COLIBRI TCS Technical Manual

Alan M. Watson

Instituto de Astronomía
Universidad Nacional Autónoma de México

6 March 2023

Contents

Chapter 1

Introduction

This is the technical manual for the COLIBRI telescope control system (TCS). It has two aims. The first is to allow the COLIBRI staff to operate the telescope conventionally, issuing commands explicitly. The second is to allow the COLIBRI staff to manage the observing blocks for robotic operations.

Chapter 2

Control System

TODO: What runs where.

TODO: Structure.

TODO: stopserver/startserver/restartserver

TODO: restartsoon (haltsoon and rebootsoon in network)

TODO: state and activity

TODO: basic requests: initialize, status, reset, stop

TODO: supervisor
request supervisor enable/disable/open/close/emergencyclose

TODO: Detectors:
expose <time> object/dark/bias/flat
setcooler on/off
movefilterwheel position/filter
movefocuser position
setfocuser position setwindow 1kx1k/2kx2k/6kx6k/default/full
setbinning 1/2/4
setreadmode mode
setsoftwaregain
focus TBD
analyze fwhm/levels

TODO: selector
request selector enable/disable
request select setunfocused

TODO: instrument
open/close

TODO: telescope
park
unpark
move HA dec

newtrack RA dec equinox
newtracktopocentric HA dec
01:23:45.67 +3h 60d
open/close

TODO: executor
open/close

TODO:
request telescope emergencyclose
sudo restartsoon
request supervisor enable

Chapter 3

JSON

JSON is used widely in TCS to represent data structures in configuration files, alert files, block files, and in inter-process communication. JSON has a simple and regular syntax that is fairly easy for humans to write, easy for computers to parse, and should be quite familiar to C, C++, and JavaScript programmers.

3.1 Dialect

The dialect of JSON used in TCS is both extended and restricted compared to standard JSON.

It is extended by the addition of comment lines that can appear wherever white-space can appear in standard JSON. A comment line is a line that begins with zero or more space and horizontal tab characters followed by two slash characters ("//"). Formally, the slash character is U+002F SOLIDUS. Other types of comments (e.g., block comments introduced by "/*" and terminated by "*/") are not allowed. Comment lines are treated as if they were empty lines. Our dialect of JSON can be converted to standard JSON by using a regular expression substitution to replace comment lines with empty lines.

It is restricted with respect to standard JSON in that it only uses array, object, and string values and does not use number, true, false, or null values. If a value represents a number, then the string representation should be used. This restriction ensures that value can be read and written without being changed.

The dialect of JSON is formally a very limited subset of JavaScript. Therefore, if you are editing data files manually, it might be useful to select an editing mode suitable for JavaScript.

3.2 Encoding

Data files should be encoded in UTF-8. ASCII is a subset of UTF-8, but ISO-8859-1 "Latin-1" is not.

3.3 Values

3.3.1 Value Types

Only array, object, and string values are used. Number and boolean values are represented by their string representation. For example, we do not write

```
0
true
false
```

but rather

```
"0"
"true"
"false"
```

Null values are not used.

3.3.2 Dates and Times

Dates and time values are written as strings containing an IS0-8601 basic date or basic combined date and time. The time zone is understood to be UTC; an explicit time zone must not be specified. The precision must not be specified more finely than seconds.

Examples of valid dates are:

```
"20101117T223815" : 2010 Nov 17 22:38:15 UTC
"20101117T2238"   : 2010 Nov 17 22:38:00 UTC
"20101117T22"     : 2010 Nov 17 22:00:00 UTC
"20101117"        : 2010 Nov 17 00:00:00 UTC
```

3.3.3 Angles

A value representing an angle may be written as:

1. a string containing a decimal number (and interpreted as *radians* not *degrees*);
2. a string containing sexagesimal notation with colons as separators and no white space;
3. a strings containing a decimal number with a unit suffix (with no intermediate white space):
   (a) “r” for radians;

(b) “h” for hours;

(c) “m” for minutes;

(d) “s” for seconds;

(e) "ad" or “d” for degrees;

(f) “am” for arcminutes; and

(g) “as” for arcseconds.

