1.0 History and Background

The vendor-supplied user-interfaces for the APF telescope and dome (written by EOST and EOS) are not well designed nor well integrated with each other or with the interfaces that SPG developed for controlling the APF spectrometer and guider. In addition, the vendor-supplied GUIs have little error checking and make it easy for observers to shoot themselves in the foot, thereby putting the safety of the facility at risk. As a result, although observers can (and did) use these GUIs to operate the APF facility interactively, this was very error-prone and did not result in efficient operations.

It had always been the goal that the APF facility be fully robotic and able to operate unattended. This is a very complex task given the overall complexity of this facility and the fragility of its underlying vendor-supplied hardware and software. Rivera was tasked with developing the software that would provide this capability, but given the lack of adequate documentation for and the unpredictability of the vendor-supplied components, this was clearly too large a task for one person.

After Rivera's efforts to develop dynamically-scheduled robotic observing software for APF bogged down in early May 2013 and with the rapid approach of summer (with its clear skies and excellent seeing), it became clear that we needed an interim solution that would enable us to perform efficiently a scripted sequences of observations with APF each clear night. As such, there was considerable urgency in developing such a solution as rapidly as possible.

At the same time, funding for APF software had dried up and SPG staff were directed not to work on APF software if they had other (paying) projects to which their time could be charged. To fill thus void, Kibrick agreed in early May to volunteer his time towards developing such an interim solution at no cost to Lick; he selected csh as the scripting language for this application since it imposed no learning curve and provided both a stable development platform and a set of well-tested KTL-based tools, thus enabling very rapid prototyping and testing. The initial versions of the script were up and running from UCSC in late May and usage from UCB commenced in mid-June. What started as a simple, quick-and-dirty prototype to enable efficient use evolved organically over the next 6 months in response to feedback from the observing teams.

To speed development, the goal was to minimize the scope of the robot.csh script so that it only concerned itself with the sequence of steps needed to efficiently perform observations of a given set of targets. It does not attempt to do any dynamic scheduling or to otherwise optimize the selection of targets or the order in which they are observed. Rather, it carries out a sequence of operations in the order in which they are specified in the observer's starlist file for the given night. It also does not monitor the weather, but leaves that task to the checkapf service. In its original form (version 1), it assumed that there is a human observer who is responsible for: 1) opening the dome at sunset and closing it at dawn, 2) monitoring the performance of the telescope, dome, guider, and instrument and intervening if any serious problems arise, and 3) resetting the dead-man timer.

Version 2 became operational in late November 2013 and is intended to provide for unattended operation (i.e., there will be no human observer watching over it or the rest of the APF facility). Once version 2 is fully debugged and tested, Kibrick intends to reduce his involvement, at which point John Gates will assume increased responsibility for the robot.csh script (and the set of csh scripts that it invokes).

In the meantime, the observing team at UCSC has recently started using the robot.csh in a new context. Previously, a single invocation of the robot.csh file would run all night long, performing the sequence of observations specified in a static file that the script read via standard input (stdin). Now, the UCSC team has started running dynamic scheduling software that determines, in real time, the next target to be observed based on the current time, weather, and seeing conditions at the telescope. In this mode of operation, the robot.csh script is invoked separately for each target, i.e., with a single-line starlist file that contains only the current target to be observed. The UCB team is developing their own dynamic scheduling software.

Finally, it should be remembered that the robot.csh script is a hastily-written prototype that was intended as an interim solution that would enable efficient use of the APF facility software until such time as a better piece of software was developed to replace it. Unfortunately, at least so far, there has been no funding available to develop such a replacement, nor is any likely to be available in the foreseeable future. Hence, it will be important to develop effective strategies to enable the robot.csh script (and related software) to keep limping along into the future.

2.0 Introduction

The robot.csh script currently has only one command line argument (-skip), and it is optional. The script reads the observer's starlist file from stdin.

2.1 **Usage**

At present, while the source code for the robot.csh script (and related software) is maintained in CVS (.../cvs/lroot/apf/robot/groupk), the runnable copies are not part of the LROOT 'make' scheme and are not run from $LROOT/bin or $LROOT/bin/robot/. Rather, the script is currently run from one of two directories in the 'user' account on host mainz. A stable, known-to-work version is always maintained in the ~user/observing_scripts/ directory, while the version currently under development resides in the ~user/devel_scripts/robot/ directory. When there is a new version ready to test, we let observers try out the version in the development directory. If they encounter problems with that version, they can quickly switch to running the stable/tested version in the operational (observing_scripts/) directory. Once the version in the development directory has been adequately tested, it gets copied to the operational directory and the cycle continues.

Thus, on a typical night, the script could be invoked in this manner:

```
cd ~user/observing_scripts/

./robot.csh < STARLIST
```

where STARLIST is the full path to the observers starlist for that night. At present, the script will process the entries from the starlist one line at a time until either:

a) all of the entries in the star list have been processed, or
b) the script determines the elevation angle of the sun is > -8.9 degrees which keyword?
c) the script encounters an unrecoverable hardware or software fault
d) the 'checkapf' process senses the arrival of bad weather and initiates a close-up
e) A sigINT is sent by the 'apftask' facility in response to either checkapf.OPEN_OK=false, checkapf.MOVE_PERM=false, checkapf.INSTR_PERM=false, or aptask.SCRIPTOBS_CONTROL=Abort

In case 'a', the script exits with a status return of 0, while in all other cases it exits with a status return of 1.

When invoked with a STARLIST consisting of only a single entry, the script will simply process that one target and then exit. This is the manner in which the script is invoked from a higher-level dynamic scheduling script.

Note that if the script is invoked during twilight and before the elevation of the sun is <= -8.9 degrees, then the script will exit without processing any of the entries in the STARLIST file.

In its current form, if the telescope, dome, instrument, or guider transitions to a 'non-ready' state, then the robot.csh script will terminate any in-progress science exposure and will start skipping over exposures and targets until the non-ready condition is cleared. We may want to think more carefully about how the scripts responds to the such non-ready states, especially when the observing mode is robotic.

As mentioned earlier, the robot.csh script currently has only one command line argument, '-skip', and it is optional. It works in conjunction with the apftask keyword 'scriptobs_lines_done', which is a read/write integer keyword:

If '-skip' is specified, then the script will skip over the first "N" lines of the STARLIST file and the first target that the script will attempt to acquire will be the one whose coordinates are specified on line "N+1", where "N" is read from the apftask keyword 'scriptobs_lines_done'.

If '-skip' is not specified, then:
a) the apftask keyword 'scriptobs_lines_done' will be set to zero, and
b) the first target that the script will attempt to acquire will be the one whose coordinates are specified on line 1 of the STARLIST file.

In either case, the script will increment the value of the apftask keyword scriptobs_lines_done after it completes processing each line of the starlist file. Thus, at any given time, this keyword should reflect the number of lines of the given starlist file on which the script has completed processing so far.

Here are some examples that illustrate the usage of the optional '-skip' command line option and the apftask keyword 'scriptobs_lines_done':

A. ./robot.csh < ~user/starlists/MY_STARLIST

This would work exactly the same as it in version 1 of the robot.csh script; the script would start processing at the first line of MY_STARLIST. Also, because the option '-skip' parameter is NOT specified on the command line, the script would first set the apftask keyword 'scriptobs_lines_done' to zero PRIOR to processing the first line of MY_STARLIST. After the script finishes processing each line of MY_STARLIST, it will increment the apftask keyword 'scriptobs_lines_done' by one.

