –V makes it extra verbose for simpler debugging.

Note that these flags can be run independently, in case you want to slowly restart the system and debug at each step. In that case, you should still try to run them in the order above (though the –f flag would be provided for both –L and –H). This is provided here as an option, though it is rarely needed.

Other Overview Notes:

If you feel comfortable enough with KPIC and python in general to troubleshoot specific issues, the rest of the instructions in this document may be useful. *With practice and familiarity, making a targeted fix can be faster than shutting down and restarting KPIC completely*, but it is more involved and may be impractical for early users. To help with deciding which debugging steps to follow, the entries below are listed in priority order of which ones are more common and useful. As such, later ones are very rare (or even one-off) occurrences which we've added just for reference in case they're helpful for solving other issues in the future. This is a living document and users should feel welcome to add entries.

Final comment: it's always possible that an issue observed in KPIC is caused upstream (e.g. if the SFP is in a faulted state). If resetting KPIC doesn't fix the issue, check that the elements used by KPIC (but not fully controlled by KPIC) are functioning correctly.

Contact - In case of major issues that you cannot resolve on your own, **ask your SA**. If you are an SA, or a KPIC team member (including observing collaborators), you can try to contact:
Dan Echeverri
Elijah Anakalea-Buckley
Luke Finnerty

Detailed System Health Check:

<u>Note:</u> this entry isn't so much an issue entry but rather a useful reference for some ways to check the system health.

<u>Checking the health of an individual stage:</u>

The fastest way to check that an individual stage is healthy is to use the KPIC command-line interface (CLI) for that stage to ask for a "status" update. This is done with <stage_name> status -push from any KPIC terminal. For example, you can check the status of the SAM by doing SAM status -push. An example of the result is shown below. By supplying the "-push" flag, you are requiring that the script pull a new value from the stage (hardware) controller itself rather than the latest value from the shared memories. Thus, the "last update" time (shown below for example) should be very recent. If the stage is not referenced, or the control script isn't running, or has some other error like that, it will warn you. If the this command hangs for an extended period of time (most stages answer within a few seconds but some take longer, so always give it 30ish seconds), then there is likely an issue with the stage/control script. In that case, ctrl-c the status request and follow the procedure below for A KPIC Stage is Acting Up.

```
[nfiudev@nfiuserver kpic]$ SAM status -push
Position: [3.1899953 4.3799987] last updated 2024/5/24 13:57:17 HST
[nfiudev@nfiuserver kpic]$
```

For simple state issues (like a control script not being running, or it not being referenced), you can usually use this same CLI to fix the issue. For example, you can start the control script by using SAM enable. Or you can reference the stage using SAM reference. Calling just the CLI without an argument (ex: just doing SAM) will print all the command options, as shown in the screenshot below.

```
[nfiudev@nfiuserver kpic]$ SAM
usage:
  SAM [rot] [CMD] [-h]

CMD:
  goto       - moves device
  status     - returns last recorded position
  loop       - sets/gets loop status (only supported with rot flag)
  reference  - references device
  connect    - connects to device (not supported with rot flag)
  disconnect - disconnects from device (not supported with rot flag)
  enable     - turns on control script
  disable    - turns off control script
  stop       - stops all movement abruptly
-h:
  add -h after a command for more info
```

<u>Checking the tmux sessions/control scripts:</u>

Since all the KPIC daemons/control scripts run in tmux sessions, you can do a quick top-level check of the system health by querying the tmux sessions. Calling tmux ls from any terminal will identify all tmux sessions. For the Devices session, you can use tmux lsw -t Devices to see which "windows" are running in that session. As of May 2024, these should look like the screenshots below. In the one on the left corresponding to tmux ls, note the 8 sessions. It is okay for the "Fiber_Bounce" to be missing. In the right screenshot, corresponding to tmux lsw -t Devices, note that there are 14 (0-13) windows. If there are one or two device windows missing, it is not a big issue, and you can start those devices separately.

```
[nfiudev@nfiuserver ~]$ tmux ls
Devices: 14 windows (created Fri May 24 08:28:31 2024) [80x23]
Fiber_Bounce: 1 windows (created Tue Apr 16 19:34:25 2024) [80x23]
HIL_Telem: 1 windows (created Fri May 24 08:29:00 2024) [80x23]
Processing: 1 windows (created Fri May 24 08:28:55 2024) [80x23]
Strehl: 1 windows (created Fri May 24 08:28:58 2024) [80x23]
Tracking: 1 windows (created Fri May 24 08:28:55 2024) [80x23]
keywords: 1 windows (created Fri May 24 08:28:24 2024) [80x23]
logging: 1 windows (created Fri May 24 08:28:55 2024) [80x23]
```

```
[nfiudev@nfiuserver ~]$ tmux lsw -t Devices
0: Light_Src (1 panes) [80x23] [layout ebe5,80x23,0,0,434] @434
1: PyWFS_Pickoff (1 panes) [80x23] [layout ebe6,80x23,0,0,435] @435
2: Coronagraph (1 panes) [80x23] [layout ebe7,80x23,0,0,436] @436
3: FAM (1 panes) [80x23] [layout ebe8,80x23,0,0,437] @437
4: Filter_Wh (1 panes) [80x23] [layout ebe9,80x23,0,0,438] @438
5: Mode_Change (1 panes) [80x23] [layout ebea,80x23,0,0,439] @439
6: Track_Cam (1 panes) [80x23] [layout 6be2,80x23,0,0,440] @440
7: PIAA (1 panes) [80x23] [layout 6be3,80x23,0,0,441] @441
8: FIU_Fiber (1 panes) [80x23] [layout 6be4,80x23,0,0,442] @442
9: ADC (1 panes) [80x23] [layout 6be5,80x23,0,0,443] @443
10: ADC_Retractor (1 panes) [80x23] [layout 6be6,80x23,0,0,444] @444
11: TCP- (1 panes) [80x23] [layout 6be7,80x23,0,0,445] @445
12: PSM* (1 panes) [80x23] [layout 6be8,80x23,0,0,446] @446 (active)
13: SAM (1 panes) [80x23] [layout 6be9,80x23,0,0,447] @447
```