For hour angles and right ascensions, sexagesimal notation indicates hours, minutes, and seconds. For all other angles, including offsets, it indicates degrees, arcminutes, and arcseconds.

Examples of valid angles are:

`"-22.5"` : −22.5 *radians*
`"-22.5d"` : −22.5 *degrees*
`"-01:30:00"` : −1.5 hours or −1.5 degrees, depending on the context
`"0.5r"` : 0.5 radians
`"-1.5h"` : −1.5 hours
`"5am"` : 5 arcminutes
`"+20as"` : 20 arcseconds

3.3.4 Durations

A value representing a duration may be written as:

1. a string containing a decimal number (and interpreted as decimal seconds);
2. a string containing sexagesimal notation with colons as separators and no white space;
3. a string containing a decimal number with a unit suffix (with no intermediate white space):
   (a) “h” for hours;
   (b) “m” for minutes; and
   (c) “s” for seconds.

Sexagesimal notation indicates hours, minutes, and seconds.

Examples of valid durations are:

`"60"` : 60 seconds
`"00:30:10”` : 30 minutes and 10 second
`"60s"` : 60 seconds
`"10m"` : 10 minutes
`"1h"` : 1 hour

3.3.5 Validation

JSON written by hand is prone to errors such as missing commas. For this reason, it is often worthwhile checking JSON with one of the many on-line validators. The following validator is especially useful, as it tolerates comments:

https://jsonformatter.curiousconcept.com/#

3.4 Bibliography

- "Introducting JSON", https://www.json.org/json-en.html

Chapter 4

Observing Blocks

4.1 Introduction

Observations are organized as *project*, *blocks*, and *visits*.

A project is associated with a successful observing proposal. It has a PI, a name, and an identifier. It consists of one or more blocks.

A block consists of a ordered set of visits associated with a set of constraints and a program. The selector and executor deal with blocks: the selector selects a block and the executor executes it. A block is only selectable if all of its visits satisfy all of the constraints. When a block is executed, its visits are carried out in order.

A visit is typically a logical operation with the telescope such as focusing, correcting the pointing, or observing an objects.

4.2 Block Files

A block file contains a JSON representation of a block.

It consists of a single JSON object with the following structure:

```
{
  "project": {
    "identifier": <project-identifier>,
    "name": <project-name>
  },
  "identifier": <block-identifier>,
  "name": <block-name>,
  "visits": <visits>,
  "constraints": <constraints>,
  "persistent": <persistent>
}
```

As for any JSON object, the order of the members is irrelevant.

The members and values are interpreted as follows:

- <project-identifier>: The project identifier.

  This is a four-digit non-negative integer represented as a string.

  By convention, calibration programs have identifiers from 0000 to 0999, alert science programs from 1000 to 1999, and non-alert science programs from 2000 to 2999. (The pipeline uses this convention to reduce alert observations immediately but delay reducing non-alert observations until the morning.)

  This member cannot be omitted.

- <project-name>: The project name.

  By convention, we use the surname of the PI followed by a brief description of the objects (e.g., `"Watson YSOs"`).

  If omitted, the default is `""`.

- <block-identifier>: The block identifier.

  This is a non-negative integer represented as a string.

  This member cannot be omitted.

- <block-name>: The block name.

  By convention, we briefly describe the main science target (e.g., `"HL Tau"`).

  If omitted, the default is `""`.

- <visits>: An array representing the visits. See below.

  If omitted, the default is an empty array.

- <constraints>: An object representing the constraints. See below.

  If omitted, the default is an empty object.

- <persistent>: Whether the block is persistent.

  Either `"true"` or `"false"`.

  If `"true"`, the block is persistent. Persistent blocks remain in the queue after they have been executed. Non-persistent blocks are removed after they have been executed.

