CCDSH

A command line shell for telescope and camera control

Latest version: `0.8.0` (2011.08.24)

Software and documentation: written and maintained by András Pál (`apal@szofi.net`)

Contents

1 Introduction

The program `ccdsh` is a modular command line interface (CLI) for controlling hardware elements related to astronomical observations: CCD cameras, telescope mount, dome and other optional related stuff (e.g. focuser, meteorological data acquisition devices and so on). This program is a user-interface fronted to various modules and it is aimed to provide a simple shell for controlling these pieces of hardware from the same environment as well as making an inter-connection between them according to the requirements of the observations.

This document describes briefly the main features of this `ccdsh` program. Section 2 summarizes the requirements and other general properties of the program. Section 3 describes the basic usage of the program and the commands that are routinely used throughout the observations. Section 4 discusses the startup procedure of the `ccdsh` program (i.e. what happens if a user starts his or her own `ccdsh` session from the command line). In the Appendices, the features of some additional modules are described (Section A) and some aspects of the currently installed `ccdsh` environments are summarized (Section B).

2 Software, hardware environment and requirements

By default, `ccdsh` do not provide any specific control for the hardware elements, thus `ccdsh` acts like a simple command-line ephemeris service program. The hardware elements and other features are managed via external modules that are usually loaded upon the initialization of `ccdsh`. However, independently from the actual type of the peripheral device, the generic control commands are the same in all environments. Using such modules, the features of `ccdsh` can be extended both by adding another hardware components and by adding functionalities aiding the telescope operation itself.

2.1 Hardware modules

The currently supported classes of hardware elements that can be controlled by `ccdsh` are the following:

- imaging detectors (CCD cameras);
- telescope mounts;
- domes;
- filter sets and filter wheels; and
- focusers.

Although it is not necessary to know the underlying software stack of the hardware controlling programs (beyond these modules), it is worth to have some information about it since failures and errors are easier to be tracked and resolved if one knows how the system works in general.

2.2 Operation modules

The currently supported classes of modules that add other operational functionalities to `ccdsh` are the following:

- command modules (making new commands available in addition to the built-in `ccdsh` commands);
- object name resolves (i.e. adding support to figure out current coordinates of moving or fixed celestial objects); and
- external loggers (i.e. logging the activities done with `ccdsh`).

As we will see later on (Sec. 4), these modules (both control modules and operation modules) are loaded during run-time using the command ‘`module`‘ (see also Sec. 4.1).

3 General usage and built-in features

The program `ccdsh` and all of the related modules and servers are installed to a control computer. Mostly, this computer has a public SSH access as well, so authorized users can access the program and data and control the whole observation process remotely.

The camera, dome, telescope mount, focuser and filter wheel are controlled via a command line shell (provided by `ccdsh`) by issuing simple commands for the specific hardware elements. There are both higher and lower levels of commands that can be used for observing and writing simple scripts. For instance, the full dome position control is implied via the commands related to the telescope mount motion commands. Namely, these higher level commands that slew the telescope to a given celestial position automatically computes the respective dome azimuth and moves the dome accordingly (so the telescope will look through the dome slit). However, commands on a lower level allows the user to set the dome slit position azimuth by hand or off from the telescope pointing. This is useful while taking dome flats or when the dome is moved to a home position after the observations.

The images acquired by the camera are stored and saved in the well-known common FITS format. In order to display these images, `ccdsh` communicates with the program `DS9`.

3.1 Getting started: Launching `ccdsh`

The program `ccdsh` can be started from any terminal, by typing '`ccdsh`' to the command line prompt. After a successful start of the program, we got a prompt,

```
CCD>
```

and the commands should be typed right after this prompt. The initialization process might require a few seconds (depending on the actual installation and hardware/software environment), so we might have to wait a littlebit after launching `ccdsh`.

Any time while running `ccdsh`, external system shell commands can be issued by preceding the command by an exclamation mark. For instance, starting `DS9` from the `ccdsh` environment is simply:

```
CCD> !  ds9 &
```

In the following, this section describes briefly the available built-in commands.

3.2 The `help` command

This command shows the list of the available other commands (in some random order, currently). A more detailed description for a given command can be asked by typing the name of the command itself right after '`help`':

```
CCD> help get
usage:   get dome {azimuth|status|slit}
        get mount {position|setpoint|status} [-d|--degrees]
        get ccd status
        get focus
        get temperature
        get filter
        get location [-x]
        get epoch
        get modes
```

3.3 The `status` command

The status command prints the status the currently loaded modules and module nodes:

```
CCD> status
CCD:       ready # 0 idle
Mount:     error # -1 unreachable dtime=48681
Dome:      error # -1 unreachable 0x0
Filter:    busy  # 1 busy (wait_wgotox)
Focuser:   error # -1 unreachable
Logger:    ready # 0 idle
Logger:    ready # 0 idle
Resolver:  ready # 0 idle
Resolver:  ready # 0 idle
Command:   ready # 0 idle
CCD>
```

Module nodes marked with the green '`ready`' are up and ready to be used, nodes marked with yellow '`busy`' are currently in operation and expected to be ready soon while the nodes marked with red '`error`' have some kind of error that does not allow the given functionality to be used without dissolving the problem. This brief status is followed usually by the full status message as reported by the module. The first number of this status message is always an integer status code. It is zero, positive or negative in the cases of '`ready`', '`busy`' and '`error`', respectively. The reasons of the errors are likely to be figured out from the full status message, that is followed by the status code.

In order to start the observations successfully, all of the modules and module nodes must be in a '`ready`' status. Usually, modules are loaded upon initialization (see Sec. 4) by the command '`module`' (see Sec. 4.1 for more details). Usually, the end-user does not really manage the modules directly. In general, the system administrator sets up the global startup scripts (Sec. 4.2), in which the modules are loaded. However, it is recommended to test the status of the modules at least before the observations.