You can quickly list all the running KPIC daemons using the KPIC_check_processes command in a terminal. The screenshot below shows the result as of August 2024. Your list should contain most of these processes. Again, if one or two stage deamons are missing, you can start those separately.

```
[nfiudev@nfiuserver Facility ~]$ KPIC_check_processes
The following KPIC processes are running:
PID: 73104 | Process: kpython3 RELDIR/bin/kpicctrl_Light_Src
PID: 73146 | Process: kpython3 RELDIR/bin/kpicctrl_PyWFS_Pickoff
PID: 73181 | Process: kpython3 RELDIR/bin/kpicctrl_Coronagraph
PID: 73214 | Process: kpython3 RELDIR/bin/kpicctrl_FAM
PID: 73233 | Process: kpython3 RELDIR/bin/kpicctrl_Filter_Wh
PID: 73252 | Process: kpython3 RELDIR/bin/kpicctrl_Mode_Change
PID: 73332 | Process: kpicctrl_Track_Cam
PID: 73448 | Process: kpython3 RELDIR/bin/kpicctrl_PIAA
PID: 73492 | Process: kpython3 RELDIR/bin/kpicctrl_FIU_Fiber
PID: 73526 | Process: kpython3 RELDIR/bin/kpicctrl_ADC
PID: 73555 | Process: kpython3 RELDIR/bin/kpicctrl_ADC_Retractor
PID: 73593 | Process: kpython3 RELDIR/bin/kpicctrl_TCP
PID: 73605 | Process: kpython3 RELDIR/bin/kpicctrl_PSM
PID: 73635 | Process: kpython3 RELDIR/bin/kpicctrl_SAM
PID: 73748 | Process: kpython3 RELDIR/bin/kpicctrl_Track_Cam_vis_process
PID: 73762 | Process: kpython3 RELDIR/bin/kpicctrl_Star_Tracker
PID: 73950 | Process: kpython3 RELDIR/bin/kpicctrl_Strehl
PID: 74110 | Process: kpython3 RELDIR/bin/kpicctrl_HIL_ioc
PID: 72979 | Process: kpython3 RELDIR/sbin/kpicd -c RELDIR/data/kpic/kpic.conf
PID: 73802 | Process: kpython3 RELDIR/bin/Log_Headers
```

You can “attach” to a tmux session using tmux a -t <session_name> to see what the control script is reporting. For example, doing this with the tracking script (tmux a -t Tracking) will give the first screenshot below (from May 2024). Most scripts print “alive…” with a time-varying number of points (“.”) to show that they are alive and running. In the tracking example below, you can see that the script has had errors in the past, but it recovered from those and is showing “alive.”.

```
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
ERROR:Star_Tracker:fit failed on line 1258 - likely no PSF in image
alive.    Avg Loop Time: 8.91 [ms]]
[Tracking]0:Control*
```

The second screenshot (May 2024) shows the Light Source Retractor tmux (tmux a -t Devices:Light_Src). Again, we see “alive.” indicating a healthy script.

```
tele:tmux:/home/nfiudev> kpicctrl_Light_Src
INFO:Light_Src:No duplicate control script, continuing with initialization
INFO:Light_Src:Shms connected successfully.
alive.


[Devices] 0:Light_Src* 1:PyWFS_Pickoff  2:Coronagraph  3:FAM  4:Filter_Wh  5:Moc
```

The third screenshot (May 2024) shows the ADC (tmux a -t Devices:ADC) with a printed error but the “alive…” is shown so the script recovered from the error. If the script has a stuck “alive…” (ie. number of “.” is constant), or is simply dead, then you’ll want to restart it.

```
tele:tmux:/home/nfiudev> kpicctrl_ADC
INFO:ADC:No duplicate control script, continuing with initialization
INFO:ADC:Shms connected successfully.
ERROR:ADC:Error in set_closed_loop:
        |
        |Control script error: invalid literal for int() with base 10: '#2'
        |Traceback (most recent call last):
        |  File "/home/nfiudev/kroot/rel/default/bin/kpicctrl_ADC", line 77, in set_closed_loop
        |    if req_state != int(self.dev.getLoopState([axis])[axis]):
        |ValueError: invalid literal for int() with base 10: '#2'
        |

alive...
[Devices] 0:Light_Src  1:PyWFS_Pickoff  2:Coronagraph  3:FAM  4:Filter_Wh  5:Mode_Change  6:Track_
```

NOTE: the Track_Cam script does not have an “alive” print; see the CRED2 Freezes section below for how to check the health of the tracking camera.

Detaching from a tmux session:

When you attach to a tmux session (using `tmux a -t <session_name>`), you enter the terminal environment where that daemon/script is running. This is reflected in the fact that the terminal has a green bar at the bottom stating what tmux session you are in.

➔ **To detach from a tmux session**, hit `Ctrl+b` then `d` (ie. press and hold the “control” key, tap the “b” key, then let go of “control” and tap “d”)

CRED2 freezes:

What it looks like:

Either the GUI image has the placeholder image (a grey image with a picture of a camera that says no image available), or the GUI image isn't updating (the peak flux on the top right of the image is static). To confirm that this is indeed the issue, try (in a kpython3 instance):

```
from Track_Cam_cmds import TC_cmds
tc = TC_cmds()
tc.Img.get_counter()
tc.Img.get_counter()
```

If the two get_counter() calls give the same number, the shm is indeed not updating and the cred2 may be frozen. If these two counter calls do indeed increase/change but the GUI image is frozen, then the camera feed is live and the issue is probably the Viewer Processing script. Follow the entry later in this document for Viewer Processing Script is Frozen instead of the steps in this section/entry.

Note: this can come up if the FPS is set to < 2. The GUI will not allow this but the python-level code does not have any safety software limits on FPS.

How to fix:

1. Kill the cred2 control script: in a normal terminal type `tmux a –t Devices:Track_Cam` then send a ctrl-c
    a. **Note**: when the control script dies, its tmux will also close and the next device's tmux will open; **be patient and check if you are still in the cred2 tmux BEFORE sending any further ctrl-c** to avoid killing other device's control scripts.
    b. To detach from (ie. cleanly exit) the tmux window, use ctrl-b, then d. (**more details here**)
    c. *If* ctrl-c doesn't kill the camera control script, then in a separate terminal run `ps -aux | grep kpicctrl_Track_Cam,` this will return multiple processes; note the pid (second column) of the line that directly matches "kpicctrl_Track_Cam". Use `kill -9 <pid>` to kill it. Since we had to hard-kill the daemon (using -9), you'll need to clear the Tracking Camera's DSTAT shm by doing: `rm $RELDIR/var/kpic/shms/Track_Cam/DSTAT.shm`
2. Now restart the track cam control script
    a. In a **new** kpython3 session:
    ```
    from Track_Cam_cmds import TC_cmds
    tc = TC_cmds()
    tc.activate_control_script()
    tc.connect_camera()
    ```
    Note: you must use a new kpython3 session if you cleared the DSTAT shm. Otherwise you'll get a missing shm error (because your old kpython3 session points to the old shm)
    b. Assuming that the last command returns, you should now have live images