B. modify -s apftask scriptobs_lines_done=0

./robot.csh -skip < ~user/starlists/MY_STARLIST

This example is functionally equivalent to example A, as it would skip zero lines at the top of MY_STARLIST and would thus begin processing with the first line of MY_STARLIST. However, because the '-skip' option IS specified on the command line, the script would NOT alter the value of the apftask keyword 'scriptobs_lines_done' prior to processing the first line of MY_STARLIST. After the script finishes processing each line of MY_STARLIST, it will increment the apftask keyword 'scriptobs_lines_done' by one.

C. modify -s apftask scriptobs_lines_done=10

./robot.csh -skip < ~user/starlists/MY_STARLIST

In this example, the script would skip over the first 10 lines at the top of MY_STARLIST and would thus begin processing with line 11 of that file. While it was still processing line 11, the 'scriptobs_lines_done' keyword would retain its value of 10. Once the script had finished processing line 11, then the 'scriptobs_lines_done' keyword would be incremented by one so that its value would then become 11, and after each subsequent line was processed the keyword would be incremented by one.

D. In this overly-simplified example, the robot.csh script could get invoked several times during the night if it is interrupted by weather-induced closures of the telescope and dome or anything other situation that caused a given instance of the robot.csh to terminate prior to completing the observations specified in the MY_STARLIST file:

```
#############################################################
# At the start of the night, set the number of lines to skip to 0
#
modify -s apftask scriptobs_lines_done=0
#
#
# Stay in this loop until starlist is done or until facility becomes “non-ready”
#
while ( `show -s apfmon -terse readysta` == "OK" )
  ./robot.csh -skip < ~user/starlists/MY_STARLIST
  if ( $status == 0 ) then
    echo "Robot.csh has successfully processed the entire starlist"
    break
  else
    set skipcount = `show -s apftask -terse scriptobs_lines_done`
    echo "Robot.csh was interrupted and did not complete starlist"
    echo "Robot.csh has processed the first $skipcount lines so far"
#   Now wait for high-level software to make telescope & dome ready
    gwaitfor -s checkapf ‘$open_ok == “true”'
    continue
  endif
end
#############################################################
```

2.2 **Format of the STARLIST file**

The starlist file is a human-readable, ASCII format file that consists of a sequence of lines, with one line for each target to be observed. Each line consists of a set of required and optional parameters. All required parameters must be specified on each line, and the those parameters must be specified in a specific, fixed order. The optional parameters must be specified AFTER the last required parameter and can be specified in any order relative to each other. Optional parameters that are not specified will be assigned default values.

The logic that parses each line of the starlist is the least developed part of the script; it is extremely basic and unforgiving; there is hardly any syntax or bounds checking, and any syntax errors in the starlist file will either cause the script to exit with a cryptic error or to behave in unexpected ways. At present, it is the observer’s responsibility to ensure that their starlist file is syntactically correct and that it contains the correct coordinates and proper motions for each target listed. They understand this.

The format of the each line of the starlist file is:
1. 1 line per target
2. No blank lines between targets
3. Fields are separated by whitespace
4. Unless otherwise specified, each field is mandatory.
5. The sequence '# ' (hash mark followed by a space) starts a comment

Field #	keyword	Description
1		Target name (cannot contain embedded whitespace)
2		RA hours (integer)
3		RA minutes (integer)
4		RA seconds (float)
5		DEC degrees (integer)
6		DEC minutes (integer)
7		DEC seconds (float)
8		EPOCH (must be 2000; coords must be J2000, epoch 2000)
9	pmra=	proper motion in RA (units= mas/year)
10	pmdec=	proper motion in DEC (units= mas/year)
11	vmag=	Visual magnitude (documentary only)
12	texp=	maximum exposure time for science exposure (seconds)
13	I2=	iodine cell position (Y=in N=out)
14	lamp=	comparison lamp (placeholder for now, must be "none")
15	uth=	UT hours for observation start (advisory only)
16	utm=	UT minutes for observation start (advisory only)
17	expcount=	Exposure meter count at which to terminate exposure
18	decker=	Slit/decker to use (see below)
19	do=	blank if target is in uncrowded field (see below)
	count=	[OPTIONAL FIELD] number of science exposures to take
	foc=	[OPTIONAL FIELD] requests an M2 focus option

Note that the coordinates for all targets must be specified as J2000, epoch 2000, equinox 2000. The vendor-supplied APF Telescope software will **not** work correctly for coordinates specified in any other coordinate system. If you have coordinates in some other system, use Simbad (or equivalent conversion facility) to render your coordinates into this form.

Field 18, the "decker=" field, can be one of the following single-letter script codes:

DECKERORD	DECKERNAM	Script code
1	Pinhole	P
2	K (1.00:12.0)	K
3	L (2.00:12.0)	L
4	M (1.00:8.0)	M
5	B (2.00:8.0)	B
6	W (1.00:3.0)	W
7	T (2.00:3.0)	T
8	S (0.75:8.0)	S
9	N (0.50:8.0)	N
10	O (8.00:8.0)	O

Field 19, the "do=" field, is used to indicate if the target object has any nearby (less than 20 arcsec away) neighbors that are either brighter than or of comparable brightness to the target object. For most targets, that will not be the case, and the value of this field is left blank. Otherwise, this field should be set to a non-blank string one or more letters. The string itself is not significant, and can be used for whatever documentary purpose the user finds appropriate; what is significant is whether or not the value of the field is blank.

In the default case (i.e., the "do=" parameter is blank), our automated acquisition procedure is to do a blind slew to the target coordinates, and once there, the autoguider locks onto the brightest target in the the guide camera's 55" x 55" field of view and centers it on the slit. As long as the target object is the brightest target in that FOV (as it will be for most targets), this works fine, since the pointing accuracy of the APF Telescope is very good (RMS pointing errors are < 2", while worst-case pointing errors are < 9").

In those cases where the target object has a nearby neighbor that is either brighter or of comparable brightness, setting the "do=" parameter to a non-blank value will invoke an alternate acquisition procedure. First, the telescope will slew to an unambiguous pointing reference star that is close to the target star. The reference star will then be centered on the slit (using the normal acquisition procedure), thereby reducing the telescope pointing uncertainty for that area of the sky. It will then slew to the target object and lock onto the brightest target that is within a radius of 3.2 arcsec of the slit center, thus ensuring that the target star (and not some nearby neighbor) is centered on the slit.

The "count=" field is optional. If not specified, the "count" will default to 1, and only one science exposure will be taken of this target. To take "N" exposures of the target (using the parameters specified on the current line of the target list), specify "count=N", where N is >= 1.

The “foc=” field is optional. It determines whether or not the M2 focus will be re-checked and/or re-measured (see section 2.3) after this target has been acquired but before any science exposures for this target are started. Valid values are 0, 1, and 2. If the parameter is omitted from the entry for a given target, it defaults to 0.

The values of this "foc=" parameter are interpreted as follows:

foc=0 No change from current behavior. The check_focus script will NOT be invoked for this target unless:
a) the automatically-determined guider camera exposure time for this target is <= 1.0s,
b) The apftask keyword scriptobs_var_1 is set to the value "robot_autofocus_enable", and
c) The target flux as measured by the autoguider is > 5,000 DN.
If the check_focus script is invoked for this target AND if it determines that M2 needs to be refocused, then the measure_focus script will also be invoked.

foc=1 If the automatically-determined guider camera exposure time for this target is <= 1.0s and the target flux is > 5,000 DN, then the check_focus script will be invoked for this target, regardless of the current value of the apftask keyword scriptobs_var_1. If the check_focus script is invoked for this target AND if it determines that M2 needs to be refocused, then the measure_focus script will also be invoked.

foc=2 If the automatically-determined guider camera exposure time for this target is <= 1.0s and the target flux is > 5,000 DN, then the check_focus script will be skipped and the measure_focus script will be called directly, regardless of the current value of the apftask keyword scriptobs_var_1. Setting 'foc=2' will ensure that the M2 focus will be re-measured, so long as the target is sufficiently bright (as determined by the guide camera exposure time and measured flux of the target).