3.4 The `date` command

This command prints the current date and time information for the current observing location:

```
CCD> date
LD=2010.11.02 LT=16:32:39.1 UT=15:32:39.1 JD=2455503.147674 ST=19:36:05.6
```

Here, `LD` is the local date (in `YYYY.MM.DD` format, years, months and days, respectively), `LT` is the local time, `UT` is the universal time, `JD` is the Julian day and `ST` is the sidereal time for the current location.

3.5 The `sky` command

This command computes and prints the celestial and apparent coordinates of celestial objects. By default (i.e. simply typing '`sky`'), the program reports information about the Sun and the Moon:

```
CCD> sky
# Object                RA        Dec          tau       Alt       Az
Sun              14:31:46.6 -14:55:35 +11:14:27.2   -56.11 +159.99
Moon             11:36:17.5 -03:27:45 -09:50:03.8   -37.85 -137.24
```

By specifying the switch `-p` or `--planets`, the program prints information for the planets:

```
CCD> sky --planets
# Object                  RA       Dec          tau      Alt       Az
Mercury          15:12:10.6 -18:53:15 +10:34:57.1   -56.57  +141.48
Venus            13:54:29.7 -17:01:23 +11:52:38.0   -59.51  +176.53
Mars             16:07:45.0 -21:31:57 +09:39:22.7   -51.68  +120.25
Jupiter          23:40:04.4 -03:48:17 +02:07:03.3   +31.64   +38.09
Saturn           12:46:26.0 -02:35:27 -10:59:18.4   -43.24  -158.97
Uranus           23:50:03.4 -01:55:09 +01:57:04.3   +34.38   +36.30
Neptune          21:52:45.4 -13:23:28 +03:54:22.3    +9.90   +57.44
```

If the name or alias for a pre-defined object follows the command line switch of `-j` or `--object`, these information are reported for this particular object:

```
CCD> sky --object m31
# Object                  RA       Dec          tau      Alt       Az
m31              00:42:44.3 +41:16:09 +01:05:28.6   +76.79   +67.99
```

Before slewing the telescope to a given object, it is recommended to check the visibility of the object by comparing the apparent coordinates (horizontal altitude and azimuth) with the telescope mount and dome limitations.

3.6 The `get` command

This command allows us to obtain information about the CCD camera, telescope mount and dome as well as some other data that highly influence the observations.

3.6.1 Chip temperature and cooling: `get temperature`

The command '`get temperature`' reads the current CCD cooling parameters, such as the actual chip temperature, the expected (set) temperature, ambient temperature (if available) and the cooling power:

```
CCD> get temperature
ccd=-9.8 amb=25.00 set=-9.8 power=16%
```

3.6.2 Focuser: `get focus`

This command simply prints the current position of the focuser:

```
CCD> get focus
25.55
```

The units of the focuser position might depend on the actual telescope system. The focuser can be altered using the command '`set focus ...`' (see Sec. 3.7.2).

3.6.3 Filter wheel: `get filter`

The command '`get filter`' reads the current filter wheel position. The position is an integer number that must be between 1 and N_{filter} (that is the currently available slots in the filter wheel). If the filter wheel is moving, the position is off from this interval (in principle it should be zero). The position and status codes are followed by the filter aliases, that are easy-to-remember names of the filters.

```
CCD> get filter
position=2 name="V" name="Bessel V"
```

See also the command '`set filter`' about more details on switching between different filters.

3.6.4 CCD readout modes: `get modes`

The command '`get modes`' asks the camera and chip capabilities about different exposing and readout modes. These information include the internal mode identifier, the gain parameters, the effective resolution and the effective pixel sizes. The binning parameters of the image acquisition commands (see also the commands '`acquire`' and '`sequence`' later on) must be in accordance with the available readout modes listed here:

```
CCD> get modes
# id sx   sy   gain  px     py
00 4008 2672 0.78  9.00  9.00
01 2004 1336 1.56 18.00 18.00
02 1336  890 1.56 27.00 27.00
03 4008    0 0.78  9.00  9.00
04 2004    0 1.56 18.00  9.00
05 1336    0 1.56 27.00  9.00
06 4008 2672 0.78  9.00  9.00
07 2004 1336 1.56 18.00 18.00
08 1336  890 1.56 27.00 27.00
09  445  296 1.56 81.00 81.00
```

3.6.5 Dome: `get dome`

The command

```
CCD> get dome azimuth
```

reads the current dome azimuth and print it (in degrees). The command

```
CCD> get dome status
```

checks the current status of the dome. If the dome does not do anything, the status is "idle". If the dome tracks a given celestial position, the status is "tracking". If the dome has not reached the desired azimuthal position, its status is "rotating".

If the dome driver has support for opening and closing the dome slit door, the status of this slit door can be checked with the command

```
CCD> get dome slit
```

This command prints the dome slit door position, that is usually a number between 0 and 1. 0 means that the dome is fully closed (implying that the whole instrumentation is protected from the weather) while 1 means that the dome is fully opened (thus, it is ready to use and the observations can be started). Fractions between 0 and 1 mean that the dome slit door is partially opened only and might be in motion at this time. Any other values (negative ones, mostly) or status messages other than a real number between 0 and 1 (inclusively) imply that there is a failure with the dome slit door.

3.6.6 Telescope mount: `get mount`

The command

```
CCD> get mount position
```

reads the current celestial position of the telescope mount while the command

```
CCD> get mount setpoint
```

prints the current setpoint of the telescope. See also the commands '`slew`' and '`match`' for more details about telescope positioning.

The command

```
CCD> get mount status
```

checks the current status of the mount. It is useful while waiting for a slewing to be completed. The mount should be "idle" before real observations start.