If that doesn't work, repeat the above but force a frame grabber reset between Steps 1 and 2 by:

- Use `/opt/EDTpdv/initcam -c 0 -f /nfiudata/OLD/nsfiu/dev/config/cred2_FGSetup_16bit.cfg` and then `/opt/EDTpdv/take -c 0 -l 10`. You should see "10 images 0 timeouts 0 overruns" as pictured below:

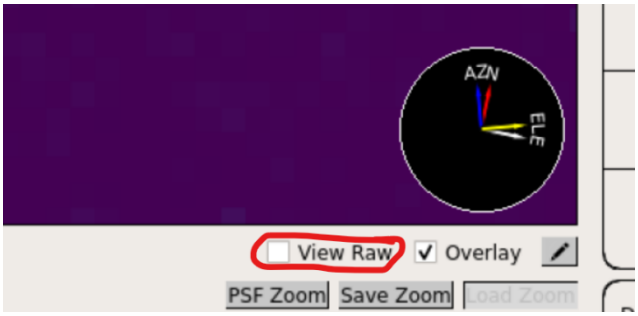
```
[nfiudev@nfiuserver ui_files]$ /opt/EDTpdv/take -c 0 -l 10
reading image from FirstLightImaging C-RED 2 640x512 (4-tap, freerun)
width 640 height 512 depth 16 hdr 32 total bytes 655360 (DMA 655392)
10 images 0 timeouts 0 overruns
21674528.725121 bytes/sec
33.071091 frames/sec
```

- After forcing an FG reset, retry Step 2 above
  NOTE: if you see an error about buffer not being configured, it probably means there's a camera control script still running. You can check for this script using the ps -aux command noted in the last bullet point of Step 1.

Viewer Processing Script is Frozen:

What it looks like:

The camera image displayed on the Viewer is either stuck on a still frame (not updating) or it shows a grey box with "no image available" (see screenshot below). You can confirm that the issue is with the processing script rather than something else by selecting the "View Raw" option under the GUI image (see screenshot below); if the image starts to update when you look at "View Raw", then the issue is probably with the processing script. If it doesn't start updating with "View Raw", then it is not a Processing script issue. In the latter (not a processing script issue), try the CRED2 Freezes section.



How to fix:

The processing script can usually be easily fixed by killing then restarting it.

1. Connect to the Processing tmux: in a regular terminal, type tmux a -t Processing
2. Kill the script by sending ctrl-c to it. *If* that doesn't work, you can hard-kill it with a kill -9 call:
   a. Run ps -aux | grep kpicctrl_Track_Cam_vis_process and note the pid in the second column.
   b. Then run kill -9 <pid> .
3. Clear the shared memory for it:
   a. In a normal terminal, go to: /home/nfiudev/kroot/var/kpic/shms/Vis_Proc/
   b. Clear the status shared memory: rm PROCSTAT.shm
4. Restart the script.
   a. Start kpython3 by typing kpython3 in any terminal. In that kpython3 instance run:
      i. from Track_Cam_process import TC_process
      ii. proc = TC_process()
      iii. proc.activate_control_script()
5. You should now have a live, processed, image on the GUI (assuming you de-select "View Raw").
6. If you still do not have a live, processed, image, try the entry in CRED2 Freezes.

A KPIC Stage is acting up significantly:

What it looks like:

Sometimes a KPIC stage will seem to be frozen such that when you try to move it, it doesn't do anything or it hangs in a python terminal. Other times the control script will crash/die so the stage can't be commanded. Other times you'll do a <STAGE> status -push and it says strange things or you'll try to do a <STAGE> enable and it says the control script is already running but you know for a fact it's not.

Basically: if a stage is acting up a lot, you can try the following procedure to recover it and it will usually recover the stage control.

How to fix:

1. **** for the procedure below, let's pretend the SAM is acting up but this applies to all devices** It also applies to most control scripts in general though not all will have the CLI tool to try status -push.**
2. (The underlying issue here is usually that the STATUSD shm did not clean up correctly so the procedure below takes care of that)
3. First, check if the lowest level code knows what's going on:
    a. In a normal terminal, run SAM status -push and see if it prints the current position of the stage with a very recent time stamp. If it hangs, ctrl-c and proceed to (3b). If it works, then the issue is not related to the STATUSD shm, and you should try a different debugging entry in this doc (ie. do not proceed with the remaining steps in this entry).
    b. Check if the tmux is running with the script inside of it: tmux a -t Devices:SAM. If it fails to connect or connects and shows that the script crashed, then try the next step (3c). If it shows a seemingly healthy control script, type ctrl-c in the tmux and then proceed to (4). If the ctrl-c in the tmux hangs or seems to not kill the script, detach from the tmux session by typing ctrl-b then d (more details available here on detaching a tmux) and then proceed to step (3c).
    c. Check if the control script is still running by typing ps -aux | grep kpicctrl_SAM into a normal nfiuserver terminal. If this says no control script is running, continue to (4). If it shows the script running, note the pid in the second column, and then use kill <pid> and proceed to (4). If that still doesn't kill the script, use kill -9 <pid> and proceed to (4).
4. Try starting the control script directly: SAM enable. If it says that the script is already running but you confirmed in the previous step that it isn't then we know STATUSD is definitely the issue.
5. At this point, we've confirmed that STATUSD is the issue. So let's clear the STATUSD shm.
    a. Go to the directory with the shms: cd $KROOT/var/kpic/shms/<script> (in this SAM case, it would be cd $KROOT/var/kpic/shms/SAM)
    b. First delete *only* the STATUSD shared memory: rm STATUSD.shm.
        i. For SAM issues, also need to rm ROTSTATUSD.shm
        ii. For PSM issues, also need to rm FOCSTATUSD.shm
6. Now restart the script following the appropriate procedure from the Startup + Shutdown.docx.
    a. For example, for the SAM we can do: SAM enable
7. Note: If this procedure didn't work, or if you want more details about the shms, please refer to the shared memory document (IPC layer (shms).docx) which has many more details and some debugging tips which include clearing the semaphores.

Recovery after unexpected server reboot/crash:

What it looks like:

If the server unexpectedly reboots or crashes, the control scripts probably won't get properly cleaned up. *Server crashes due to KPIC software are relatively rare.* The main source of crashes was a memory leak in the Viewer, but this leak was fixed in May 2024 such that crashes should be even rarer now. If a crash happens, some of the more obvious symptoms are: vnc sessions won't be running so you won't be able to log into them, tmux sessions will not be running (running `tmux ls` in a normal terminal says "no server running[…]", etc.). You can confirm that the system did indeed go down using the `uptime` command or `last | grep reboot`.