Note that the optional "count=" and "foc=" parameters can be specified individually or together and in either order, but in either case they should follow the required "do=" parameter. Here are some examples:

```
    HR7236  19 06 14.9 -04 52 57.2  2000 pmra=-18.69   pmdec=-91.02   vmag=3.4
texp=900 I2=Y lamp=none uth=13 utm=34 expcount=1e+09 decker=W do=

    HR7236  19 06 14.9 -04 52 57.2  2000 pmra=-18.69   pmdec=-91.02   vmag=3.4
texp=900 I2=Y lamp=none uth=13 utm=34 expcount=1e+09 decker=W do= count=2

    HR7236  19 06 14.9 -04 52 57.2  2000 pmra=-18.69   pmdec=-91.02   vmag=3.4
texp=900 I2=Y lamp=none uth=13 utm=34 expcount=1e+09 decker=W do= count=2 foc=2

    HR7236  19 06 14.9 -04 52 57.2  2000 pmra=-18.69   pmdec=-91.02   vmag=3.4
texp=900 I2=Y lamp=none uth=13 utm=34 expcount=1e+09 decker=W do= foc=2 count=2

    HR7236  19 06 14.9 -04 52 57.2  2000 pmra=-18.69   pmdec=-91.02   vmag=3.4
texp=900 I2=Y lamp=none uth=13 utm=34 expcount=1e+09 decker=W do= foc=2
```

If the observer prepares a target list in the above described format, it can then be supplied as the standard input to the robot.csh script so that the indicated sequence of exposures will be obtained without the observer having to take any further action. The script will perform all of the actions needed to acquire each target, adjust the parameters of the guide camera for a proper exposure level, center the star on the spectrometer slit, configure the spectrometer, and obtain the requested number of science exposures of that target. The script will continue to run until it reaches the end of the target list, determines that dawn is near, encounters a non-recoverable error, or until it is interrupted (e.g., via control-C or receipt of SIGINT).

2.3 The M2 focus scripts

The M2 focus scripts were developed by Brad Holden. They are written in Python and are designed to check and measure the focus of the APF's secondary mirror. These scripts can be run independently or can be invoked from the robot.csh script, based on the setting of the apftask keyword 'scriptobs_var_1' and/or the value of the optional "foc=" parameter that is specified on the STARLIST file entry for a given observation (as described above).

2.3.1 check_focus.py

This script is designed to perform a quick "spot-check" of the current M2 focus. It assumes that the telescope is currently tracking a relatively bright star (i.e., guide camera exposure time <= 1 second and flux > 5000). The script will first collect several measurements of the apfguide keyword FWHM using the currently-set value of the M2 focus. (The FWHM keyword reports the full-width half-maximum of that star as measured by the sextractor program, which is invoked by the apfguide process for each guide camera image as it becomes available). From those measurements, the script will then calculate an average FWHM at the current M2 focus.

Next, it will command the telescope control software (via the eostele keyword FOCUS) to increment the M2 focus by an amount sufficient to drive the image of star (on the guide camera) significantly out of focus. It will then collect several measurements of the star's FWHM at that focus setting; from those measurements, it will compute an average FWHM for that "plus-side out-of-focus" setting of the M2 focus.

It will then decrement the M2 focus by an amount sufficient to drive the image of the star significantly out of focus (on the other side of focus) and will collect several FWHM measurements at that focus setting; it will compute an average FWHM for that "minus-side out-of-focus" setting of the M2 focus.

Once it is done making these measurements, the script will restore the M2 focus to its original value.

The script will then fit a parabola to the 3 data points it collected, using the M2 focus settings as X and the corresponding average FWHM values as Y. From that fit, it will determine the M2 focus value (i.e., the X coordinate) that corresponds to the minimum Y value for that parabola. If that M2 focus value is sufficiently close to the original M2 focus setting, then the script will output to stdout a message of the form:
"No need to refocus".
In that case, the robot.csh script will initiate the specified science exposures.

Otherwise, if that "parabolic minimum" M2 focus value is not sufficiently close to the original M2 focus setting, then the robot.csh script will invoke the measure.focus.py script (see below).

2.3.2 measure_focus.py

The measure_focus.py script can be called directly or invoked from the robot.csh script. In either case, it assumes that the telescope is currently tracking a relatively bright star (i.e., guide camera exposure time <= 1 second and flux > 5000).

It will be invoked from the robot script if either:
a) the check_focus.py script was invoked and it concluded that a re-determination of M2 focus was needed, or
b) the optional "foc=" parameter was specified on the starlist entry for this target and was assigned a value of 2.

This script is similar to the check_focus.py script, but it performs a more extensive set of measurements to accurately determine the nominal M2 focus. If that process succeeds, the script then commands the M2 focus to that newly-determined nominal value.

Like the check_focus.py script, the measure_focus.py obtains multiple measurements of the FWHM of the guide star image while stepping the M2 focus through several different settings. The goal is obtain these measurements over a range of settings that spans both sides of focus; measure_focus.py currently obtains measurements at 5 different M2 settings on one side of focus and 5 more settings on the other side of focus. Currently, the step size between these settings is fixed, but in the next version of the script, the step size will be a function of the current atmospheric seeing, as measured by the autoguider software.

At each M2 focus setting, several measurements of the FWHM are obtained and are used to compute both the average FWHM and the RMS error for that setting. A parabola is fit to those averages, using a fitting method which gives more weight to those M2 focus settings whose average FWHMs have lower RMS errors. If the fitting process succeeds, the M2 focus is set to the value determined from that fit. Otherwise, if the fitting process fails, an error is logged and the M2 focus is reset to its original value.

2.4 **The apftask SCRIPTOBS series of keywords**

The robot.csh script identifies itself to the apftask service as the 'scriptobs' (for scripted observing) task. Like any other apftask task, there are a set of standard keywords for the scriptobs task, such as scriptobs_control, scriptobs_message, etc. These standard task keywords are described in the documentation for the apftask service.

In addition to those standard keywords, there are some additional scriptobs task keywords that have meanings that are specific to this task, including the following:

scriptobs_lines_done	Number of lines of starlist file that have been processed (See more detailed description in Section 2.1, Usage)

scriptobs_vmag	The V magnitude of the current target
scriptobs_var_1	If set to 'robot_autofocus_enable', and if the current starlist entry does not have "foc=2" specified, then the robot.csh script will invoke the check_focus.py script after the target specified in that starlist entry has been acquired but before any science exposures commence for that target. Otherwise, if the 'scriptobs_var_1' keyword is set to any other value, then the robot.csh script will only invoke the check_focus.py script if the current starlist entry has "foc=1" specified.
scriptobs_var_2	An integer value (in the form of a "Unixtime" value, i.e., the number of seconds since 1/1/1970) that corresponds to the absolute date/time of the last APF slew retry attempt. This keyword is used to remember that time across restarts of the robot.csh script and is used to ensure that such retries are not attempted more than once every 12 hours.
scriptobs_var_3	Currently unused
scriptobs_control	If set to "Abort", will cause the apftask service to send a SIGINT signal to the robot.csh script. Upon receipt of that signal, the robot.csh script will transfer control to its interrupt_exit block, where it will stop and read out any in-progress science exposures before exiting with a return code of 1. If set to "Pause", the robot.csh script will pause whenever it finishes acquiring the next target; it will remain paused until this keyword is either set to "Proceed" (in which case the script will resume running from the point at which it had been paused) to "Abort" (in which case the script will exit as described above).

