APF Software

High-Level Overview

Overview

- How we know all relevant state
  - *(telescope, dome, spectrograph, ccd, weather…)*
- How we control all relevant state
  - *(except the weather…)*
- Debra’s Observation Flowchart
- How the Robotic Observer Replaces Debra
- What’s Hard About This?
- Supplement: Software *Tasks* for Organizing the Work.

How We Know State

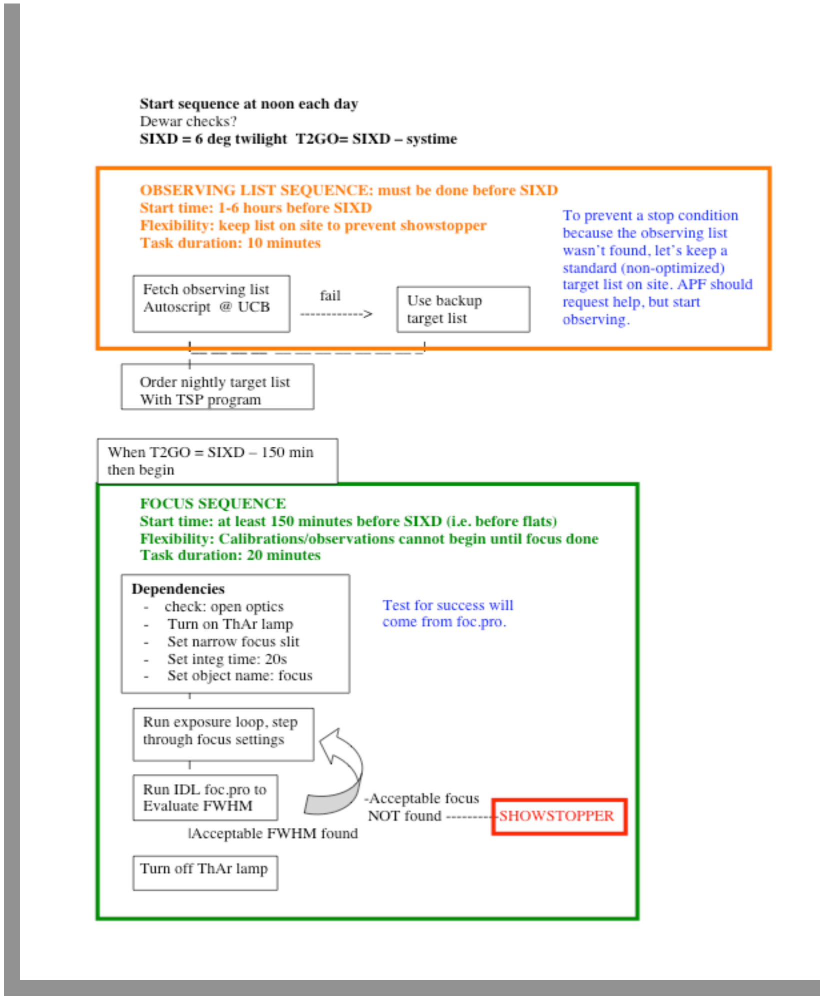
- All state represented by *KTL keywords*.
- Keywords are usually scalar values
  - *(float, int, bool, enum, string; small arrays poss.)*
- Examples: RA, Dec, Focus position, Lamp on/off, exposure time, readout window, humidity…
  - Shell: `show -s eos ra`
  - Tcl: `ktl read apfmot(calmrpos)`
- Can be read (polled) or broadcast (triggers calback)
- Keywords collected into *services*. Examples: spectrograph service, CCD service, weather service.

How We Control State

- Some (most) KTL keywords can be written:
  - Shell: `modify -s` *`service kwd=newvalue`*
  - Tcl: `ktl write $kwd $newvalue`
  - C: `ktl_write(kwd, …)`
- Examples:
  - Write CALMRPOS=2 to instruct Calibration Mirror stage to move to position 2.
  - Write EXPOSE=true to tell CCD controller to begin a new exposure

Debra’s Observation Flowchart

- Debra Fischer has supplied a detailed flowchart for planet-hunting observations.
- All its actions are mapped into corresponding keywords.



Start sequence at noon each day
Dewar checks?
SIXD = 6 deg twilight T2GO= SIXD – systime

OBSERVING LIST SEQUENCE: must be done before SIXD
Start time: 1-6 hours before SIXD
Flexibility: keep list on site to prevent showstopper
Task duration: 10 minutes

Fetch observing list
Autoscript @ UCB

fail ------------>

Use backup
target list

To prevent a stop condition because the observing list wasn’t found, let’s keep a standard (non-optimized) target list on site. APF should request help, but start observing.

Order nightly target list
With TSP program

When T2GO = SIXD – 150 min
then begin

FOCUS SEQUENCE
Start time: at least 150 minutes before SIXD (i.e. before flats)
Flexibility: Calibrations/observations cannot begin until focus done
Task duration: 20 minutes

Dependencies
- check: open optics
- Turn on ThAr lamp
- Set narrow focus slit
- Set integ time: 20s
- Set object name: focus

Test for success will come from foc.pro.