On a similar note, the KPIC code uses shared memories for ipc and to track the state of the stages/control scripts. These try to capture many of the OS signals sent during a shutdown/reboot, but sometimes this capture does not work for all control scripts. As such, it is *very strongly recommended* to run `KPIC_shutdown` before a reboot or power cycle. If this is not done, some shms may not clean up fully.

The key point is: if the server reboots/crashes you'll likely notice it, and if the control scripts were running at the time, they probably won't start up easily.

How to fix:

1. First, check that no instances of the KPIC dispatcher (`kpicd`) is running. This can be done from a normal nfiuserver terminal by running `ps –aux | grep kpicd` in any terminal.
    a. If this shows no dispatcher instances, proceed to step (2) below.
    b. If this shows that an instance is running, you should kill that instance (see c and d below for details on how to kill).
    c. If that instance was started by the KPIC software, it would be in the "keywords" tmux. In that case, you can kill it by connecting to that tmux (`tmux a –t keywords` from any terminal) and sending `ctrl-c`. If the `ctrl-c` in the tmux hangs or seems to not kill the script, detach from the tmux session by typing `ctrl-b` then `d` (more details available here on detaching a tmux) and then proceed to step (1d).
    d. If `ctrl-c` does not work in the tmux, or if a `kpicd` instance exists outside of the tmux, run `ps –aux | grep kpicd` again and look at the second column to get the pid. Then use `kill <pid>` to kill the instance. If that doesn't kill it, you can use `kill –9 <pid>`.
2. Now that any spurious `kpicd` instances are dead, call `KPIC_init` from a terminal to start up all the KPIC software. The default options (ie. using no flags) should be fine.
    a. Note: if you see some errors, don't panic. Some errors are minor and easily fixed (see the "KPIC Stage is acting up significantly" section to see how minor ones can be handled)
3. If that fails, run `KPIC_shutdown -rmsems -V`. Since this has the -rmsems flag, it will clean up any stale shared memories or semaphores that may be leftover from the earlier crash/reboot.
4. Now try restarting the software again using `KPIC_init`.
5. At this point, this should have successfully restarted everything, because we've cleared all pertinent semaphores, locks, and shared memories such that the control scripts should be starting from a clean slate. If it seems like it didn't work, it's okay to try a cycle or two more of the `KPIC_shutdown -rmsems -V` followed by `KPIC_init`. This is unlikely to be needed but it won't hurt to give it another try.

6. If one or two stages/control scripts don't start up, but the bulk of the software does, just independently and explicitly start the ones that didn't work. The "KPIC Stage is acting up significantly" section can be helpful here.
7. You can confirm that everything started correctly by doing some of the tests listed in Detailed System Health Check above.

- **NOTE 1: you may see an error about "cannot connect to traffic".** It is unclear what is causing this but it does not seem to be critical to KPIC software.
    - First, make sure that no kpicd is running before you try a KPIC_init.
        - You can check by repeating Step (1) in this troubleshooting entry.
    - Then, when you get that message, ignore it and continue with the usual startup procedure/the debugging procedure listed above, ignoring when the message comes up.
    - At the end, you'll likely see that the system mostly started except for the tracking camera. Treat this as independent issue with the tracking camera (see other troubleshooting entry regarding cred2 freezing).

Force-killing the dispatcher:

What it looks like:

Trying to ctrl-c the dispatcher (from inside its tmux session) does nothing such that you can enter ctrl-c many times in a row and that session just hangs (the command keeps running and doesn't return you to the command-line).

How to fix:

1. (There's a couple of places from which this can be done but I find it easiest to do it all from the same terminal so that's how I do it in the steps below)
2. Connect to the tmux session for the dispatcher: type `tmux a –t keywords` into a normal terminal
3. Try to `ctrl-c` one last time, just in case it works. You'll know it worked if the `kpicd` script ends and returns you to a command-line input where you can type commands into the terminal. If it's still hanging, then it did not work.
4. Zombify the dispatcher using `ctrl-z`. This should return you to the command-line such that you can now type terminal commands again.
5. Set the dispatcher to run in the background using `bg`.
6. Get the pid for the dispatcher using `ps –aux | grep kpicd`. Find the dispatcher command as the entry that has `kpicd` followed by some flags (not the grep command you just used). Note the pid of this script as the second column in the output (should be an integer).
7. Try killing the dispatcher kindly using `kill <pid>` where you replace `<pid>` with the integer from the last step. You'll know this worked because doing the `ps –aux | grep kpicd` command again no longer shows that same `kpicd` instance (odds are that this doesn't work though since the dispatcher doesn't play nice when it's acting up so don't worry if nothing seems to have happened; just go to the next step)
    a. If that worked, you're done troubleshooting! The dispatcher is now dead as you wanted it to be originally. You can now type `exit` into the tmux session itself and then you can stop following the steps below now.
8. If that didn't work, try killing the dispatcher in a more forceful way using `kill –9 <pid>` where you, again, replace `<pid>` with the integer from the previous step. You'll know this worked if the ps –aux command now doesn't return the `kpicd` instance that it did before.
    a. If that worked, you're done troubleshooting! The dispatcher is now dead as you wanted it to be originally. You can now type `exit` into the tmux session itself.
    b. I have no other steps since that should really have worked... if it didn't, we'll need to figure out what's going on.

Track Cam Control won't start/processing script won't work after cropping change:

What it looks like:

This has appeared in 2 ways so far. The first is fairly harmless, the processing just fails to find a valid background to use so it reports this error though freezes. This is the more common appearance of the issue and is a bug I'm working to fix.

The second way it appears is more complicated and has only happened once so far. After changing the cropping, the python interface seemed to say that the change was successful but the Viewer was frozen and I got seg faults in python if I tried to query the image counter (tc.Img.get_counter). Trying to restart the camera control script kpicctrl_Track_Cam manually also led to seg faults and core dumps. Even after fixing the tracking camera part of things (ie. I could see fresh images in the "raw" mode of the viewer), the processing script seemed to be failing to run and was reporting an error about not finding a good background to use.

How to fix:

The first case (valid images but no processing) is relatively easy to fix. Inside a kpython3 instance, type:

- from Track_Cam_cmds import TC_cmds
  tc = TC_cmds()
- Now, confirm camera images are coming in correctly using tc.Img.get_counter(). Two consecutive calls should show the counter incrementing. If it's not, skip to the block of bullets for case 2.
- Now just take a new background for the processing script using (in kpython3): from Track_Cam_process import TC_process; proc=TC_process(); proc.save_dark(). This should fix the processing issue. You *may* need to manually restart the processing script as in: Startup + Shutdown.docx.
  - e.g. proc.activate_control_script()

For the second way this issue appears, I tried a ton of stuff to fix it so it's unclear exactly what did it but below are the steps that I think were critical/that seemed to make a difference.