2.5 **Output files**

At present, the robot.csh script writes to two separate output files in the directory from which the script is run:

2.5.1 The robot.log file

This file logs all of the actions that the script takes as it carries out the observations that are specified in the STARLIST file which it reads from standard input. It also logs any error conditions that arise in the course of those observations. At present, messages are simply appended to any existing 'robot.log' file in the directory from which the robot.csh script is run. Hence, at present, this file will grow until it is backed up and truncated.

2.5.2 The observed_targets file

This file logs all of the STARLIST entries that have been successfully observed since the 'observed_targets' file was created or last truncated to zero length. After the robot.csh script completes the processing of a given entry from the STARLIST file, that entry is appended onto the 'observed_targets' list. Any STARLIST file entries that the robot.csh script is unable to observe (e.g., because the target isn't visible) are not copied to the 'observed targets' file.

If the 'observed_targets' file is truncated to zero length at the start of the night, then, during the course of the night, it will provide a running record of which targets have successfully been observed so far. By performing a 'diff' between the STARLIST and the 'observed_targets' file, one can obtain a list of the targets that have not been observed. If, at the end of the night, all of the targets in the STARLIST file have been successfully observed, then the 'observed_targets' file should be identical to the STARLIST file.

3.0 The companion CSH scripts invoked from the robot.csh script

For each line of the STARLIST file, the robot.csh script attempts to acquire the specified target and to perform the requested observations on that target. In doing so, the script invokes a series of helper scripts (one of which invokes a separate C program), as follows:

3.1 **windshield.csh**

The primary goals of the windshield.csh script are to:
a) determine if the requested observation of the specified target is possible, and, if so,
b) provide maximum shielding of the telescope from the wind during that observation.

The script accomplishes this "wind shielding" function by positioning the two dome shutter panels so as to create the smallest opening that is sufficient for the observation to be conducted without those shutter panels vignetting the telescope's field of view and without those panels needing to move during the observation.

The secondary goals of the script are to determine the appropriate dome shutter operating mode and the appropriate azimuth cable wrap selection to use for the requested observation, taking into account, where possible, the preferences expressed by observer.

To accomplish both these primary and secondary goals, the script needs to evaluate the trajectory that the specified target will follow across the sky during the course of the requested observation. The calculation of that trajectory is performed by a separate C program (windshield.c) that is invoked by the windshield.csh script (see Section 3.1.4).

3.1.1 Background

Recall that the APF dome features a dual up-and-over shutter design. When the dome shutters are opened for observing, they can be operated in two distinct modes: "split shutters and "up-and-over".

split-shutters mode:	The upper (rear) shutter panel is drawn back over the apex of the dome and the lower (front) shutter panel is dropped down so that the telescope looks out through the opening between the two shutter panels.
up-and-over mode:	Both shutter panels are moved together up and over the apex of the dome so that the telescope looks out the large opening that is underneath both panels.

Objects whose position in the sky is between 43.5 and 81.5 degrees of elevation can be observed with the shutters operating in either mode, although the split-shutters mode will provide more effective wind shielding because it will result in a smaller opening to the wind. To avoid vignetting from the dome shutters, objects whose position on the sky is above 81.5 degrees elevation must be observed in split-shutters mode, while those with positions below 43.5 degrees elevation must be observed in the up-and-over mode; however, even in that mode, objects whose position is below 15 degrees elevation will be vignetted by the bottom edge of the dome shutter opening.

For a requested observation of a specified target, the windshield.csh script selects the dome shutter operating mode by looking at the relevant parameters (i.e., the minimum and maximum elevation angles) that are associated with the trajectory (as computed by the windshield.c program) that the target will trace across the sky during the course of that observation, as follows:

a) If the trajectory's maximum elevation is > 81.5 degrees, select split-shutters mode
b) If the trajectory's jminimum elevation is < 43.5 degrees, select up-and-over mode
c) Otherwise, select the mode according to the 'shutter mode preference' specified.

In case 'c', if no "dome shutter operating mode preference" has been specified, then the windshield.csh script will select whichever mode will result in a shorter acquisition time for the specified target based on the current position of the dome shutter panels.

However, given that most of the time the nighttime wind speeds on Mt. Hamilton are above 5 MPH and that even at such moderate wind speeds the APF Telescope is very sensitive to wind shake, version 2 of the robot.csh script currently indicates to the windshield.csh script that the "split-shutters" mode is preferred and should always be selected in case 'c', even if doing so will result in a longer acquisition time for the specified target. In future versions of the robot.csh script, overall observing efficiency might be improved by making that preference conditional on the current wind speed, and to instead specify "no preference" whenever the windspeed is close to zero.

Just as the selection of the dome shutter operating mode depends on the trajectory that the selected target will follow over the course of the requested observation, so does the selection of which “wrap” of the azimuth cable wrap should be selected.

Recall that the azimuth cable wrap limits run from -110 degrees to +310 degrees of azimuth, for a total range of 420 degrees. The target’s trajectory will determine where in that range the observation should begin so that the telescope will not encounter an azimuth cable wrap limit during the course of the observation.

For example, assume that the starting azimuth for a given observation is 300 degrees; such an observation could be started at an azimuth cable wrap angle of either +300 degrees or -60 degrees, depending on the maximum duration of that observation and the corresponding trajectory. If the trajectory the target will follow will take it to larger azimuth angles and if the observation duration will result in more than 10 degrees of azimuthal tracking motion, then in order to avoid hitting the positive azimuth cable wrap limit at +310 degrees, the observation should be started at an azimuth cable wrap angle of -60 rather than +310 degrees. 300

In other cases, the characteristics of the trajectory associated with the requested observation of a specified target may result in a choice of two acceptable azimuth cable wrap angles (located 360 degrees apart) at which such an observation could be started. In that case, if no preference is specified, then the windshield.csh script will select whichever choice would result in a shorter time to acquire the specified target based on the current azimuth (on the -110 to +310 degree scale) of the telescope. At present, version 2 of the robot.csh script expresses “no preference” with respect to the starting azimuth angle for an observation; hence, in those cases where there are two acceptable choices for the starting azimuth angle, the windshield.csh script will select the one which provides the faster acquisition time for the specified target.

In summary, the trajectory analysis that is performed by the windshield.c program (invoked from the windshield.csh script) plays a critical role in selecting the dome shutter operating mode and the starting azimuth for a given observation of the specified target; it also is used to determining the angles to which the front and rear dome shutter panels should be positioned to maximize wind shielding of the telescope while also ensuring that the telescope’s field of view will not be vignetted by the dome shutter panels during the course of the requested observation.

3.1.2 Wind shielding implementation overview

The wind shielding algorithm (which determines, for a given telescope elevation angle, the corresponding positions of the front and rear dome shutter panels that will minimize the area of the shutter opening without vignetting the field of view of the telescope) was developed by the engineers at EOS (the vendor that supplied the APF Dome) and included as part of the proprietary “Observatory Server” software that was delivered as part of the contract; see the file:
dresden:/cifs/dcc/eos/bin/servers/ObservatoryServer@Observatory/windShield.txt

Unfortunately, that software was inflexible, difficult to use, and did not fully meet our requirements. As a result, the EOS-written Observatory Server is not currently used and is disabled via the EOS System Management Interface.