3.6.7 The current epoch: `get epoch`

The command '`get epoch`' can be used to ask the currently implied epoch value. By default, this is 2000.0, so both the input coordinates (i.e. object definition coordinates, slew target coordinates) and the output coordinates are treated as J2000.0 coordinates:

```
CCD> get epoch
2000.000
```

The default epoch can be altered by the '`set epoch`' command.

3.7 The `set` command

Using this command we can either explicitly control the CCD camera, the telescope mount and the dome and/or imply some automatic features.

3.7.1 Temperature regulation: `set temperature`

This command sets the regulated temperature value:

```
CCD> set temperature -30
```

Due to the thermal inertia of the CCD chip, the regulated temperature is not obtained instantly and one has to wait a few minutes until this temperature is reached. After the regulation is set, one can track the current temperature by issuing the '`get temperature`' command. In order to have a stable thermal environment, it is worth not to go above a regulating power of $60 - 70\%$. If the power is above these limits, even very small increments in the ambient temperature[1] may lead to instabilities and the desired temperature regulation setpoint cannot be reached and more importantly, the actual chip temperature will not be constant at all.

In order to completely switch off the temperature regulation, use the command

```
CCD> set temperature off
```

3.7.2 Focuser: `set focus`

This command allows the alternation of the focuser (focal plane position). The tokens '`set focus`' must be followed by the desired focus position, in the units of the current system:

```
CCD> set focus 25.65
```

Use the command '`get focus`' to check if the position setpoint has been reached.

[1] That can be caused by the camera itself if the air or water circulation is insufficient!

3.7.3 Filter wheel: `set filter`

The actual filter can be set by the '`set filter`' command. By default numeric filter identifiers can be specified here, however, by defining filter aliases (see also '`define filter ...`', below), one can use more descriptive identifiers for the filters. Note that the filter wheel always needs a little time to move to position, thus the current status of the filter positioning should be monitored with '`get filter`' before issuing any serious command.

```
CCD> set filter V
```

3.7.4 Low level mount control: `set mount`

By default, the movements of the telescope mount is controlled by the '`slew`' command. In order to control the mount on a lower level, some commands are available starting as '`set mount ...`'.

If the current slewing operation should be stopped due to some reasons, the mount can be stopped using the command

```
CCD> set mount stop
```

If the mount driver supports turning off or on the sidereal tracking, use the command

```
CCD> set mount track off
```

or

```
CCD> set mount track on
```

to turn off or turn on the sidereal tracking, respectively.

If a little time is needed after the mount slew is completed (for instance, the mount driver does not support it), one can use the command like

```
CCD> set mount relax 5
```

to make `ccdsh` wait 5 seconds after the last slew operation is completed and before the next image exposure starts. By default, this relax time is zero, but sometimes it is set to a few seconds in the global startup script.

3.7.5 Dome: sidereal tracking, azimuth and slit door control: `set dome`

Automatic sidereal tracking By issuing the command

```
CCD> set dome auto on
```

`ccdsh` will move the dome slit automatically so it will always follow the telescope. In most of the cases, this is fine. The this automatic behavior can be turned off by issuing

```
CCD> set dome auto off
```

Manual rotation If the dome position should be adjusted manually, first switch off this automatic movement (`set dome auto off`). In order to move the dome to a given azimuth, use the command

```
CCD> set dome azimuth=<A>
```

where `<A>` is the desired azimuth angle in degrees (0° is south, 90° is west, −90° is east and so on). The dome rotation can explicitly be stopped by issuing

```
CCD> set dome stop
```

anytime.

Manual start of sidereal tracking The dome control mechanism is capable to track fixed celestial targets by issuing '`set dome track`', followed by the declaration of the celestial position. See also `slew` and `match` commands how such celestial positions are defined in `ccdsh`. For example, for asking the dome controller to follow the apparent location of the object M31, use

```
CCD> set dome track -j m31
```

Some notes on asymmetrical telescope mounts (e.g. German equatorial mounts): it is important to know that both this explicit dome tracking and the previously mentioned automatic tracking expect that the telescope mount piering is proper. Namely, objects that are on the east side of the meridian (i.e. rising objects, before culmination) are observed from west pier side while objects on the west side of the meridian (i.e. setting objects, after culmination) are observed from east pier side. In order to override this default behavior, use the command line switches `-e` and `-w` to expect east pier side and west pier side, respectively. Note that objects that have been started to track before culmination are observed in west pier side, however, following culmination (at $\tau = 0$) the telescope mount does not switch between the pier sides automatically and continue the tracking in west pier side. This might yield an unexpectedly large telescope slew (due to switching between the pier sides) even for a very small absolute mount position correction. Of course, if the dome driver knows that the telescope mount is a symmetric (e.g. fork) mount, these command line switches (`-e` and `-w`) are simply ignored.

The dome tracking can be turned off by issuing '`set dome stop`' and automatic dome movements (see '`set dome auto on`') also override the latest dome track command.

Slit door control If the dome driver has support for controlling the dome slit door, it can be opened using the command

```
CCD> set dome slit open
```

Similarly, the dome slit door can be closed using the command

```
CCD> set dome slit close
```

while the command

```
CCD> set dome slit stop
```

stops any motion of the dome slit door. We refer here to the command '`get dome slit`', that can be used to check the dome slit door status (Sec. 3.6.5).

3.7.6 Epoch: `set epoch`

If the default epoch of J2000.0 must be changed during run-time, use the command

```
CCD> set epoch 1950.0
```

or something similar. This little example above sets the current epoch to B1950.0. Note that the coordinates reported by the commands like '`get mount position`' also depend on the current epoch, so altering the epoch might yield different output coordinates even if the telescope does not move at all.