- First checked if camera was working correctly using the /opt/EDTpdv/initcam and /opt/EDTpdv/take commands. (see second part of "CRED2 freezes" for more details)
- Roughly followed the entry for Recovery after unexpected server reboot/crash to shutdown all control scripts, including using the -rmsems flag and then restarting the control scripts with the KPIC_init -DLCHMV call. I had to do this a couple times using different flags (including the -f flag to clear STATD shms automatically) so I'm not certain what combination fixed the tracking camera. However, this was basically all that was needed to get the camera reading out again.
- With the camera reading out using our code and all our control scripts running, try the procedure in the previous block of bullets above ("Now just take a new background for the […]")
- If that doesn't work, try clearing the sems and shms for the processing script's PROCSTAT and PROCIMG elements. For how to do this, refer to the Control Scripts.docx document, towards the bottom where Tobias describes how to clear semaphores and locks for a single device (he uses

the FAM in the example). Refer to the Startup + Shutdown.docx document to see how to start just the processing script without having to shutdown/restart the full system.

- Then once the processing script is up and running normally again, try taking a bias as described two steps above and that should (hopefully) work.

Tracking and Processing fail after setting too-large Astrometry/Offsets:

What it looks like:

This happened while I was doing some offsetting tests. I was increasing the offset using the "Sep (mas)" option in the "Tracking" tab of the Viewer. I accidentally set an offset that would have put the PSF out of the CRED2 image. The tracking script and processing scripts then crashed. I was unable to restart them using the usual methods (ie. .activate_control_script() function from their python libraries as well as direct calls to the control scripts from terminal).

When doing the latter (direct calls in terminal), I was able to see that the scripts were throwing an error regarding nans in the fits headers. This was because some of the tracking shms, and their corresponding keywords, were set to nan when I tried to set the large offset.

How to fix:

(NOTE: a software fix was implemented to address the source of this issue, so it should not come up again, but the troubleshooting entry remains as a reference just in case)

- Connect to the tracking shms from kpython3:
  - kpython3 in any terminal, then in that python session:
    from Star_Tracker_cmds import Tracking_cmds; track = Tracking_cmds()
- Check the Astro and Distorted Goal shms:
  - track.Astro.get_data() and track.Goal_dist.get_data()
- Set any nans and large numbers (e.g. 700) in these shms to smaller numbers like 1.0. (data type must match). Don't touch the other values that are good.
  - res = track.Astro.get_data()
  - res[np.isnan(res)] = 1.0
  - res[res>50.0] = 1.0
  - track.Astro.set_data(res)
  - (repeat with Goal_dist shm)
- Restart the tracking script (which will in turn update the keywords used by the fits headers)
  - track.activate_control_script()
- This should fix the issue. Restart the processing script if needed following the procedure described in the Startup + Shutdown.docx document.

Micronix (SAM Rotator and PyWFS Filter Wheel) fails to communicate:

What it looks like:

This came up after Keck IT did penetration testing on the terminal server that hosts the KPIC stages. The testing affected multiple other instruments in similar ways as well and our internal logs show that the micronix started logging issues with time tags immediately after the testing was started so we are almost certain that the error was due to the penetration testing.

The issue presented itself initially as a failure to connect to the SAM rotator or the PyWFS_Filter. Sometimes, the filter would be able to connect but the tmux session with the control script for it would display errors constantly. Deleting the status shms and trying again did not fix things. Also, manually starting the control scripts did not work; the script would seem to start but would then hang and if you ctrl-c'd it, it would show that it was stuck instantiating the Micronix_Device connection.

To further debug, we killed all KPIC control software (using KPIC_shutdown) and then connected to the device from the python interface (See Micronix.docx) using the Micronix_Device.py library. From this layer, we are bypassing all shm and are communicating with the controllers directly at the lowest level. However, when querying the position (using dev._query("1POS?")) or querying any other parameters ("1LCG?", "2DBD?", etc.), we would sometimes get a response and sometimes not. When we did get a response, they would sometimes make sense (for example, 121.0000 for 1POS?) but other times they would clearly be responses for other queries (for example, 299.000 for 1LCG?). When we didn't get a response, we'd get a timeout error. We'd seen this in the lab before when multiple connections were open to the controller at the same time such that responses to one query from one script were being intercepted by another script.

TLDR: this shows up as garbled/confused/missing/unreliable communication with the micronix controller even at the lowest level.

How to fix:

- First, make sure that the issue is truly with the micronix controller communications and not just the shms. That means trying to clear the STATUSD shms and restarting the control script. If the issue is definitely not with the shms and is definitely with the communications, then you can proceed to the below steps.
- Kill all KPIC software using KPIC_shutdown -rmsems –V .
- Reboot the nfiuserver. If possible, ask Keck software on call to reboot the terminal server at IP = 192.168.108.54 - might not be necessary but Paul R. (Keck) suggested we do it just to be safe.
- Reboot the micronix controller:
    - Note that rebooting the controller will require it to re-home on power-up. This should be fine almost all of the time but if the stage was manually moved (or commanded at a very low level) to a position between 0 and the negative hard stop, then this could cause issues. See the Micronix.docx document for details. This will generally not be a concern but is worth noting and keeping in mind since if it does come up, trying to home it will cause it to collide with our safety hard-stop which could in turn damage the stage.
    - To reboot the controller, make sure all connection to it are closed and then power cycle it using the pdu_gui software that can be started from any terminal on the nfiuserver.
- Once all devices (nfiuserver, terminal server, and micronix) are powered back up, follow the normal start procedure for KPIC and the micronix should no longer be an issue.

KPIC Keywords not updating in the NIRSPEC header:

What it looks like:

This mostly happens when there are power-outages at Keck, when servers are rebooted, or when the server crashes. It is identifiable in the NIRSPEC fits files because the KPIC keywords (for example FAMX or FIUDELX) don't change between frames. You can explicitly test for it by taking a NIRSPEC frame, moving a stage (for example the coronagraph) to a new position, then taking another NIRSPEC frame – the relevant keyword should reflect the change.