The wind shielding implementation that EOS developed did not perform any trajectory analysis and instead required the observer to determine (prior to acquisition) which dome shutter operating mode was appropriate for a given observation of a specified target. In addition, it was designed so that the dome shutter panels would continuously follow the telescope as it tracked the target across the sky. This has the advantage of providing the smallest possible dome shutter opening and hence the most effective wind shielding for the telescope.

Unfortunately, there were several problems with that approach. First, it meant that the motors for the dome shutter panels would be cycling on and off every 30 seconds or so, resulting in added wear and tear for those motors and the gear boxes that couple them to the shutter drives. We had already experienced a failure of one of these gear boxes and, because these are a very long lead time item, that failure rendered the dome inoperative for many weeks. Accordingly, we preferred a wind shielding implementation that would only require the dome shutter panels to be repositioned once per observation.

Second, given the potential for components to come loose from the dome shutter mechanisms or for grease or other debris to be knocked loose while the shutter panels are in motion, we were reluctant to move the dome shutter panels over an unprotected, up-looking primary mirror since any such items falling onto the mirror could cause serious damage.

As a result, the wind shielding approach implemented via the windshield.csh script is a compromise that attempts to balance these concerns. While it may not provide as small a dome shutter opening as would be the case if the dome shutter panels continuously followed the telescope as it tracked a target across the sky (as was the case in the original EOS implementation), it still provides a relatively small opening while only requiring that the dome shutter panels be moved once per observation.

To avoid the problem of moving the dome shutter panels over the unprotected primary mirror, one solution would be to close the primary mirror covers prior to moving the dome shutter panels and then re-opening the mirror cover after those motions are completed. We discarded that solution for two reasons. First, the APF primary mirror cover is very slow, requiring about one minute to close and one minute to open. Second, the mirror cover mechanisms are not terribly robust and we want to avoid incurring any unnecessary wear. Barring closures (during observing) triggered by deteriorating weather conditions, we typically operate the mirror covers only twice per night, opening them at the start of the night and closing them at the end of the night.

Our alternate approach for protecting the primary mirror during motions of the dome shutter panels combines the slewing of the telescope in azimuth (to acquire the next target) in parallel with moving those shutter panels while also pointing the telescope away from the zenith prior to moving the panels. These motions are overlapped in time so as to minimize the overall time need to acquire the next target to be observed.

Once the windshield.csh script has confirmed (via the trajectory computations performed by the windshield.c program) that the specified target will be visible over the entire course of the requested observation, it will then attempt to carry out the following steps:

1. If the current positions for both shutter panels are already each within 1 degree of the corresponding nominal position as computed for this observation by the windshield.c program, then the script will consider both panels to be positioned "close enough" to their respective nominal positions and will skip steps 2 through 6 below. Otherwise, the script will:
2. Switch the atmospheric dispersion compensator (ADC) from "Track" mode to "Pos" mode and command it to slew to the appropriate position that corresponds to the starting elevation angle for this observation (as computed by the windshield.c program); this avoids having the ADC follow the elevation slew of step 3 below.
3. Start the telescope slewing to the approximate starting azimuth angle for this observation (as computed by windshield.c) and to an elevation of 15 degrees, so as to point the primary mirror away from the zenith in preparation for moving the dome shutters.
4. Wait for the telescope elevation angle to drop below 16 degrees (NOTE: the telescope may still be slewing in azimuth).
5. Command the dome shutter panels to start moving to their respective nominal positions (for this observation) as computed by the windshield.c program.
6. Wait for both dome shutter panels to finish moving to their respective commanded positions.
7. Command the telescope to start slewing to the starting azimuth and elevation angles computed for this observation by the windshield.c program.
8. Set the ADC to "Track" mode so that it will track the telescope in elevation.
9. Wait for the telescope to arrive at the starting azimuth (so as to ensure that we start on the correct "wrap" of the azimuth cable wrap) and starting elevation angles that were commanded in step 7
10. Command the telescope to start slewing to the specified RA and DEC (as conditioned by the specified proper motions and parallax) for this target.
11. Wait for the telescope to finish slewing and start tracking at the RA and DEC specified in step 10.
12. Wait for the ADC to report "Tracking" status

3.1.3 Wind shielding implementation details

The windshield.csh script obtains some of its inputs via keywords and others via the command line. The target coordinates, proper motion, and parallax are obtained via the EOSTELE keywords targra, targdec, targmuad (proper motion in declination), targmuar (proper motion in right ascension), and targplax (parallax). Prior to invoking the windshield.csh script, the robot.csh script sets those keywords using the parameters specified on the current line of the STARLIST file that it is processing.

Four other parameters are passed from the robot.csh script to the windshield.csh script via the command line in the following order:

length (seconds) - maximum time needed to observe this target
prefcfg (flag) - Dome shutter mode preference: 0=split shutters 1=up&over 2=none
verbose (flag) - Set to a non-zero value to enable debug output to standard output
wrapcfg (flag) - Az cable wrap preference: -1=lower wrap +1=upper wrap 0=none

Note that the robot.csh script derives the maximum observation 'length' from the "texp=" and "count=" parameters specified on the STARLIST file entry for the current observation; if the optional "count=" parameter is not specified on that entry, it defaults to a value of 1. The 'length' also includes 40 seconds for each CCD readout that can't be overlapped with the slew to the next target, and an additional 360 seconds to allow time for target acquisition and for the possibility of checking and re-measuring the M2 focus. Hence, the robot.csh script computes the value of the 'length' parameter as:

length = (count*texp) + (40*(count-1)) + 360

As noted in Section 3.1.1, the robot.csh scripts specifies "split shutters" as the preferred dome shutter operating mode and "no preference" for the azimuth cable wrap.

The windshield.csh script first takes a snapshot of the current position of the telescope and the dome shutters. Next, it invokes a separate C-language executable (the 'windshield.c' program, see section 3.1.4 below) that will compute the approximate alt/az trajectory (based purely on the target's RA and DEC, while neglecting proper motion and parallax) that the specified target would follow if acquired at the current time and tracked for the number of seconds specified in the script's 'length' parameter.

If, at any point within that computed approximate trajectory the target would:
a) drop below 15 degrees elevation (i.e., too low to be observed from the APF Dome),
b) track within 0.5 degrees of the zenith (i.e., too close to the zenith 'keyhole'), or
c) exceed either the clockwise or counterclockwise limits of the azimuth cable wrap,
then the 'windshield.c' program will return an error status to windshield.csh script, which will in turn return an error status to the robot.csh script to indicate that the requested observation is not possible at this time.

If the windshield.c program reports that the specified observation is not possible, and if that observation involved more than one science exposure on the specified target (i.e., if the “count=” parameter specified on the STARLIST entry for this target had a value > 1), then the robot.csh script will decrement its copy of the desired exposure count for this target (thereby reducing the overall observation time required), and as long as the resulting exposure count remains > 0, it will then re-invoke the windshield.csh script, in the hope that the reduction in overall observation time on that target would result in an acceptable trajectory. If the exposure count is or becomes 1 and the trajectory computed by the windshield.c program remains unacceptable, then the robot.csh script will skip over that target and start processing the next entry on the STARLIST.

If the ‘windshield.c’ program determines that the computed trajectory (for the current observation parameters for this target) is acceptable (i.e., that it does not violate constraints ‘a’, ‘b’, and ‘c’ listed above), then it will return an exit status of 0 (success) to the windshield.csh script, along with various computed parameters that it outputs to standard output.

