PlaneWave Interface 4 API

Table of Contents

Introduction 5
Control Interfaces 6
  PWI4 HTTP server 6
  ASCOM drivers 6
  Choosing PWI4-HTTP or ASCOM 7
HTTP Server request format 7
Status Response Format 8
HTTP Response Status Codes 9
  200: OK 9
  400: Bad Request 9
  404: Not Found 9
  500: Internal Server Error 10
Using mount motion commands 10
Status values 11
  pwi4.version 37
  pwi4.version_field[0] 37
  pwi4.version_field[1] 38
  pwi4.version_field[2] 38
  pwi4.version_field[3] 38
  response.timestamp_utc 38
  site.latitude_degs 38
  site.longitude_degs 38
  site.height_meters 38
  site.lmst_hours 39
  mount.is_connected 39
  mount.geometry 39
  mount.timestamp_utc 39
  mount.julian_date 39
  mount.slew_time_constant 39

mount.ra_apparent_hours 40
mount.dec_apparent_degs 40
mount.ra_j2000_hours 40
mount.dec_j2000_degs 40
mount.target_ra_apparent_hours 40
mount.target_dec_apparent_degs 41
mount.azimuth_degs 41
mount.altitude_degs 41
mount.is_slewing 41
mount.is_tracking 41
mount.field_angle_here_degs 41
mount.field_angle_at_target_degs 42
mount.field_angle_rate_at_target_degs_per_sec 42
mount.path_angle_at_target_degs 44
mount.path_angle_rate_at_target_degs_per_sec 46
mount.distance_to_sun_degs 46
mount.axis0_wrap_range_min_degs 46
mount.offsets.[axis]_* 47
mount.offsets.[axis]_arcsec.total 47
mount.offsets.[axis]_arcsec.rate 47
mount.offsets.[axis]_arcsec.gradual_offset_progress 48
mount.spiral_offset.x 48
mount.spiral_offset.y 48
mount.spiral_offset.x_step_arcsec 48
mount.spiral_offset.y_step_arcsec 49
mount.axisN.* 49
mount.axisN.is_enabled 49
mount.axisN.rms_error_arcsec 49
mount.axisN.dist_to_target_arcsec 49
mount.axisN.servo_error_arcsec 50
mount.axisN.min_mech_position_degs 50
mount.axisN.max_mech_position_degs 50
mount.axisN.target_mech_position_degs 50
mount.axisN.position_degs 50

mount.axisN.position_timestamp 50
mount.axisN.max_velocity_degs_per_sec 51
mount.axisN.setpoint_velocity_degs_per_sec 51
mount.axisN.measured_velocity_degs_per_sec 51
mount.axisN.acceleration_degs_per_sec 51
mount.axisN.measured_current_amps 52
mount.model.filename 52
mount.model.num_points.total 52
mount.model.num_points.enabled 52
mount.model.rms_error_arcsec 53
focuser.is_connected 53
focuser.is_enabled 53
focuser.position 53
focuser.is_moving 53
rotator.is_connected 53
rotator.is_enabled 53
rotator.mech_position_degs 54
rotator.field_angle_degs 54
rotator.is_moving 54
rotator.is_slewing 54
m3.port 55
autofocus.is_running 55
autofocus.success 55
autofocus.best_position 55
autofocus.tolerance 55
Commands 13
/status 13
/mount/connect 14
/mount/disconnect 14
/mount/enable 14
/mount/disable 15
/mount/stop 15
/mount/goto_ra_dec_apparent 16
/mount/goto_ra_dec_j2000 16