3.7.7 User interface settings: `set verbosity` and `set xpa`

Verbosity These commands can be used to change some of the user interface behaviors. The command '`set verbosity ...`' sets the verbosity level of some of the other commands (e.g. '`acquire`' or '`sequence`'). The higher the number number given here, the more the number of messages printed by these commands:

```
CCD> set verbosity 1
```

Displaying images using DS9 and the XPA protocol The command '`set xpa ...`' can be used to turn on or off the default XPA usage: if the command

```
CCD> set xpa on
```

is issued, the subsequent '`acquire`' or '`sequence`' commands will try to use the XPA interface to display the acquired images (mainly using the program `DS9`). This can be turned off by issuing

```
CCD> set xpa off
```

as well.

3.8 The `acquire` command

With this command, one can acquire single images. The filter wheel and temperature regulation must be set and checked accordingly before issuing this command. By default, this command acquires a full frame with the highest available resolution (i.e. with a binning of 1×1 pixels), one second of exposure time and tries to display it in the `DS9` window. Of course, this default behavior is not adequate for most of the cases, so further tuning of this command is available via command line arguments:

`-h|--help`: shows a brief list of available options for the command `acquire`, with the basic syntax of these;

`-V|--verbose`: be verbose during image acquisition, so displays in the screen what is going on currently (exposing, readout, ...);

`-t|--time <time>`: the total integration/exposure time `<time>`, in seconds;

`-o|--output <file>`: the name of the output file into which the image is saved in FITS format. Note that if we do not provide such a file, the program *does not* save it, only displays in `DS9` (if available);

`-b|--bin <bx>,<by>`: the effective binning during readout will be `<bx>` $\times$ `<by>` – only those binning resolutions can be provided here that are supported by the underlying camera hardware (see also: `get modes`);

`-d|--dark`: during image acquisition, the camera does not open the shutter;

`-a|--auto-dark`: *after* acquiring a normal frame, a dark image is also acquired with the same parameterization (i.e. binning and exposure time) that is automatically subtracted from the original frame;

`-s|--size <sx>,<sy>`, `-f|--offset <x0>,<y0>`: the image is read out only from the sub-frame

[`<x0>`:`<x0>`+`<sx>`,`<y0>`:`<y0>`+`<sy>`].

this might be useful during focusing or making images with a smaller field-of-view and also yields a (proportionally) faster readout time;

`-x|--xpa`: the image is displayed in a `DS9` window, independently whether it is saved or not.

3.8.1 Some examples

Acquire an image with 10 seconds of integration time, with a binning of 3×3 and the image is saved in the file `test.fits` and also displayed in the `DS9` window:

```
CCD> acquire -b 3,3 -t 10 -o test.fits -x
```

Acquire a sub-frame in the section [400:500,1700:1800] of the chip with the highest available resolution while the image is *not* saved, just displayed in `DS9`:

```
CCD> acquire -b 1,1 -t 5 -f 400,1700 -s 100,100
```

3.9 The `sequence` command

This command is for creating a series of individual images. Although its syntax is a bit complex, one can create almost arbitrary sequences of images with various additional constraints. There are some common options with the command '`acquire`' that are also described above.

`-h|--help`: shows a brief list of available options for the command `acquire`, with the basic syntax of these;

`-V|--verbose`: be verbose during image acquisition, so displays in the screen what is going on currently (exposing, readout, ...);

`-b|--bin <bx>,<by>`: the effective binning during readout will be `<bx>` × `<by>` – only those binning resolutions can be provided here that are supported by the underlying camera hardware (see also: `get modes`);

`-s|--size <sx>,<sy>, -f|--offset <x0>,<y0>`: the image is read out only from the sub-frame

[<x0>:<x0>+<sx>,<y0>:<y0>+<sy>].

This might be useful during focusing or making images with a smaller field-of-view and also yields a (proportionally) faster readout time;

`-x|--xpa`: the image is displayed in a `DS9` window, independently whether it is saved or not.

`-n|--basename <file-base>`: the resulted images are saved in files named accordingly to `<file-base>`. This base filename is followed by the filter name and the image sequence number, each of these are separated with an underscore (_) character and the files will have an extension of `.fits`. Like the command line argument of `-o|--output` in the case of '`acquire`' command, if this command line option is omitted, the resulted images are *not* saved, just displayed in `DS9`. Be careful.

`-j|--object flat|name=<name>,ra=<RA>,dec=<DEC>,epoch=<EPOCH>`: This switch defines the properties of the observed object by implying that the camera *really* observes this celestial position. This option is merely informal: it does not have any "side effects", just write the coordinates and the name of the object in the appropriate FITS headers.

`<frame>|<N>*(<sequence>),...` This general term defines the image sequence itself. It is a comma-separated list of tags where each tag can be either an individual image or a repetition number followed by a sub-sequence. Each sub-sequence must be put between parentheses (like `(...)` this) while options for the individual frames are defined in square brackets (like `[...]` this, see below. The operations that move the telescope or alter the focus must be written between curly brackets, see also below.

`[dark|bias|time=<time>|filter=<filter>|delay=<d>|<filter>|<time>]` The syntax for acquiring a single image frame. Between the square brackets, one should specify at least the image type or the integration time if the image type is not a bias frame and optionally a filter name. For short, a single number will imply the integration time and a single alphabetic term will imply the filter name. For instance, the declarations `[filter=V,time=10]` and `[V,10]` are equivalent, both yield a light frame, using V filter with 10 seconds of exposure time. Therefore, for short, the declaration `[0]` can be used to take bias frames as well. The filter specification can either be a numeric filter identifier or a declared filter name (see also the command '`set filter`', Sec. 3.7.3). Note that if the filter is specified by the numeric filter identifier, one should explicitly use `filter=...`, since a single integer would imply the exposure time instead of a filter identifier. See some examples below (Sec. 3.9.1).