Upon a successful return from the windshield.c program, the windshield.csh script will attempt to perform the 12 steps described at the end of Section 3.1.2 above. If all of those steps are successful, the windshield.csh script will return an exit status of 0 (success) to the robot.csh script.

The complete list of exit status values returned by the windshield.csh is as follows:

0: successful
1: Error processing trajectory
2: Target not visible for specified observation length
3: Instrument not released or permission not granted
4: Telescope not released or permission not granted
5: Unrecoverable telescope slew fault
6: Error commanding dome shutters to move
7: Invalid front shutter position for up-and-over configuration
8: Invalid shutter positions computed
9: Dome shutters failed to complete motion within allotted time
10: Dome shutters repeatedly failed to achieve commanded positions
11: ADC failed to achieve tracking status within allotted time

The robot.csh script will treat windshield.csh exit status values of 1 or 2 as described earlier, i.e., it will reduce its copy of the exposure count for this observation and (so long as that count remains > 0) re-invoke the windshield.csh so see if that will result in an acceptable trajectory. If not, the robot.csh script will skip over that target.

Windshield.csh exit status values > 2 are currently considered to be fatal errors and will cause the robot.csh script to take its error exit.

3.1.4 windshield.c

This windshield.c program can be used both as:
- a stand-alone observation planning tool, and
- a component of the robotic observing software suite.

This program is purely computational and does not actually move either the dome or the telescope. The only libraries to which it links are the Unix standard I/O and math libraries. You should be able to compile and link this on just about any Unix-based system using the command:

```
cc -o windshield windshield.c -lm
```

This program reads 11 parameters from the command line and outputs 15 parameters to standard output. All other messages and debugging info are written to stderr.

Among its outputs are the angles to which the front and rear shutters of the APF Dome should be pre-positioned prior to acquiring the next target so as to minimize the size of the dome shutter opening (and hence the exposure of the telescope to wind) while at the same time ensuring that the opening will be large enough to enable the specified target to be observed for the indicated duration (see 'length' input parameter described below) without being vignetted by the dome shutters.

This is accomplished by computing the trajectory (in alt-az coords) that the target will follow over the specified period of time and determining the minimum and maximum telescope altitude (elevation) angle over the course of that trajectory.

The minimum and maximum elevation angles are then used to determine the corresponding angles to which the front and rear shutter panels should be positioned, using the mathematic model provided by EOS/EOST.

The minimum and maximum azimuth angles are also determined for the calculated trajectory. These are checked against the telescope azimuth cable wrap limits. If the calculated trajectory would encounter these limits, then (if possible) the azimuth angles are adjusted modulo 360 degrees to avoid hitting the cable wrap limit.

Note that whenever possible, this program will generate dome shutter panel positions that correspond to the preferred mode of shutter operation (i.e., split-shutter versus up-and-over mode) as specified by the user as a command line input parameter (see input argument 5). If the user does not specify a shutter mode preference, and the trajectory for the specified observation can be accommodated by either shutter mode, then this program will select the mode which provides the fastest acquisition time given the "current" position of the shutter panels (as specified on the command line; see input arguments 9 and 10). However, for some observations, the choice of the dome shutter operating mode is dictated by the trajectory associated with the specified observational parameters; in those cases, the user-specified preferred shutter mode may not be possible.

Similarly, whenever possible, this program will generate starting azimuth positions for each observation that correspond to the preferred wrap (as specified on the command line; see input argument 11) of the azimuth cable wrap. If the user does not specify a preference, and the trajectory for the specified observation can be accommodated by either wrap of the cable wrap, then this program will select the wrap which provides the fastest acquisition time given the "current" position of the telescope azimuth (as specified on the command line; see input argument 7). However, for some observations, the choice of azimuth wrap is dictated by the trajectory associated with the specified observational parameters; in those cases, using the user-specified wrap may not be possible.

The program will also compute the estimated number of seconds it will take to reposition the telescope and dome shutters from their current positions (as specified on the command line) to the ones required for tracking the given target for the specified period of time starting at the specified start time of the observation. The program also computes two other quantities related to whether or not the specified target will remain visible for the entire length of the specified observation:
'timetorise' (output parameter 13) and 'timetoset' (output parameter 14); see below for details.

Note that if the specified target is not feasible to observe (starting at the specified started time and for the indicated observation length), then the estimated time to acquire the target will be set to a ridiculously large (i.e., > 86400) number so as to indicate that this observation should not be attempted at this time.

The windshield.c executable takes the following 11 command-line inputs:

argv[1]: (targra) RA of the target (in radians)
argv[2]: (targdec) Dec of the target (in radians)
argv[3]: (last) local apparent ST (in radians) at observation start
argv[4]: (length) observation duration (integer seconds) - defaults to 3600
argv[5]: (prefcfg) preferred shutter config: 0=split-shutters, 1-up&over, 2-none
argv[6]: (verbose) specify the verbosity level: 0=quiet, 1=some, 2=lots
argv[7]: (aznow) Current telescope azimuth (in degrees)
argv[8]: (elnow) Current telescope elevation (in degrees)
argv[9]: (fsnow) Current dome front-shutter-panel position (in degrees)
argv[10]: (rsnow) Current dome rear-shutter-panel position (in degrees)
argv[11]: (wrap_pref) azimuth wrap preference: -1=lower +1=upper, 0=no preference

It outputs to stdout (white-space separated) the following 15 values:

fsmove (in degrees) - angle to which front shutter should be pre-positioned
rsmove (in degrees) - angle to which rear shutter should be pre-positioned
minel (in degrees) - minimum elevation angle of observation trajectory
maxel (in degrees) - maximum elevation angle of observation trajectory
minaz (in degrees) - minimum azimuth angle of observation trajectory

maxaz (in degrees) - maximum azimuth angle of observation trajectory
startel (in degrees) - starting elevation angle of observation trajectory
endel (in degrees) - ending elevation angle of observation trajectory
startaz (in degrees) - starting azimuth angle of observation trajectory
endaz (in degrees) - ending azimuth angle of observation trajectory
shutcfg - shutter configuration: 0=split-shutters 1=up-and-over -1=ERROR
acqtime - (in seconds) - estimated time to acquire target from current pos'n
timetorise (in seconds) - number of seconds before target becomes visible
timetoset (in seconds) - number of seconds before target becomes unvisible
wrap_cfg - az wrap configuration: -1=lower wrap 0=middle +1=upper wrap

Note that 'acqtime' will be >= 86400 if specified observation is not possible for any reason (e.g., object is not yet up, observation would exceed cable wrap limits, etc.)

Note that 'timetorise' should be 0, indicating that the object will be visible at the start of the observation. If > 0, this indicates that the object will not become visible until 'timetorise' seconds after the specified observation would have commenced.

Note that 'timetoset' should be -1, indicating that the object will not become unvisible (i.e., drop below the minimum elevation angle at which it would be visible) during the observation. Otherwise, if >= 0, this indicates that the object will become unvisible 'timetoset' seconds after the specified observation would have commenced.

3.1.5 Loose Ends

For the most part, the wind shielding strategy implemented by the windshield.csh script works very reliably. Retry logic has been incorporated which enables the script to recover gracefully from the occasional failure of the dome shutter panels to arrive at their commanded positions. In most cases, that logic is successful and the robot.csh script runs all night without need for human intervention.