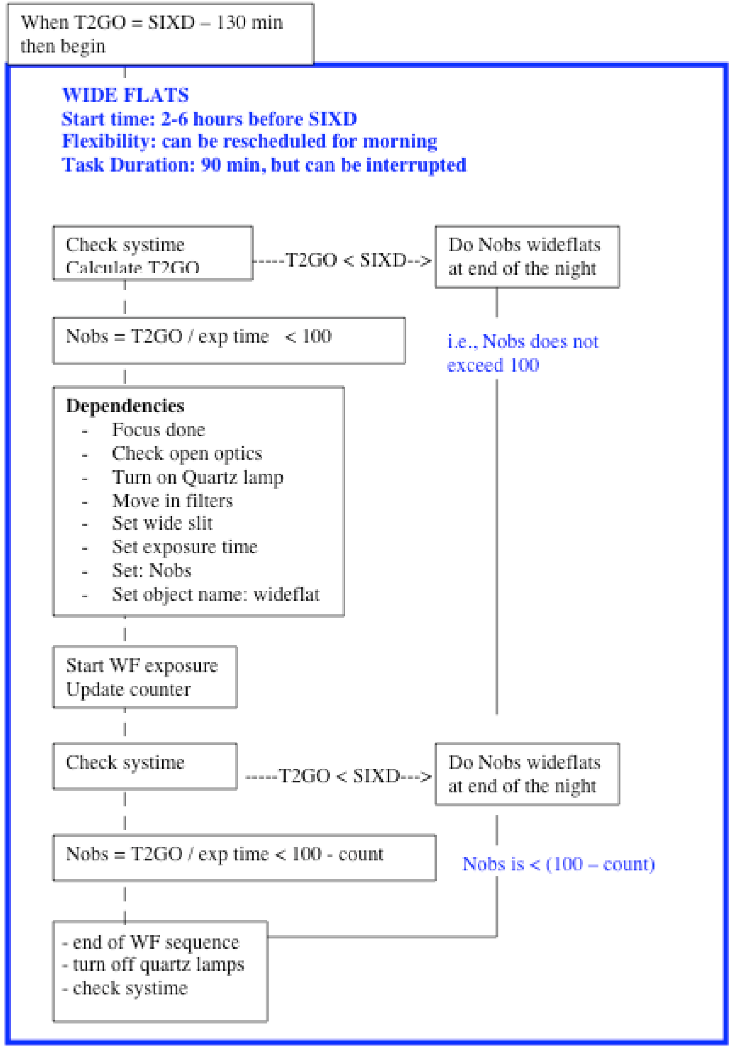
Run exposure loop, step
through focus settings

Run IDL foc.pro to
Evaluate FWHM

-Acceptable focus
NOT found ---------SHOWSTOPPER

|Acceptable FWHM found

Turn off ThAr lamp



When T2GO = SIXD – 130 min then begin

WIDE FLATS
Start time: 2-6 hours before SIXD
Flexibility: can be rescheduled for morning
Task Duration: 90 min, but can be interrupted

Check systime
Calculate T2GO

-----T2GO < SIXD-->

Do Nobs wideflats at end of the night

Nobs = T2GO / exp time < 100

i.e., Nobs does not exceed 100

Dependencies
- Focus done
- Check open optics
- Turn on Quartz lamp
- Move in filters
- Set wide slit
- Set exposure time
- Set: Nobs
- Set object name: wideflat

Start WF exposure
Update counter

Check systime

-----T2GO < SIXD--->

Do Nobs wideflats at end of the night

Nobs = T2GO / exp time < 100 - count

Nobs is < (100 – count)

- end of WF sequence
- turn off quartz lamps
- check systime

07-4-30

How a Human Observes

- Is guided by the Observation Flowchart
- Read state via GUI or command line
- GUI’s in Tcl/Tk. Monitors state and displays on screen.
- Commands actions via GUI or command line.
- GUI’s send ktl-write commands for user.

The Robots Takes Over

- We have a detailed observing plan;
- A complete mapping between that plan and keywords; and
- Complete control of state via keywords.
- To create a robot: a (conceptually) simple script implements each step of the observing plan.

So What's Hard About This?

- There are *many* conditions to be tested as part of ensuring safe observing.
- We have not spent much time considering all the subtle details of what might go wrong, and *how to respond* when we detect a non-normal condition.

Software Tasks

Robotic software will be divided into *tasks*. A *task* encapsulates a large set of actions into a single object, the software equivalent of a motor stage: it has completely standard start, stop, and status commands, and it encapsulates a

Software Tasks

- The main observing software is divided into *tasks*. A *task* encapsulates a large set of actions into a single object:
  - Focus task
  - Wide Flats task
  - Darks task
  - Observations task
  - *Etc*
- Any task knows how to do its own work and doesn’t care about others.
- All tasks share uniform start, stop, and status commands.
- Result: the Robotic Observer simply runs a series of tasks.