`{focus=<focus>|ra=<ra>,dec=<dec>|object=<object>}` The syntax for controlling the focus and telescope between frames. Between the curly brackets, one can either alter the focuser position using the term `focus=...` (see also the command '`set focus ...`', Sec. 3.7.2), or alter the telescope position. The latter can be done by either explicitly setting the coordinates or using a pre-defined object name. See also some examples below (Sec. 3.9.1).

3.9.1 Some examples

Create 10 bias frames with a binning of 3×3, and these bias frames are saved in the files `bias_<N>.fits`, where `<N>` is the sequence number (between 1 and 10):

```
CCD> sequence -b 3,3 -x -n bias 10*([bias])
```

Create 20 dark frames with an integration time of 5 seconds, also with a binning of 3×3. The frames are saved as `dark_<N>.fits`, where `<N>` is the sequence number (here, between 1 and 20):

```
CCD> sequence -b 3,3 -n dark 20*([dark,time=5])
```

Create a series of 2×100 images from the object M13 with a binning of 3×3 and altering between V and R filters. The exposure times are 10 and 20 seconds for these filters, respectively. The files are named as `m13_<N>_V.fits` and `m13_<N>_R.fits`, for each filter where `<N>` is the current image sequence number (between 1 and 200):

```
CCD> sequence -b 3,3 -V -x -n m13 -j name=M13 \
        100*([time=10,filter=V],[time=20,filter=R])
```

Note that this command only implies that the telescope points to the object M13, it *does not* slew to the desired position. For short, one can write simply:

```
CCD> sequence -b 3,3 -n m13 -j name=M13 100*([V,10],[R,20])
```

Observe two adjacent fields 10 times:

```
CCD> sequence -n scan 10*({ra=11:20,dec=+25},[I,10],{ra=11:20,dec=+26},[I,10])
```

Alter the focus between the filters:

```
CCD> sequence -n target 10*({focus=25.62},[B,20],{focus=25.50},[V,15])
```

3.10 The `slew` command

This command moves the telescope to the desired celestial position. The position can either be a celestial coordinate (in the form of right ascension and declination) or an alias for a planet or pre-defined object. For example:

```
CCD> slew ra=00:42:44.3 dec=+41:16:09
```

```
CCD> slew -j m31
```

```
CCD> slew -p moon
```

The first command slews the telescope to the coordinates $\alpha = 00^{\mathrm{h}}42^{\mathrm{m}}44^{\mathrm{s}}.3$, $\delta = +41°16'09"$, the second slews to the object having the name or alias `"m31"` while the last one slews the telescope to the Moon.

During the slewing of the telescope, the current position of the mount can be tracked using

```
CCD> get mount position
```

and the status of the mount slewing can be tracked using

```
CCD> get mount status
```

If the slew is completed, this position should be within a few arcseconds from the slewing setpoint:

```
CCD> get mount setpoint
```

Note that some telescope mounts have a noticeable inaccuracy in the positioning, even greater (tens of arcminutes) if the movement was large (tens of degrees or even more).

3.11 The `match` command

Some of the telescope control hardwares do not provide information about the absolute position of the mount (i.e. the hour angle and/or declination cannot be obtained without some kind of initialization and synchronization). For such mount drivers, `ccdsh` provides this `match` commands that aids this position matching between the telescope mount and the sky.

This command matches the telescope mount coordinates with the given celestial position. Like in the case of the '`slew`' command, the position can either be a celestial coordinate (in the form of right ascension and declination) or an alias for a planet or pre-defined object:

```
CCD> match -j vega
```

This command is useful at the beginning of the observations when the telescope is adjusted manually to a bright star.

Note that like most of the telescope mounts, the GTO-CPx mounts can also be controlled via an intelligent remote controller unit. If the observer uses `ccdsh` *and* this controller unit nearly simultaneously, the coordinates might not be synchronized, depending on the actual telescope mount hardware. In order to retrieve the current position of the mount and synchronize it with `ccdsh`, use the command

```
CCD> match --sync
```

or simply

```
CCD> match -y
```

This command is equivalent by reading the current position using

```
CCD> get mount position
ra=11:22:33.4 dec=+55:44:33
```

and use the `match` command with the obtained coordinates:

```
CCD> match ra=11:22:33.4 dec=+55:44:33
```

Subsequent call of the '`match`' command might be useful after larger telescope slews (see the note above at the end of the previous subsection), if the positioning is not so accurate.

Note that some telescope mount modules (e.g. the `mod_pschtcm.so` module, that controls the fork mount of the Piszkés/Schmidt telescope) have a complete support for absolute positioning, that is completely independent from the "past" of the mount positioning. Therefore, such mounts and the respective modules simply ignore these '`match`' commands.

3.12 The `resolve` command

If `ccdsh` has modules that support object name resolution, the command '`resolve`' can be used to obtain object coordinates via these modules. The command is simply followed by the name of the object, while `ccdsh` prints the object coordinates and reference epoch for this object:

```
CCD> resolve M31
M31 ra=00:42:44.32 dec=+41:16:07.5 epoch=2000.0
```

Note that a single '`resolve`' command can resolve only a single object. If more arguments are given after '`resolve`', the program simply concatenates them with spaces and use this concatenated name for the object name resolution queries. If an object is not found in any of the databases supported by resolver module(s), `ccdsh` prints a warning message:

```
CCD> resolve NGC 6823
"NGC 6823" ra=19:43:10.00 dec=+23:17:53.9 epoch=2000.0
CCD> resolve NGC 9999
ccdsh:  resolve:  no object with the name 'NGC 9999' has resolvable coordinates.
```

3.13 Changing directories: the `cd` and `pwd` commands

Like in most of the shells, in the `ccdsh` environment, the '`cd`' and '`pwd`' commands change or print the current working directory:

```
CCD> pwd
/home/apal
CDD> cd /data/apal
CCD> pwd
/data/apal
```