Unfortunately, it appears that the position that is reported from the front dome shutter panel is occasionally inaccurate, i.e., the position that is reported may be offset by as much as a few degrees from the actual position of that panel. This can sometimes confuse the wind shielding software if that inaccurate position conflicts with the readings of other sensors on the shutter panels.

We suspect that these intermittent inaccuracies in the position reported for the front dome shutter panel are the result of the “home switch” for that panel being misaligned relative to the mechanism that activates it. That misalignment appears to sometimes cause false triggers of the home switch, which in turn messes up the reported position for that shutter panel. Once the new “man lift” arrives on Mt. Hamilton in early 2014, we hope that it can be used to access that mechanism so that it can be re-aligned. If that home switch can be made to trigger reliably, it should restore the accuracy of the reported positions for the front shutter panel.

3.2 **autoexposure.csh**

This script is used to determine the appropriate guide camera settings that will yield a properly-exposed image of the target object in the guide camera. It is invoked from robot.sch after the target object has been acquired and is within the guide camera's field of view.This version assumes the guider camera's normal (i.e., non-charge-multiplying amplifier) readout amplifier is being used and that readout window is set to "full frame".

The script determines the guide camera's exposure level by monitoring the NFOUND and FLUX keywords that are broadcast by the apfguide process. The values of those keywords are determined by the 'sextractor' program, which apfguide invokes to analyze each successive image from the guide camera.

Sextractor will attempt to detect any targets that are within apfguide.MAXRADIUS pixels of the autoguider's reference position; that position is specified by the apfguide keywords GUIDEX and GUIDEY, which normally specify the (x,y) coordinates of the center of the currently-selected slit. (The values of GUIDEX,GUIDEY are updated by the autoslitbox.csh script whenever the position of the decker stage or the iodine cell stage changes so as to ensure that they reflect the projected center of the currently-selected slit as viewed by the guide camera.)

Apfguide will set its NFOUND keyword to indicate the number of objects that sextractor detected (within MAXRADIUS pixels of the slit center) in the current image from the guide camera. If NFOUND is > 1, then apfguide will sets its FLUX keyword to indicate the flux that it measured for the brightest object it found within the specified MAXRADIUS.

If NFOUND is 0, that means that sextractor did not detect any objects within that radius, in which case the value of the FLUX keyword will not be meaningful for that image. Usually, if NFOUND is 0 that means that there are no detectable objects within MAXRADIUS pixels of the slit center. However, if an object is extremely bright, it can saturate the guide camera so badly that sextractor will not detect it and will erroneously report that there is no object present. To resolve that ambiguity, the script will check apfguide's NPTHRESH keyword, which reports the number of saturated pixels (i.e., pixels whose value exceeds apfguide.PTHRESH) in that guide camera image. If NFOUND is 0 and NPTHRESH exceeds 25, then the script will conclude that the guide camera image is badly saturated and the effective exposure time needs to be reduced.

Unlike most guide camera systems, APF's guide camera does not have in its optical path a filter wheel with ND filters that can be used to attenuate the signal from bright objects. Instead, to prevent saturation by bright objects, APF's guide camera uses sub-second exposure times. However, such sub-second exposure times result in higher frame rates. For example, setting a guide camera exposure time of 0.1 seconds would result in the eosgcam process delivering images to the apfguide process at a rate close to 10 Hz, and apfguide can't keep up with images arriving at such a high rate.

To prevent that problem, when sub-second exposure times are specified, eosgcam's SUMFRAME keyword is used to specify that some number of images should be summed together within eosgcam so that a summed frame is delivered to apfguide at a more modest rate. Also note that each guide camera frame takes about 65 milliseconds to read out of the camera, and this will limit the actual unsummed frame rate for images with sub-second exposure times.

The script adjusts the following guide camera parameters in an attempt to obtain an acceptable exposure level:

guide camera exposure time (eosgcam keyword GEXPTIME)
guide camera pre-amplifier gain (eosgcam keyword GCGAIN)
guide camera frame summing (eosgcam keyword SUMFRAME)

Currently, it uses the following algorithm, which works reasonably well but is not as time-efficient as possible, especially for very faint objects. The script starts out by setting the these initial camera settings:

exposure time = 1s
preamplifier gain = low
frame summing = 1

If sextractor's analysis of the guide camera image finds no objects, the script will check to see if the image had more than 25 saturated pixels. If so, it will conclude that the object is too bright, and that the camera's exposure time needs to be decreased to prevent its CCD from saturating (see below). Otherwise:

If nfound==0 or the flux is < 1000, then on subsequent images the preamp gain is successively increased to the next higher level (i.e., from "low" to "medium" or from "medium" to "high").

If the preamplifier gain is set at "high" and either nfound==0 or the flux is < 1000, then on subsequent images the exposures time is successively incremented by 1s, up to a maximum of 5 seconds.

If the preamplifier gain is at "high", the exposure time is at 5s, and either nfound==0 or the flux is < 1000, then on subsequent images the frame summing is successively incremented by 1, up to a maximum of 6 (yielding a total effective exposure time of 30s).

If the frame summing is incremented to 6 and nfound==0 or the flux remains low, then the script concludes that either the object is too faint or the clouds are too thick and that it is not possible to obtain a reasonably-exposed image of this target; it will return an exit status of 1 to signal failure.

In the case of brighter objects, if either:

a) apfguide.NFOUND == 0 and apfguide.NPTHRESH > 25, or
b) apfguide.NFOUND > 1 and apfguide.FLUX > eosgcam.SUMFRAME * 15600
c)
then the script concludes that the exposure level is too long and needs to be reduced.

Specifically:

If the object's flux exceeds sumframe*15600, the script first sets frame summing to 6 and the camera exposure time to 0.1 second.

If the object's flux still exceeds sumframe*15600, then the script will set frame summing to 8 and the exposure time = 0.05 seconds.

If the object's flux still exceeds sumframe*15600, then the script will set frame summing to 12 and the exposure time = 0.010 seconds.

If the object's flux still exceeds sumframe*15600, then the script concludes that the target is excessively bright but will treat this condition as acceptable for guiding and will report success. Although not ideal, the guider can guide on a moderately saturated image.

In clear skies, this algorithm appears to function effectively for targets spanning a visual magnitude range of 2.0 to 15.0.

However, as noted above, the current approach requires an excessive amount of time to determine the appropriate exposure time for targets at the faint end of the range or for brighter targets observed during periods of relatively thick cloud cover. The script could be made more efficient by successively increasing the effective exposure time by multiplies of 2 (i.e., 1s, 2s, 4s, 8s, 16s, 32s), rather than in its current linear fashion (i.e., 1s, 2s, 3s, 4s, 5s, 10s, 15, 20, 25, 30s).

The script also assumes that the sky transparency remains reasonably constant during its attempts to determine the proper exposure settings. If scattered clouds happen to be blowing through, that assumption can be violated. To avoid getting stuck in an endless cycle of exposure time increases and decreases, the script will give up and report failure after it has made 15 attempts to determine the correct settings.

Finally, in addition to trying to determine the correct guide camera settings for the current target, the script will set the eosgcam TODISK keyword to ‘true’ to initiate recording of a guide camera “movie” in those cases where sextractor fails to detect any objects (i.e., NFOUND=0) in the guide image; it will set TODISK back to ‘false’ prior to exiting the script. This will result in a movie being recorded (for diagnostic purposes) to host dresden’s /tmp directory during the script’s attempts to determine the camera settings for this target.

3.3 **centering scripts**

The centering scripts are used to request the apfguide process to command offsets to the telescope position so as to center the image of the guide star on the currently-selected spectrometer slit.