  If omitted, the default is `"false"`.

4.3 Constraints

The <constraints> value specifies constraints which shall be satisfied by all of the visits in a block in order for the block to be selectable. If any constraint is not satisfied by any visit, the block will not be selectable.

For a block to be selectable:

- All time-based constraints (`mindate`, `maxdate`, `minfocusdelay`, and `maxfocusdelay`) must be satisfied at the start of the first visit.
- All condition-based constraints (e.g., on transparency) must be satisfied at the start of the first visit. (In the present version of the selector, there are no constraints on unpredictable properties, but this may change in future versions.)
- All other constraints must be satisfied at the start and end of *each* visit.

Syntactically, the <constraints> value is a JSON object with the following structure:

```
{
  "mindate": <date>,
  "maxdate": <date>,
  "minsunha": <ha>,
  "maxsunha": <ha>,
  "minsunzenithdistance": <distance>,
  "maxsunzenithdistance": <distance>,
  "minmoondistance": <distance>,
  "maxmoondistance": <distance>,
  "minha": <ha>,
  "maxha": <ha>,
  "mindelta": <delta>,
  "maxdelta": <delta>,
  "minairmass": <airmass>,
  "maxairmass": <airmass>,
  "minzenithdistance": <distance>,
  "maxzenithdistance": <distance>,
  "minskybrightness": <skybrightness>,
  "maxskybrightness": <skybrightness>,
  "minfocusdelay": <delay>,
  "maxfocusdelay": <delay>,
}
```

As for any JSON object, the order of the members is irrelevant.

The members and values are interpreted as follows:

- The <mindate> and <maxdate> values specify the earliest and latest date and time on which to the visit shall be executed.

  Valid values are dates.

- The <minsunha> and <maxsunha> values specify the minimum and maximum values of the HA of the Sun.

  Valid values are angles.

  These are of most interest to calibration blocks, where these can be used to determine whether it is morning or evening.

- The <minsunzenithdistance> and <maxsunzenithdistance> values specify the minimum and maximum values of the zenith distance of the Sun.

  Valid values are angles.

  These are of most interest to calibration blocks.

- The <minmoondistance> and <maxmoondistance> members specify the minimum and maximum values of the distance of the Moon from the visit targets.

  Valid values are angles.

- The <minha> and <maxha> values specify the minimum and maximum values of the HA of the visit targets.

  Valid values are angles.

  These can be used to program several blocks on the same target in the a given night by giving each a HA range disjoint from the others.

- The <mindelta> and <maxdelta> values specify the minimum and maximum values of the declination of the visit targets.

  Valid values are angles.

- The <minairmass> and <maxairmass> members specify the minimum and maximum values of the zenith distance of the visit targets.

  Valid values are numbers.

- The <minzenithdistance> and <maxzenithdistance> values specify the minimum and maximum values of the zenith distance of the visit targets.

  Valid values are angles.

- The <minskybrightness> and <maxskybrightness> values specify the minimum (faintest) and maximum (brightest) sky brightness in which the visit shall be executed.

  Valid values are: `"daylight"`, `"civiltwilight"`, `"nauticaltwilight"`, `"astronomicaltwilight"`, `"bright"`, `"grey"`, and `"dark"`.

  Daylight and the twilights are defined conventionally by the altitude of the Sun. Bright time is any time between twilights when the moon is above the horizon and is 50% illuminated or more. Grey time is s any time between twilights when the moon is above the horizon and is 50% illuminated or less. Dark time is any time between twilights when the moon is below the horizon.

- The <minfocusdelay> and <maxfocusdelay> values specify the minimum and maximum time since the telescope was last focused. So, for example specifying a <minfocusdelay> of `"1h"` will constrain the block to be run no sooner than 1 hour after focusing and specifying a maxfocusdelay of `"1h"` will constrain the block to be run no later than 1 hour after focusing.