How to fix:

Greg Doppman has fixed this for us in the past. Here are the instructions he provided for us. He said that it is okay for us to run this fix ourselves if the KPIC keywords are not updating on the NIRSPEC headers correctly. If you have questions about this procedure, are uncertain about whether you should do it, or run into an issue, reach out to Greg (available in the Keck Slack, Mawet Group Slack, or via email).

- From an xterm as nspeceng@nirspecserver, run:
  - nirspec stop! nsheaders
  - nirspec start nsheaders
- After it restarts, you can check that it's running here:
  - [nspeceng@nirspecserver ~]$ nirspec status nsheaders
  - Output should be:

    nsheaders : is running on nirspecserver.
    nsheaders : PPID PID PGID UID VSZ RSS %CPU TIME COMMAND
    nsheaders : 1 23902 23694 2121 4927040 117316 10.2 00:01:22 /kroot/rel/default/bin/kpython /kroot/rel/default/bin/keygrabber -c /kroot/rel/default/data/nsheaders/keyheaderd.conf -y /kroot/rel/default/data/nsheaders/keyheader.conf

kpic_sfp_source returns error about incorrect number of arguments

This occurs when the SFP is in a faulted state. Contact an SA and ask them to check the SFP.

BMC DM Humidity Interlock Switch Not Connecting

This is a very rare occurrence and has only happened once - after some telescope-wide network issues at the summit. However, just in case it comes up again, here are some notes on it.

What it looks like:

The most likely symptom to be noticed is that the HIL ioc will not be able communicate with the HIL. This means that the kpic_pystage_gui will report strange numbers (-999 for humidity) and trying to read any EPICS channels from the HIL (k2:ao:kpic:hil:connected or k2:ao:kpic:hil:humidity) will fail. Connecting to the HIL tmux session (tmux a -t HIL) will also show several "CRITICAL" errors suggesting communication issues. Finally, the web browser for the HIL will fail to connect (ie. will just hang when trying to load the web page). To test the web browser connection, open Firefox from nfiuserver, and enter "10.136.1.100" into the url (or just select the "KPIC BMC DM HIL" bookmark if it's available). A proper web browser connection will show a web page with "iServer3G LOGIN" and a prompt for a password.

How to fix:

1. First, make sure this is truly a full-on communication issue and not just that the comms got out of sync:
    a. Kill the kpicctrl_HIL_ioc script. This is done by connecting to the tmux session (type tmux a -t HIL into a terminal) and then sending a ctrl-c. Close the tmux session by typing exit, if it doesn't close automatically.
    b. Now restart the just the HIL script with KPIC_init -KV in a terminal. Give it a moment to start, then check if things are working correctly (see "What it looks like" above, or check the tmux session to see if the script is raising errors). If it is working, you are done.
2. If the HIL is still not connecting/responding, we'll need to reboot the HIL controller:
    a. Kill the kpicctrl_HIL_ioc script again (see 1a above).
    b. Turn off the HIL using NPS 3 Outlet 1. This can be done from the pdu_gui, in which case you should select "k2aopower" from the dropdown at the top, then the Outlet is called "KPIC BMC DM+Humidity Interlock". Or it can be done from keywords directly using modify -s k2aopower OUTLET_HE1=Off in a terminal.
    c. Wait 30 seconds. Note: A lengthy pause is crucial here. A shorter pause does not seem to solve this issue. So make sure to wait 30s or more.
    d. Turn on the HIL (see 2b but turn it on instead). Wait a moment (30s or so) for it to fully boot back up.
    e. (Optional: Check if the web browser is working – see the "What it looks like" section above for instruction on how to access the web browser. If the web browser doesn't work, then proceed straight to step 3 below.)
    f. Restart the HIL control script (KPIC_init -KV) and check if it is working. If so, you are done.
3. If the HIL is still not working, you'll need to power cycle the PyWFS network switch:
    a. Kill the kpicctrl_HIL_ioc script again (see 1a above).
    b. Power cycle the switch (speak to SWOC to do this)
    c. Then Try the steps in section 2.
    d. This is the end of this troubleshooting entry; the above steps worked when the issue happened before. If this doesn't work, reach out to the KPIC team for more debugging ideas.

A Conex Stage (SAM X for example) is Not Responding:

This is a very rare occurrence that has only happened once – after a power cycle of a switch that handles the comms for many of our stages. Based on what we saw, and the solution for this issue, it seems very likely that the SAM X lost connection right in the middle of a message to/from the server. This seems to have left the stage comms in a bad state where it couldn't communicate. If this is indeed the source of the issue, then it should be extremely rare since the chances of that happening are fairly slim (our daemons only query the affected stages on a 60s cadence and each query is very short). This is supported by the fact that we've never seen it come up in the past.

However, if the timing of a network issue is precisely right, it could come up again. This affected the SAM X stage this time, but it's possible it would affect any of our Conex stages. As such, this entry holds for failure-to-connect issues with any of our Conex stages (listed below). Now that we know the issue, the fix is very straightforward and could be executed fairly quickly (<10 min). KPIC Conex stages:

- SAM X and Y (Conex TRB6CC)
- PSM Focus 1, 2, or 3 (Conex TRB6CC)
- Light Source Retractor (Conex AG-LS25)
- Mode Change – Pupil/Focus stage (Conex AG-LS25)
- PyWFS Pickoff (Conex AG-100P)
- Track Cam Filter Wheel (Conex PR-100P)

What it looks like:

For the SAM X, this presented itself when restarting the KPIC software after the switch was power cycled. In this case, I knew something was up since the startup script reported issues connecting to the stage. More generally, if this issue occurs, it would likely be identified any time you went to interact with the stage. For example: a move request would show that the stage didn't move or would return an error; if the GUI is open, it's likely that the GUI would have had the SAM in red with a note that it wasn't connected; if you check the tmux session for the given stage, you'll see that the comms script is *actively* reporting connection issues. (I say *actively* because our scripts implement features to try to reconnect automatically, so a tmux might have an old connection issue reported but then will show that the stage is fine by having the "alive…" updating live with a varying number of periods "." – In that case, the stage has recovered. If it is still actively reporting a connection issue, ie. printing new errors, then it didn't reconnect).

A key feature of this bug is that a simple software restart and reconnect *will not* work. That is, if you follow the software shutdown and restart procedure above (Overview Section) it does not work. This issue has the stage in a strange state where the comms are seemingly frozen or unable to respond, so just restarting software will not help and instead a power cycle (described below) is needed. **Similarly, clearing the STATUSD shm**, as described above (Stage Acting Up Significantly Section), ***will not* work,** again because this is a comms issue in the stage controller itself, not in our software.