3.3.1 centerup.csh

The centerup.csh script sets the apfguide.MAXRADIUS keyword as specified on the command line that invoked it. It then sets the apfguide mode to ACQUIRE. The apfguide process should then lock onto the brightest target that is within MAXRADIUS pixels of the slit center, determine its offset from the slit center, and command the telescope to offset in the opposite directions so that the image of star will be driven onto the center of the slit.

The script then invokes the centerwait.csh script, which will monitor the image statistics published by apfguide for the subsequent image from the guide camera so as to confirm that the image of the star did in fact land within 0.75 arcseconds of the slit center and remained within that radius for two successive images from the guide camera. Upon return from centerwait.csh, the script will reset the apfguide mode to GUIDE and will set the apfguide keyword CLEARSUM to “now”; that will cause apfguide to zero out it various keywords that keep track of the average and RMS values of various image statistics over the course of tracking the current target.

3.3.2 centerwait.csh

As noted above, the centerwait.csh script monitors the image statistics published by apfguide so as to confirm that the image of the target object is adequately centered (and stable) on the spectrometer’s slit. It is invoked from two different locations within the robot.csh script:

a) immediately after a successful return from the autoexposure.csh script
b) immediately after completion of the check_focus/measure_focus sequence (because guiding is deliberately disabled during that sequence)

3.4 **specify.csh**

This script simplifies the process of setting target coordination information into the relevant keywords in the eostele service used for specifying slew in the RA/Dec coordinate reference frame. It sets these keywords but does NOT initiate the slew:

targra -	target RA (J2000, epoch 2000, equinox 2000)
targdec -	target DEC (J2000, epoch 2000, equinox 2000)
targmuar -	proper motion in RA (units of milli-arcsec/year)
targmuar -	proper motion in DEC (units of mili-arcsec/year)
targplax -	parallax (in arcsec)

3.5 **slewlock.csh**

This script provides a shorthand method for slewing to a specified target, adjusting the guide camera parameters (by invoking autoexposure.csh) and centering the target on the slit (by invoking centerup.csh). It accepts coordinates for “target” objects (whose proper motions are specified in units of milli-arcsec/year) or “reference” objects (whose proper motions are specified in the units used in the EOST-supplied StarCatalog.dat file. The coordinate type (“target” or “reference”) and the coordinates themselves are specified on the command line as described in the comments at the top of the script.

This script is invoked from the robot.csh script only for those targets whose entry in the starlist file contains a non-blank value for the “do=” parameter, i.e. , for those targets that are flagged as having a nearby neighbor of comparable or greater brightness. In such cases, slewlock.csh is first use to acquire the pointing reference star, and is then used to acquire the target object itself.

In typing up the documentation for this command, I note that it does not include logic to retry the slew attempt in the case of a transient slew failure (as is done for those slews that are initiated from the windshield.csh script). We should fix that, and confer on the changes that should be made to this script to include that slew retry logic.

3.6 **camread_process_recovery.csh**

This script is invoked from the robot.csh script in order to check for, and if necessary, recover from camread process failures of the APF UCAM CCD data taking system. The apfucam software uses its COMBO_PS keyword to signal when such camread process failures have occurred. The robot.csh script invokes this script prior to starting each APF UCAM CCD exposure. If this script determines that a camread process failure has occurred, it will take the actions needed to recover from that failure, including:
a) shutting down the APF UCAM CCD data taking system that runs on host warsaw,
b) if that leaves behind zombie process, then remotely rebooting host warsaw,
c) restarting the data taking system on host warsaw within VNC desktop mainz:5

3.7 **robot_power_cycle_ucam.csh**

This script is invoked from the robot.csh script if the APF UCAM CCD controller locks up and needs to be power cycled. The APF UCAM CCD data taking system detects that condition and reports it be setting the apfucam keyword CTALKTO to a value other than “okay”. The robot.csh script checks the value of that keyword at various points in the CCD exposure process; if it finds that CTALKTO != “okay”, then it will invoke this script, which will carry out the sequence of operations needed to power cycle the CCD controller. The robot.csh script will keep a count of the number of times this script is called, and will exit with an error is this script is invoked more than 5 times within a given instance of the robot.csh script. (Hmmm...this limit should be per night. Fix!)

3.8 **robot_power_cycle_fousb.csh**

This script is nearly identical to the robot_power_cycle_ucam.csh script, except that instead of power cycling the APF UCAM CCD controller, it will power cycle the APF UCAM fiber optic USB (fousb) box that interfaces the fiber optic connection from the CCD controller to the USB interface on host warsaw. At present, this script is unused, because it is not entirely clear under what circumstances it should be invoked. At least for APF's existing APF UCAM system, we have not experienced any problems with its fousb box for the last 1.5 years.

3.9 **dome_move.csh**

This is a "wrapper" script that is used to "wrap" any 'modify -s eosdome' commands that would initiate motions of any mechanisms (such as the dome shutter panels) that are part of the eosdome. This script pre-dates the 'safemodify' command, and hence it duplicates the constraints imposed by that command. However, this script also imposes some additional constraints beyond those imposed by the safemodify command, namely, that the dome shutters should already be in a "not-closed" state and that the telescope's elevation angle should be < 16.0 degrees. This first added constraint ensures that the dome shutters are first opened up via a script such as 'openatsunset' and not as a result of an inadvertent (premature) invocation of the robot.csh script. The second added constraint ensures that the dome shutters are not moved across an up-looking primary mirror.

3.10 **tele_move.csh**

This is a "wrapper" script similar to dome_move.csh, except that it is used to wrap any 'modify -s eostele' commands that would initiate motions of any mechanisms that are part of the telescope, such as the azimuth or elevation axes. This script also imposes additional constraints beyond those imposed by the safemodify command. In particular, it requires that the shutters must be in a non-closed and stationary (i.e., "in-position") state and that the primary mirror cover must be open. Slewing the telescope with the mirror cover closed is to be avoided since the telescope is out of balance when the mirror cover is closed.

3.11 **inst_move.csh**

This scripts wraps any 'modify -s apfmot' commands that would initiate motions of any mechanisms that are of the APF instrument, such as the stage for the atmospheric dispersion compensator (ADC). At present, this script only checks to see that the checkapf keywords INSTR_PERM and INSTRELE are both true. It might be the case that this wrapper script may be redundant, since these checkapf keywords might now be checked within the apfmot service itself. (Check with Kyle on this.)

3.12 **slew_retry.csh**

This script is invoked only in the case where an attempted slew of the APF Telescope does not complete within the expected period of time, as might occur if the servo drives faulted out. Such a fault might be due to some transient failure or might indicate a more serious problem (e.g., a blown servo amplifier). If a transient failure, a subsequent slew attempt will succeed.

We observe such transient slew failures about once every one to two months. Such failures are cleared simply by trying another slew attempt. To recover from such transient failures, this script will attempt to clear the slew fault condition by first making a very small slew (0.1 degrees in az and 0.1 degrees in el) from the telescope's current position. If that slew fails, the script will give up and signal failure. Otherwise, if that small slew succeeds, then the script will retry the original slew that failed, and signal whether it succeeded or failed.

Since these transient failures occur so rarely, such slew retries should seldom be needed, and we do not expect them to be needed more than once per night. If more frequent slew retries are needed, that probably indicates a serious problem, and observing should not continue.

To prevent such slew retries from being attempted more than once per night, the time at which the last slew retry was performed is recorded in the scriptobs_var_2 keyword in the apftask service. This script will not attempt to perform a slew retry operation unless the last slew retry occurred more than 12 hours ago.