  Valid values are durations.

  These are of most interest to focus blocks.

In addition to any explicit constraints, the scheduler requires that all visit targets are within the telescope pointing limits at the expected start and of the visit.

If no explicit constraints are given, the implicit telescope pointing limits are the only constraint and, for example, a block may be executed at very high airmass, close to the mount, or during twilight. Therefore, it is advisable to include at least minimal constraints, for example:

```
{
  "maxskybrightness": "astronomicaltwilight",
  "maxairmass": "2.0",
  "minmoondistance": "15d"
}
```

4.4 Visits

Syntactically, the <visits> value is a JSON array containing zero or more <visit> objects:

```
[
  <visit>,
  <visit>,
  <visit>,
  ...
  <visit>
]
```

Each <visit> value is a JSON object with the following structure:

```
{
  "identifier": <visit-identifier>,
  "name": <visit-name>,
  "targetcoordinates": <target-coordinates>,
  "estimatedduration": <estimated-duration>,
  "command": <command>
}
```

As for any JSON object, the order of the members is irrelevant.

The members and values are interpreted as follows:

- <visit-identifier>: The visit identifier.

  This is a non-negative integer represented as a string.

  By convention, science visits have identifiers from 0 to 999 and focusing, pointing correction, and other non-science visits from 1000 onward. (The pipeline uses this convention to avoid reducing non-science visits.)

- <visit-name>: The visit name.

  By convention, we used names that describe the activity generically, such as `"focussing"`, `"pointingcorrection"`, `"science"`.

  If omitted, the default is `""`.

- <target-coordinates>: The target coordinates. See below.

- <estimated-duration>: The estimated duration of the visit.

  Valid values are durations (e.g., `"10m"` for 10 minutes).

  This is used to by the selector to check the constraints at the estimated end of the visit.

- <command>: The command to execute the visit. See below.

4.5 Target Coordinates

The <target-coordinates> value is a JSON object with one of the following structures.
For equatorial coordinates, the structure is:

```
{
  "type": "equatorial",
  "alpha": <alpha>,
  "delta": <delta>,
  "equinox": <equinox>
}
```

The <alpha> and <delta> values are angles. The <equinox> value is a number.
For fixed topocentric coordinates, the structure is:

```
{
  "type": "fixed",
  "ha": <ha>,
  "delta": <delta>
}
```

The <ha> and <delta> values are angles.
For the zenith, the structure is:

```
{
  "type": "zenith"
}
```

For the idle position, the structure is:

```
{
  "type": "idle"
}
```

For a solar system body, the structure is:

```
{
  "type": "solarsystembody",
  "number": <number>
}
```

The <number> is a number and refers to the number part of the minor-planet designation. For example, for (388188) 2006 DP_{14}, it would be `388188`.

4.6 Commands

The <command> value is a string representing the Tcl command that will be run to execute the visit. Here we describe the three main commands of interest to science blocks and omit the other commands that are used in calibration blocks.

Many parameters of commands have default values.

4.6.1 Focus

The `focusvisit` command focuses

The parameters are:

- `filter`: the filter in which to focus. The default is `i`.
- `exposuretime`: the exposure time in seconds. The default is `5`.

Examples:

```
"focusvisit"
"focusvisit z"
"focusvisit r 10"
```

4.6.2 Pointing Correction

The `pointingcorrectionvisit` command attempts to correct the pointing.

The parameters are:

- `filter`: the filter in which to expose. The default is `i`.
- `exposuretime`: the exposure time in seconds. The default is `15`.

Examples:

```
"pointingcorrectionvisit"
"pointingcorrectionvisit z"
"pointingcorrectionvisit z 5"
```

4.6.3 Grid

The `gridvisit` command takes exposures in possibly multiple filters over grid of dithers.

The parameters are:

- `gridrepeats`: The number of times the whole grid is repeated.
- `gridpoints`: The number of points in the grid.