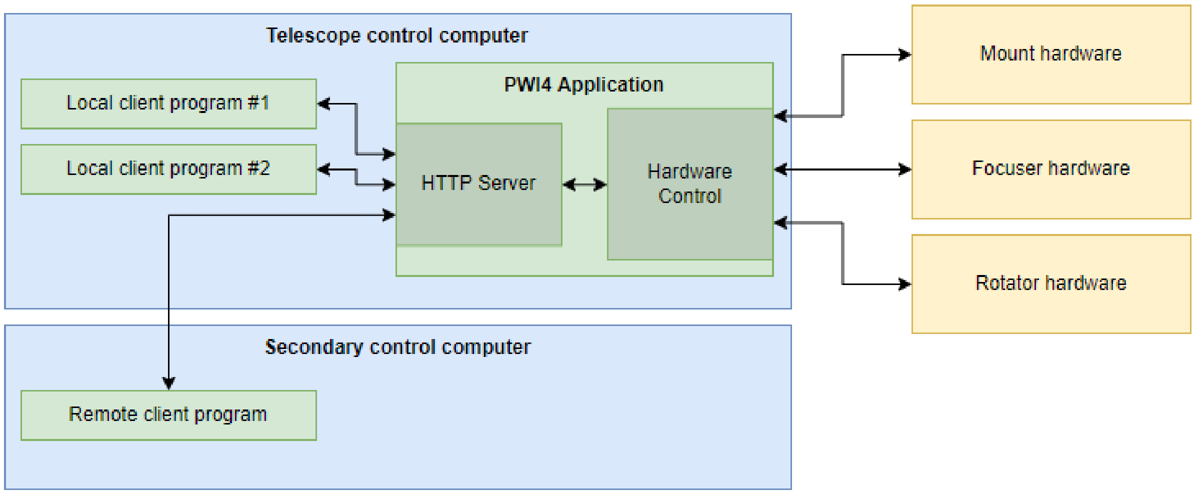
/mount/goto_alt_az 16
/mount/goto_coord_pair 17
/mount/park 18
/mount/set_park_here 18
/mount/offset 18
/mount/spiral_offset/new 20
/mount/spiral_offset/next 21
/mount/spiral_offset/previous 22
/mount/tracking_on 22
/mount/tracking_off 22
/mount/follow_tle 22
/mount/find_home 23
/mount/set_slew_time_constant 23
/mount/set_axis0_wrap_range_min 24
/mount/model/add_point 24
/mount/model/clear_points 25
/mount/model/delete_point 25
/mount/model/enable_point 25
/mount/model/disable_point 26
/mount/model/save_as_default 26
/mount/model/save 27
/mount/model/load 27
/mount/radecpath/new 28
/mount/radecpath/show 28
/mount/radecpath/add_point 28
/mount/radecpath/apply 29
/mount/custom_path/new 29
/mount/custom_path/add_point_list 29
/mount/custom_path/apply 30
/focuser/enable 31
/focuser/disable 31
/focuser/goto 31
/focuser/stop 32
/rotator/enable 32

/rotator/disable 32
/rotator/goto_mech 33
/rotator/goto_field 33
/rotator/offset 33
/rotator/stop 33
/fans/on 34
/fans/off 34
/m3/goto 34
/m3/stop 34
/virtualcamera/take_image 35
/temperatures/pw1000 35
/autofocus/start 36
/autofocus/stop 36
/autofocus/log 36
/internal/crash 37
Python reference implementation 11
Appendix A: Sample status response 56

Introduction

PlaneWave Interface 4 (PWI4) is a software package that can control a variety of PlaneWave equipment, including L-series direct drive mounts, Series 5 focusers and rotators, the PW1000 1-meter telescope system, and more. Typically, these hardware devices expose a basic set of low-level capabilities, such as reading the current position of a motor and commanding a desired velocity. PWI4 extends these capabilities by performing astronomical coordinate transformations, measuring and applying pointing model corrections, calculating field rotation effects, following trajectories for a variety of targets including satellites, and much more.

User applications will generally communicate with PlaneWave equipment using PWI4 as an intermediary. The diagram below shows how multiple programs on multiple computers can use the PWI4 HTTP server to access the connected hardware.



Control Interfaces

It is currently possible for applications to communicate with PWI4 in two ways:

1. PWI4-hosted HTTP server
2. ASCOM drivers

PWI4 HTTP server

When PWI4 is running, it hosts an HTTP (web) server that can be used to issue commands to PWI4 and read the status of the system. Essentially every programming language in common use is able to make HTTP requests, either using a built-in construct (e.g. urllib in Python, HttpURLConnection in Java, WebRequest in .NET) or a third-party library/tool (e.g. libcurl for C, curl for shell scripts). Users can experiment with commands simply by entering URLs in a web browser, and could potentially create a custom user interface to their telescope using HTML and Javascript.

This document will primarily describe the HTTP interface to PWI4.

ASCOM drivers

ASCOM provides a standard driver model for allowing applications to communicate with astronomical devices, such as mounts, focusers, rotators, and cameras. A wide variety of applications can use ASCOM-compatible devices, including TheSkyX, MaxIm DL, FocusMax, ACP, CCDAutoPilot, Sequence Generator Pro, NINA, and more. The PWI4 ASCOM drivers allow all these applications, along with any ASCOM-compatible applications developed in the future, to use PlaneWave equipment out of the box.