How to fix:

0. NOTE: The solution for this issue boils down to power cycling the stage so that the comms in the device are reset. However, there is a power leach in the conex power lines such that just turning off the outlet for the given stage (ex: NP1 Port 6 for the SAM XY) does not resolve the issue. This

powers off the motors in the stage, but does not seem to kill the comms. We therefore need to do a full power cycle of the system, as described below.

1. First, make sure this is indeed a comms synchronization issue as described above by first trying a full KPIC software restart (Overview Section) and then trying to clear the STATUSD shm (Stage Acting Up Significantly Section). If none of those works, proceed to the subsequent steps below.
2. Turn off all KPIC software so that we can safely power everything off. In a terminal, run:
   `KPIC_shutdown -rmsems -V`
3. Turn off all outlets for KPIC.:
   a. NOTE: we will power cycle the nfiuserver in step 4, so you will need someone with either physical access to the server or a Keck software person (SWOC) with root access. Similarly, once we power things off, stages will need to be re-homed, which can be an issue if you are relying on the current SAM or SAM rotator positions for observing. **Consider that before proceeding, in case having the given conex stage running is not as critical as having the rest of the instrument (ie. the track cam) running right now.**
   b. Run `pdu_gui` in a terminal. Then select "k2aopower" from the dropdown next to "Area"
   c. Run down the list powering off devices on KPIC NPS 1, KPIC NPS2 and KPIC NPS 3. The power-off sequence does not matter; I just went down the list from the top on the GUI.
   d. ** For the two Adnaco ports (KPIC NPS 2 Port 8 and KPIC NPS 3 Port 3), you will need to remove the software lock. There is a lock on these because powering these off means you have to reboot the server. So, we locked them to prevent unwanted power cycles.
      i. Remove the locks by running the following commands in a terminal:
         `modify -s k2aopower OUTLET_HC8_LOCK=Unlock`
         `modify -s k2aopower OUTLET_HE3_LOCK=Unlock`
      ii. Then turn these off (can use the regular PDU GUI buttons now)
4. Wait a moment for everything to finish powering off (~2 minutes).
5. Turn on all KPIC devices again:
   a. Run through the pdu_gui outlets again, turning on all the outlets you turned off.
   b. **Please reset the Adnaco outlet locks!** This will make sure people don't accidentally turn them off in the future:
      `modify -s k2aopower OUTLET_HC8_LOCK=Lock`
      `modify -s k2aopower OUTLET_HE3_LOCK=Lock`
   c. Unless you specifically want the KPIC lights on, you should probably leave them off (especially if they were off originally). The lights are:
      KPIC NPS 1 OUTLET 3, KPIC NPS 3 OUTLET 4, KPIC NPS 3 OUTLET 6
   d. Similarly, you can leave the Flip Mount + PD off unless you are actively using it:
      NPS 2 OUTLET 7
6. Power cycle the nfiuserver (this is needed since the Adnaco's were turned off). You can either:
   a. Ask someone to physically press the power button on the front of the server. This is located in a server rack at the summit in the K2 Computer Room.
   b. Or, ask a Keck software person (SWOC) to reboot it.
7. With everything back on, just run `KPIC_init` in a terminal to restart the KPIC software.
8. Home/Reference any stages that need it. You can do this from the KPIC GUI (Viewer) easily by running down the devices tab and clicking on the "more" button for stages that have it red.
9. This should have fixed your issue. If it did not, reach out to the KPIC team.

Anomalously low flux on NIRSPEC while observing after AO setup switch

This is usually due to the K2AO "DFB" mirror not being the right position. Thus, **check that DFB is in 'mirror' first**. You can check the DFB from the k2aoserver-new "SC Gui".

If you still have issues after ensuring DFB='mirror', then double-check the position of all other AO stages and then the position of all KPIC stages. For KPIC stages, you can compare against the list provided in Appendix 0 of the KPIC Facility Observing instructions.

Visualizer Processing Remains Frozen (unrelated to cropping)

What this looks like:

The tmux for the processing script seems to show that the script is alive and well (the "…" is updating correctly and no errors appear in the display). However, the processed image in the Viewer remains stale/bad and the counter for the Vis_Proc image shared memory does not increment despite the fact that the raw track cam image does increment and is live.

** Basically, this issue is for when the Visualizer Processing image goes stale and won't refresh despite seeming okay at a script level and despite multiple attempts to shutdown with (-rmsems).

How to fix:

** FIRST TRY THE USUAL SOLUTIONS:

- Try restarting the software a few times using KPIC_shutdown -rmsems -V and KPIC_init
- Try clearing the PROCSTAT.shm and restarting the processing script on its own (see the Viewer Processing Script is Frozen troubleshooting entry earlier in this doc)
- Try taking a new background for the CRED2:
    - Use the save dark button on the GUI
    - If that doesn't fix it, save a background separately from a kpython3 session: type kpython3 into a normal terminal, then run
      from Track_Cam_process import TC_process; proc=TC_process(); proc.save_dark()

** Only if that doesn't work, follow the steps below.

The final solution (the one time this has happened) was to clear the semaphores *and* **lock** files for the visualizer processing script. We hardly ever deal with these files explicitly because KPIC_shutdown -rmsems is supposed to clear them. However, for some reason, it seems like the -rmsems flag does not iterate through all the lock files of the image processing script. This is a bug that will be fixed at some point but in the meantime, you need to clear the files manually. Refer to the Control Scripts.docx document for instructions on how to clear the semaphore and lock files correctly. Towards the bottom of that file, Tobias describes the procedure for a single device (he uses the FAM in the example).

** NOTE: **other than the PROCSTAT shared memory, do NOT rm the any of the other shared memory (.shm) files for the processing script**. This should not be necessary and actually when the script tries to make new ones, it seems to have an issue. That is a minor bug that will be fixed eventually but has only come up a single time, so is very low priority. You can clear the .sem and _lock files without issue though.

KPIC BMC DM Not Moving Despite RTC Seeming Okay

What this looks like:

When you command the DM to change shape, it doesn't appear to change. For example, if you try `from DM_newRTC import DM as dmlib; DM = dmlib(); flat = DM.setFlatSurf(); DM.pokeZernike(0.2, 4, bias=flat)` in a `kpython3` session to apply some defocus, you should normally see defocus on the CRED2, but if the PSF doesn't seem to change, then the DM likely isn't responding.

This troubleshooting entry is only for when the issue is with the K2 RTC, NOT when it's due to the HIL. Thus, you should double check that the HIL hasn't triggered into alarm and that the DM is indeed powered. You should also check that the RTC is in the right mode for BMC DM control (ie. in 'NORMAL'). If neither of these are the source of your issue, then proceed with this troubleshooting entry.