  Valid values are 1 to 9.

  The grid points are distributed in a square that is 1 arcmin to a side. The grid points, in order, are the center, the four corners, and the four midpoints of the sides. So, for example, if a value of 5 is given, the grid will consist of the center and the four corners.
- `exposurerepeats`: The number of exposures taken in each grid repetition for each filter/dither combination.
- `exposuretime`: The exposure time of each exposure.
- `filters`: The filters.

  A list of filters in which to observe.

  Remember that lists in Tcl are surrounded by curly brackets. For example, `{g r i z}`.

  Filters can be repeated. So, for example, to do an ABBA sequence in *g* and *r*, you might use `{g r r g}`.
- `offsetfastest`: Whether to offset fastest (`true`) or change filters fastest (`false`).

  The default is `true`.

  If `offsetfastest` is `true`, the code will observe a whole grid in the first filter, then the observe a whole grid in the second filter, and so on.

  If `offsetfastest` is `false`, the code will observe in each filter at the first grid point, then observe in each filter at the second grid point, and so on.
- `readmode`: The read-mode.

  The default is `fastguidingmode`.

The total number of exposure is the value of `gridrepeats` multiplied by the value of `gridpoints` multiplied by the value of `exposurerepeats` multiplied by the number of `filters`. The total exposure time is this multiplied by the value of `exposuretime`.

Examples:

```
"gridvisit 4 9 1 30 {r}"
"gridvisit 1 5 1 60 {g r i z}"
"gridvisit 1 5 1 60 {640/10 656/3} false"
```

4.7 Managing the Block Queue

The telescope maintains an alert queue and a block queue. We will not discuss the alert queue here, except to note that observations are selected at a higher priority from the alert queue than from the block queue.

The block queue is maintained in a repository in GitHub:

This repository contains block files, shell scripts to generate block files, and a BLOCKS file to specify which block files are loaded, when, and with which priority.

At 00:00 UTC each day (16:00 PDT or 17:00 PST), the telescope control system fetches a copy of the repository from GitHub. At 00:01 UTC each day, it loads blocks from the copy of the repository into the queue. These actions are separate, so that even if the telescope is not able to fetch the latest version of the repository, it will still load the queue using a previous version of the repository.

The blocks that are loaded are specified in the BLOCKS file in the repository.

In the BLOCKS file, empty lines and lines beginning with # are ignored.

The remaining lines shall have four obligatory fields possibly followed by additional fields that give a time specification. Fields are separated by tabs or spaces.

The fields are:

1. Action. Either the the word `load` or the word `unload`. This specifies whether the block will be loaded or unloaded.
2. Priority. A letter, with a being highest priority and z being lowest priority.
3. Duplicates. The number of copies of the block file that are loaded or unloaded. This is useful when breaking long observations into shorter blocks; simply set this field to the number times you want to run the shorter block.
4. Block file. This can be a file name or a shell glob pattern to match a set of file names. Omit the trailing `.json`.

After the first four columns, additional optional fields can give a time specification. If no time specification is given, the block files are loaded or unloaded every day. Valid time specifications are:

- Load or unload the blocks on a fixed UTC date.

  In this case, the 5th field is the word `date` and the 6th field specifies the date as YYYYMMDD.

  This is useful when you only want to add blocks to the queue once.

- Load or unload the blocks every N days.

  In the case, the 5th field is the word `day`, the 6th field is a number M, and the 7th field is another number N. The blocks are loaded into the queue when the day of year D (1 to 366) satisfies $M = D \bmod N$.

  Note that by choosing different values of M you can cycle through a set of blocks.

Example BLOCKS file:

```
load   a 1 0004-initial-focus-*
load   b 1 0004-focus-*
load   f 1 2001-smith-*
load   g 1 2000-jones-0      day 0 2
load   g 1 2000-jones-1      day 1 2
load   h 3 2003-harris-0     date 20220218
unload h 3 2003-harris-0     date 20220318
load   i 1 2002-bloggs-0
load   x 1 0001-twilight-flats-evening-0
```

Chapter 5

Archive

5.1 Introduction

The control system generates many data files during operations.