As of this writing, `ccdsh` does not have any built-in commands to create new directories (and like so, remove directories or files), however, one can use the plain shell itself to create new directories:

```
CCD> pwd
/data/apal
CCD> !  mkdir 20110825
CDD> cd 20110825
CCD> pwd
/data/apal/20110825
```

Since the system-level command for changing working directories alter only the directory of the currently running process, escaping to the shell and change the directory outside won't alter the working directory of `ccdsh` itself:

```
CCD> pwd
/data/apal/20110825
CCD> !  cd /data/apal
CCD> pwd
/data/apal/20110825
```

In order to manage the files easily, the user can invoke system shell commands like '`ls`' from the `ccdsh` command line. Like so, higher level utilities, such as a Midnight Commander (`mc`) or other terminal windows (`xterm`, for example) can also be started and run from the `ccdsh` prompt.

4 Startup procedure and software stack

After starting `ccdsh`, the commands found in the files

`/usr/local/ccdsh/startup.ccdsh`

and

`$HOME/.ccdsh_startup`

are executed automatically in this order. The first file contains commands for the *global startup* (that is, de facto executed for all `ccdsh` users) and the latter one may contain some local definitions that is suitable for the user who started the `ccdsh` environment (for instance, definitions for personally interested celestial objects).

4.1 The `module` command

The purpose of the '`module`' command is to load, manage or remove external modules during run-time. By simply typing the command '`module`', it lists the currently loaded modules in loading order:

```
CCD> module
mod_log_file.so      [logger]
mod_log_passive.so   [logger]
mod_pschtcm.so       [mount] [dome] [focus]
mod_resolv_sesame.so [resolver]
mod_resolv_mpc.so    [resolver]
mod_staralt.so       [command]
mod_qpaso2server.so  [ccd]
mod_ifw_serial.so    [filter]
CCD>
```

Using the command line argument `-f` or `--full-list` gives a more detailed list of information about the loaded modules:

```
CCD> module --full-list
...
mod_log_passive.so:
        Capabilities:   [logger]
        Command line:   mod_log_passive.so --port 8998
        Description:    A passive forwarding logger module
mod_pschtcm.so:
        Capabilities:   [mount] [dome] [focus]
        Command line:   mod_pschtcm.so --port 193.225.174.141:8873
        Description:    Wrapper module to the customized PSCHTCM server backend
        Documentation:  http://ccdsh.konkoly.hu/
mod_resolv_sesame.so:
        Capabilities:   [resolver]
        Command line:   mod_resolv_sesame.so
        Description:    A wrapper to the CDS Sesame resolver
        Documentation:  http://cdsweb.u-strasbg.fr/doc/sesame.htx
...
```

In order to remove running modules, use the command line argument `-r` or `--remove` for the command `module`, that is followed by the name of the module (full name of the shared object file, e.g. `mod_pschtcm.so`). Although modules can safely be removed and inserted at any time, it is worth to consult experienced users before doing so.

4.2 Global startup commands

As of this writing, these commands below are found in `/usr/local/ccdsh/startup.ccdsh` and executed inevitably for all `ccdsh` users.

This global startup script is installation-specific since it mostly loads the respective modules, initializes them and specify some constants and declarations (location coordinates, filter aliases and so on). See also the following section about more details on some environments and features.

4.3 Local startup commands

After the execution of the commands found in `/usr/local/ccdsh/startup.ccdsh`, the program seeks for a `.ccdsh_startup` file in the user's home directory. If this file exists, the commands found in this script file are also executed. This file might be used to define some additional objects that makes the whole observation easier and more convenient. For instance, a set of definitions for some well-known Messier objects can be:

```
define object M57 ra=18:53:35.1 dec=+33:01:45 epoch=2000.0 alias=m57
define object M27 ra=19:59:36.3 dec=+22:43:16 epoch=2000.0 alias=m27
define object M31 ra=00:42:44.3 dec=+41:16:09 epoch=2000.0 alias=m31
define object M74 ra=01:36:41.8 dec=+15:47:01 epoch=2000.0 alias=m74
define object M45 ra=03:47:24.0 dec=+24:07:00 epoch=2000.0 alias=m45
```

See also '`define object ...`' for further details.

Appendix

A Operation modules

This section briefly describes the modules and functionalities that are not part of the `ccdsh` main source tree, but shipped with the standard distributions, therefore available independently from the current observing and hardware environment.

A.1 Resolvers

A.1.1 The Sesame resolver

The Sesame resolver, available in the file `mod_resolv_sesame.so` as a `[resolver]` class of module uses the CDS databases Simbad, Vizier and NED via the common query interface names Sesame. See also the page at `http://cdsweb.u-strasbg.fr/doc/sesame.htx` for more details about the Sesame service.

The Sesame resources are accessed via TCP/IP and persistent internet connection is required to operate this service successfully.

A.1.2 The MPC resolver

The MPC resolver, available in the file `mod_resolv_mpc.so` as a `[resolver]` class of module uses the Minor Planet Center's "Minor Planet & Comet Ephemeris Service" to resolve coordinates of small bodies in the Solar System. See also the page at `http://www.minorplanetcenter.net/iau/MPEph/MPEph.html` for more information about the MPC and this particular service.

Since these small bodies in the Solar System moves relatively fast, all queries initiated by this module contains the current time as well, so the resolved coordinates are always up-to-date. The epoch of these coordinates are usually for J2000.0.

The MPC ephemeris services are queried via TCP/IP and persistent internet connection is required to operate this service successfully.

A.2 Loggers

The full source tree of the `ccdsh` package is shipped with two logger modules, `mod_log_passive.so` and `mod_log_file.so`. These modules log the `ccdsh` actions to clients that are connected to the logger module (as done by `mod_log_passive.so`) or writes the log entries to separate files (as it is done by `mod_log_file.so`).