How to fix:

*** FIRST TRY THE USUAL SOLUTIONS:

- Check that the HIL isn't triggered. If it is, reset the HIL alarm. You can check and reset the HIL alarm by running `kpic_pystage_gui` in a terminal and then hitting the "Reset" button. Similarly, check that the DM is powered (ie. that the HIL is on at the outlet NPS3 Port 1 (OUTLET_HE1))
- Check that the RTC is in normal. You can connect to the k2aoserver (`ssh -X k2obsao@k2aoserver-new`) and open the wavefront controller GUI (`k2wfc.dsp`). From there, you should see that it is in 'NORMAL'; if it is not, you can move it slowly into normal (following intermediate steps like LOADED and STANDBY if needed).

*** Only if that doesn't work, follow the steps below.

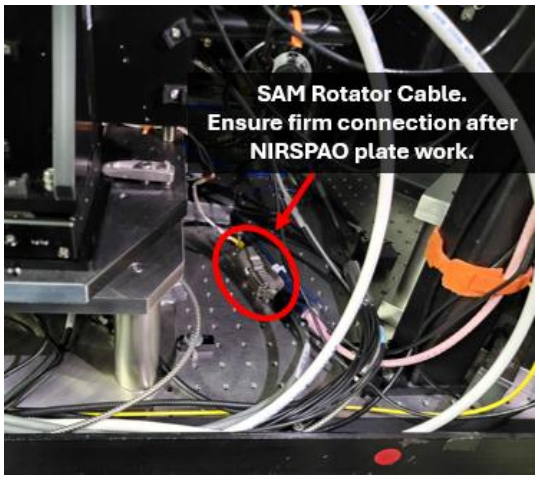
On September 24, 2024, Percy and Dan had this issue for the first time. Percy had to do a full restart of the RTC, and that fixed the issue. However, Percy says that this solution is heavy-handed for an issue like this, and there is likely a better solution. As such, Sylvain has suggested a less-invasive method. Here is a summary of Sylvain's suggestion as rehashed from an email:

1) Restart the RTC pipeline:
   a) As **k2obsao@k2aoserver-new**, in a terminal: `rtcRestartPipeline`
2) If still not working, power cycle the IM:
   a) As **k2obsao@k2aoserver-new**, in a terminal:
      i) `modify -s ao wcstat=LOADED`
      ii) `k2rtcPowerCycleIM.csh`
      iii) Wait about 30 seconds
      iv) `modify -s ao wcstat=NORMAL`
3) If still not working, fully reboot the RTC:
   a) As **k2obsao@k2aoserver-new**, in a terminal: `k2rtcReboot.csh`
      i) (This can take several minutes)

SAM Rotator Encoder Not Found

What this looks like:

This is a physical connection (cabling) issue that has only come up once, after a NIRSPAO plate removal and reconnect. The underlying issue was that the SAM rotator cable was loose at the connection point between the 1m cable coming out of the stage and the 10m extender going into the E-Vault. This connection is made between to DB headers in the AO bench near the NIRSPAO plate (see picture below).



The fix is ultimately to tighten the connection between these two DB headers. However, this requires a person physically present at the summit, and this issue has only come up once, so it is a very rare occurrence. As such, the steps below are primarily debugging steps to ensure this is the issue.

Also note that this is an issue with the SAM *rotator* not the SAM *X/Y* stages. It's important to note this difference since a *rotator hardware* issue can not be fixed by debugging the *X/Y* stages, nor vice-versa. The SAM daemon is a single daemon though, with two threads, so *software* issues with one element can affect the other. When it's a software issue with the SAM, restarting the KPIC software (see the first troubleshooting entry above) is a good start, then check the shared memories (see the "stage is acting up significantly" entry).

How to fix:

First identify that the issue is with the physical cable connection.

The main symptom here is that the stage reports it can't connect to the encoder, but you can still query its position and other elements that don't require the encoder.

1. Since the encoder isn't identified by the controller, trying to close the loops on the stage will fail (you'll try to close the loops but the stage will still report that they are open). Thus, try to close the loops on the stage using the normal KPIC software (either on the GUI or from a terminal by doing SAM rot loop closed then querying the loop state with SAM rot loop). If the loops close correctly, the issue is not with the encoder connection, so **see a different troubleshooting entry**.

2. Once you know the loops are failing, try querying the stage at a lower level. To do this, first disconnect the software daemon from the hardware: SAM disable.
3. Now connect to it directly from the low-level library:
    a. Start a kpython3 session: run kpython3 in any nfiuserver terminal
    b. In that **kpython3 session**, connect to the stage:
    ```
    from Micronix import Micronix_Device
    dev = Micronix_Device()
    dev.open_Telnet('192.168.108.54', 2027)
    ```
    c. Check that you can talk to the stage by querying its position:
    dev.getPos(1) should return {1: '<a number>'}
4. Query the current loop state:
    dev.getLoopState(1) should return {1: '<an integer: 1 or 2 or 3>'}
5. Now try toggling the loop state:
    dev.setLoopState({1:0}) followed by dev.setLoopState({1:3})
6. Note that Check for errors:
    dev.getError(1)
    a. This is the key element of this error. If the response is {1: 0}, then the encoder was found without issue and the cable is okay. If so, **see a different troubleshooting entry**. Make sure to close the direct connection though! (use dev.close()) You can also re-connect to the stage using SAM enable, and SAM connect in a **normal nfiuserver terminal**.
    b. If the response is **{1: 12}** then this corresponds to a "No Encoder Detected" error in the controller firmware. Thus, this is a cable connection issue.
7. Either way, you should make sure to close the direct connection to the stage:
    dev.close()

If you confirm that the issue is with the physical cable connection (ie. 6b above returns {1: 12}), then you should ask someone at the summit to tighten the connection shown in the picture at the top of this troubleshooting entry.

If you need to, you can turn off the SAM rotator completely (using modify -s kpic OUTLET_HC7=Off) and have the summit person disconnect the stage then reconnect. Remember to turn the rotator back on (modify -s kpic OUTLET_HC7=On) once they've reconnected it.

Once that's all done, you should be able to now talk to and control the stage like normal. Just remember to reconnect to it with the normal KPIC software using SAM enable, and SAM connect in a **normal nfiuserver terminal.**

NOTE: Generally speaking, this version of debugging the SAM rotator by connecting directly and using dev.getError(1) is very handy. The stage will report firmware errors as error codes that can then be looked up in the manual (link) starting on Page 5-78. Thus, if you've tried all other debugging steps for the SAM rotator in this troubleshooting document, consider trying steps 2 through 6 above.