Most of the data files are organized by first date and then component. For example, all files for the UTC date YYYYMMDD generated by the sensors component are located in

```
/usr/local/var/tcs/YYYYMMDD/sensors/
```

Some components have subdirectories at the top level for data files that are applicable to more than one night. For example, the selector maintains the alert files in

```
/usr/local/var/tcs/selector/alerts/
```

These files are copied from the control system computers at the telescope to the archive. This is a central NAS on the private transients subnetwork at the OAN. Public access is via SSH, RSYNC, HTTP access to `transients.astrossp.unam.mx`. The NAS is shared by the control system computers and the data pipelines for RATIR, COATLI, DDOTI.

The archive volume on the NAS is mounted on other computers on the transients network at:

The data files on the control system computers are copied by rsync to the subdirectory `raw` in the archive volume. The log files are copied every minute, the FITS data and header files every five minutes, and the other files every hour.

5.2 Logs

The log files are created under

```
/usr/local/var/tcs/YYYYMMDD/log/
```

They are mainly of interest to the engineering and operations team.

5.3 Image Files

The image files are created under

```
/usr/local/var/tcs/YYYYMMDD/executor/images/
```

The files are created below this directory in subdirectories whose names are the program, block, and visit identifiers. For example, the files for program 2022A-2001, block 10, visit 0 are created in

```
/usr/local/var/tcs/YYYYMMDD/executor/images/2022A-2001/10/0/
```

The base name of each image created by the executor is the UTC time in ISO 8601 basic combined format followed by the channel name (e.g., `C0`, `C1`, `C2`, . . .), followed by a letter indicating the type of exposure (`o` for object, `f` for flat, `b` for bias, and `d` for dark). For example,

```
20220405T072311C0o
20220405T072341C0b
20220405T072351C0f
20220405T072411C0d
```

the full FITS image compressed losslessly with fpack (with suffix `.fz`) and a text version of the header (with suffix `.fits.txt`). In the text version, each record is separated with a newline character.

5.4 FITS Header Records

The FITS header records are largely self-documented by comments. However, these are the most relevant header records for searching for particular data:

- `DATE-OBS`: The UTC date of the start of the exposure.
- `CCD_NAME`: The channel (e.g., `C0`, `C1`, `C2`, . . .).
- `EXPTIME`: The exposure time (seconds).

- EXPTYPE: The exposure type (e.g., object, flat, bias, dark).
- FILTER: The selected filter name.
- BINNING: The detected binning (pixels).
- READMODE: The detector read mode.
- PRPID: The proposal identifier (integer).
- BLKID: The block identifier (integer).
- VSTID: The visit identifier (integer).
- STRSTRA: The J2000 RA of the target at the start of the exposure (degrees).
- STRSTDE: The J2000 declination of the target at the start of the exposure (degrees).
- STROBHA: The observed HA of the target at the start of the exposure (degrees).
- STROBDE: The observed declination of the target at the start of the exposure (degrees).
- STROBAZ: The observed azimuth of the target at the start of the exposure (degrees).
- STROBZ: The observed zenith distance of the target at the start of the exposure (degrees).
- STROBAM: The observed airmass of the target at the start of the exposure.
- SMNZD: The observed zenith distance of the Moon at the start of the exposure (degrees).
- SMNIL: The illuminated fraction of the Moon at the start of the exposure.
- SMNTD: The distance between the target and the Moon at the start of the exposure (degrees).
- SSNZD: The observed zenith distance of the Sun at the start of the exposure (degrees).

For all of the records that begin with S and refer to the start of the exposure, there is another record that begins with E and refers to the end of the exposure. So, for example, STROBAM gives the airmass at the start of the exposure and ETROBAM gives the airmass at the end of the exposure.