The logger module `mod_log_passive.so` is passive in the following sense, by default: it does not write any log information unless TCP/IP clients are connected to its port. The default port is `*:8998`. This module is usually used by archivers: archiving scripts or programs tries to connect to this port, and if successfully connected, these scripts or programs retrieve information from `ccdsh` and do the necessary steps to archive the images acquired by the user.

A.3 The `staralt` command

With the '`staralt`' command, provided by the module `mod_staralt.so`, one is able to generate object visibility graphs, similarly to the well-known service of the IAC, as it is available on the web page `http://catserver.ing.iac.es/staralt/`. Unlike the previously described resolver modules, this module does not require an internet connection, so it just generates *something similar* to the Staralt service of IAC/ING. However, this module is able to use the resolvers available to `ccdsh`, so plotting visibility graphs for objects of which name is resolved upon calling this command might require internet connection due to the needs of the resolver itself.

This module uses the `gnuplot` plotting tool to display the graphs, therefore a proper installation of this program is also required on the host that runs `ccdsh`.

The usage of the '`staralt`' command is simple, the arguments followed by the command are the names and/or coordinates of the objects that should be plotted on the visibility graph. Each object must be a single argument, thus objects with names containing spaces must be put in quotation marks. The optional

`-n` or `--night` arguments can be used to specify other nights than the current one. The nights are specified in the format `YYYY.MM.DD`, meaning the night between this day and the *next* day:

```
CCD> staralt -n 2011.08.24 Eris "NGC 6823"
```

The graph given by this small example above are shown in Fig. 1. Note that the location information is extracted from `ccdsh`, as well as the current timezone. These can also be altered via command line arguments to the command '`staralt`', using the switches `-l`, `--location` and `-z`, `--timezone`, respectively. If the object name is replaced by two coordinates (either in traditional sexagesimal or in simply degrees), separated by comma, the visibility of this coordinate point is also plotted to the chart.

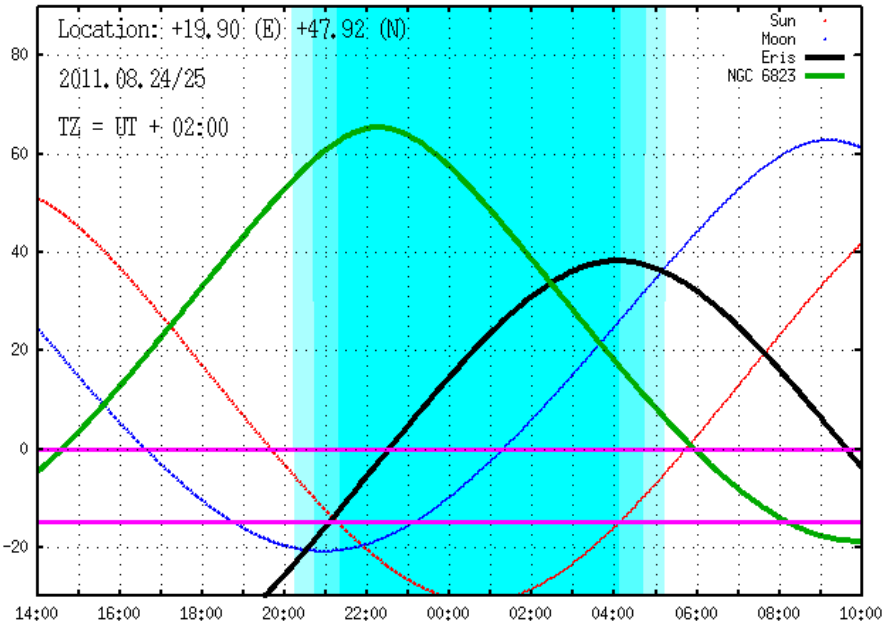

Figure 1: A simple object visibility graph drawn by the '`staralt`' command which is provided by the `mod_staralt.so` module. Note that the coordinates of the two objects given in the command line arguments, Eris and NGC 6823 are resolved by external modules (`mod_resolv_mpc.so` and `mod_resolv_sesame.so`, respectively).

B Currently installed systems and modules

This section summarizes in brief the customized features of `ccdsh` as it is installed at the Piszkéstetoő/Schmidt (PSCH) and at the Observatory of the Eötvös University in Lágymányos (ELTE).

B.1 The PSCH environment

Currently, the telescope mount, the dome and the focuser are controlled via a single serial line and the `TCM` interface language using the custom hardwares developed by László Döbrentei. This serial line is connected to the host computer `m9.konkoly.hu`, that runs a server, `pschtcm-server`, which both binds the serial port to the network and provides higher level support for accessing the telescope system. The observers *should not* login to the computer `m9.konkoly.hu`, this server runs automatically as the host computer has booted properly the Debian/Linux system.

Locally, `ccdsh` communicates with the `pschtcm-server` using the module `mod_pschtcm.so`. This module connects remotely (using a TCP/IP network socket) to the serial line host computer and initializes the dome, telescope mount and focuser accordingly. The '`module`' command should list this module with the capabilities `"[dome]"`, `"[mount]"` and `"[focus]"`, if the system works properly.

B.1.1 Global startup script

The global startup script of the PSCH system is:

```
module  -x mod_log_file.so --file /var/local/ccdsh/log/ccdsh-%Y%m%dT%H%M%S.log
module  -x mod_log_passive.so --port 8998
set     verbosity 1
set     xpa on
set     epoch 2000.0
set     location longitude=+19:53:54 latitude=+47:55:08 timezone=local
set     location fixed
module  mod_pschtcm.so --port 193.225.174.141:8873
set     dome auto on
set     mount relax 5
module  mod_qpaso2server.so
module  mod_ifw_serial.so --device /dev/ttyS0
define  filter 1 E "Clear"
define  filter 2 B "Bessel B"
define  filter 3 V "Bessel V"
define  filter 4 R "Bessel R"
define  filter 5 I "Bessel I"
module  mod_resolv_sesame.so
module  mod_resolv_mpc.so
module  mod_staralt.so
```

The first two commands loads the logger modules (a file logger and a passive client-server logger). The next 5 lines sets the verbosity and XPA usage, defines the epoch as J2000.0 and the two other lines (`set location ...`) sets the current location to the geographic longitude and latitude specified above and asks `ccdsh` to keep these location coordinates be fixed. In other words, the '`set location fixed`' command disables further alternation of the location coordinates.

The command `module mod_pschtcm.so ...` loads the specified external module that are responsible for dome, telescope mount and focuser control. If there is something wrong with these devices, some error messages are reported via `ccdsh`. In this case, the user should check other software components as well as check whether the dome and/or telescope mount control boxes are turned on and connected properly to the control computer(s).

The command '`set dome auto on`' enables the automatic dome positioning, i.e. the dome will always track the telescope position. The command '`set mount relax 5`' tells `ccdsh` to wait 5 seconds after the mount slew is completed.

As of this writing, the support for CCD camera and filter wheel is done by the modules `mod_qpaso2server.so` and `mod_ifw_serial.so`. The subsequent 5 lines define the actually employed filters. The last three '`module ...`' commands load the resolvers (Sesame and MPC, see Sec. A.1) and starts the `staralt` object visibility graph module (Sec. A.3).

B.2 The ELTE environment

This section summarizes some features of the `ccdsh` installation at ELTE. While `ccdsh` runs in the foreground and lets the user to control the observation, in the current implementations there are some servers that should run properly in the background. Although by default users launching `ccdsh` do not meet directly with these small programs, it is worth to check if these are also running properly.

B.2.1 Dome control

The control of the ELTE dome is currently done by a custom hardware element that connects the dome azimuth rotation switches to the computer as well as an image detector (web camera) that reads the current absolute position of the dome azimuth. These two are controlled simultaneously by the program `eltedm-server`. If there is some unexpected behavior for the dome (e.g. it does not work), before starting `ccdsh`, check if `eltedm-server` is running:

- this process should be listed by `ps -A`; and
- there should be a UNIX socket (that appears as a simple file) at `/var/local/ccdsh/dome.sock`.

If this dome server is not running, start it from the command prompt:

```
$ eltedm-server start
```

If this server is running or there is some error with the cable connections, this server startup command should report an error message (and does not start). In such a case, check the dome control box and webcamera connections.

Locally, `ccdsh` communicates with the `eltedm-server` using the module `mod_sdome.so`. The '`module`' command should list this module with the capability `"[dome]"` if the system works properly.

B.2.2 Telescope mount control

The low-level telescope mount control is currently implemented also in a form of a server, named `gtocpx-server`. In order to ensure that this program is running in the background, check nearly the similar things like in the case of the dome control:

- this process (`gtocpx-server`) should be listed by `ps -A`; and
- there should be a UNIX socket at `/var/local/ccdsh/mount.sock`.

If this scope mount server is not running, start it from the command prompt:

```
$ gtocpx-server start
```

Note that in cases when the telescope is not powered on, this server might also start successfully. In this case, the first real command issued from `ccdsh` might fail with an error code of 202 (`ETIMEOUT`). By default, the global startup script (see above) contains an initialization command for the telescope mount. In the case of connection or power problem, one gets the following message while starting `ccdsh` from the command prompt:

```
@enlil: $ ccdsh
ccdsh:  set:  mount:  initialization failed (error code:  202).
CCD>
```

Check cable connections and power supplies if such error codes are reported.

Locally, `ccdsh` communicates with the `gtocpx-server` using the module `mod_mount.so`. The '`module`' command should list this module with the capability `"[mount]"` if the system works properly.

B.2.3 Camera and filter wheel control

Currently, a detector with a type of SBIG STL-11000 is installed at the ELTE telescope, that also features a built-in 5-position filter wheel. This camera is controlled locally using the USB kernel interfaces as well as the user-space library `/usr/lib/libsbigudrv.so` (that depends, therefore, on `/lib/libusb-0.1.so`).

Locally, `ccdsh` loads the module `mod_ccd_stl.so` that uses explicitly the `libsbigudrv.so` library. The '`module`' command should list this module with the capability `"[ccd]"` and `"[filter]"` if the system works properly.

B.2.4 Global startup script

The global startup script of the ELTE system is:

```
set     location longitude=+19:03:44.3 latitude=+47:28:26.8 timezone=+01:00
set     location fixed

module  mod_sdome.so
set     dome auto on

module  mod_mount.so
set     mount initialize

define  filter  1 B     "Bessel B"
define  filter  2 V     "Bessel V"
define  filter  3 R     "Bessel R"
define  filter  4 I     "Bessel I"
define  filter  5 clear "Clear"
```

The first two commands (`set location ...`) sets the current location to the geographic longitude and latitude specified above and asks `ccdsh` to keep these location coordinates be fixed. In other words, the second command disables further alternation of the location coordinates.

The commands `module ...` load the specified external modules that are responsible for dome and telescope mount control, respectively. If there is something wrong with these devices, some error messages are reported via `ccdsh`. In this case, the user should check other software components as well as check whether the dome and/or telescope mount control boxes are turned on and connected properly to the control computer.

The command '`set dome auto on`' enables the automatic dome positioning, i.e. the dome will always track the telescope position (hopefully properly). The command '`set mount initialize`' initializes the telescope mount. This step might also fail if the telescope mount is not powered or not connected to the control computer.

The last five lines defines the filter aliases for the current filter wheel setup. Note that the user might alter these filter names later on, but it is highly not recommended.