ASCOM started as a Windows-only technology, using the Component Object Model (COM) system to allow clients and drivers to interact. Recently the ASCOM Alpaca standard has enhanced ASCOM with an HTTP interface that can be used across machines from any operating system (Windows, Mac, or Linux). Existing COM-based ASCOM drivers (including the PWI4 drivers) can be wrapped with an Alpaca interface using the *ASCOM Remote* tool. Depending on customer interest, PWI4 may also expose a native Alpaca HTTP interface in the future. This would be useful if, for example, an ASCOM client wanted to talk to a copy of PWI4 running under Linux.

In PWI4, the ASCOM drivers are in fact simple wrappers around the HTTP interface discussed below. Because ASCOM only supports capabilities that are commonly found across all astronomical equipment, the PWI4 ASCOM drivers expose only a subset of the full capabilities provided by PWI4. For custom-developed applications, it may make more sense to use the HTTP interface directly.

For more details on working with ASCOM drivers, refer to https://ascom-standards.org/. The ASCOM Remote tool for making PWI4 compatible with ASCOM Alpaca is available from that site as well.

This document will primarily describe the direct HTTP interface to PWI4 rather than the ASCOM wrapper. However, if you are interested in using ASCOM to control PWI4, please contact PlaneWave Instruments for sample code that illustrates how to get started with ASCOM from a variety of programming languages.

Choosing PWI4-HTTP or ASCOM

Generally, you would choose to target ASCOM if your software:

- is primarily performing astronomical functions (Slewing to RA/Dec coordinates, observing sidereal targets, etc.)
- will be working with a variety of ASCOM-compatible mounts, focusers, or rotators, and not just PlaneWave hardware
- does not need to take advantage of PlaneWave-specific functionality that may not be exposed by ASCOM

You should target the PWI4 HTTP server directly if your software:

- will be observing non-sideral targets, such as satellites
- will need to interface with custom features of PlaneWave hardware, such as the M3 Nasmyth port selector
- will need to collect telemetry that is not reported by ASCOM drivers, such as position tracking errors, Alt-Az field rotation information, accurate position timestamps, raw motor position information, etc.

If you plan to control PWI4 across a network, there is also a slight advantage to communicating with the PWI4 HTTP server directly rather than using the ASCOM Alpaca wrappers. This consideration may be removed in the future if native ASCOM Alpaca support is added to PWI4.

HTTP Server request format

By default, the HTTP server is hosted on port **8220**.

A request to PWI4 will consist of a URL with the following format:

```
http://host:port/subsystem/command?param1=value1&param2=value2&...
```

host: The hostname or IP address of the machine running PWI4. This will often be **localhost** for the common case where the client program is running on the same machine as PWI4.

port: The TCP port number that the PWI4 HTTP server is listening on. By default, this is **8220**.

/subsystem/command: The command that is being sent to PWI4. In many cases both a subsystem and a command are specified (for example, **/mount/stop** and **/focuser/goto**), but in some cases no subsystem is specified (for example, **/status**).

param1=value1: For commands that take parameters, one or more parameters can be specified as name=value pairs. For example, the **/mount/goto_ra_dec_j2000** command takes parameters named **ra_hours** and **dec_degs**, and each parameter takes a numeric value. Generally, the units are included in the name of a parameter.

As a basic example, the following URL retrieves the status from a copy of PWI4 running on the local machine:

http://localhost:8220/status

Here is an example of a request with a two-part command (specifying both the subsystem and the action) that takes parameters:

http://localhost:8220/mount/goto_ra_dec_j2000?ra_hours=10.5&dec_degs=78.321

Most commands will reply with a Status Response, discussed below in the section **Status Response Format**.

Status Response Format

Most commands will return a Status Response, which is a snapshot of the state of the system at the time the response was generated. An example of the first several lines of a typical status response are shown below to provide a sense of the structure. A more complete example of a response can be found in **Appendix A: Sample status response**.

```
pwi4.version=4.0.9 beta 15
pwi4.version_field[0]=4
pwi4.version_field[1]=0
pwi4.version_field[2]=9
pwi4.version_field[3]=15
response.timestamp_utc=2021-03-11 17:59:43.925011
site.latitude_degs=33.4999722222222
site.longitude_degs=-118
site.height_meters=50
site.lmst_hours=21.4366499466139
mount.is_connected=true
[remaining lines excluded from this example]
```

The response consists of a series of lines separated by the newline character (ASCII character 0x0A, commonly represented as '\n' in most programming languages). Each line is of the form:

[keyword]=[value]

Keywords in the response are generally divided into [category].[property] or [category].[subcategory].[property].

Values are generally one of the following types:

Floating point (float): Can be either standard decimal notation, such as "-1.234", or scientific notation to express very large or very small values, such as "6.294E-08". The period character (.) is always used as a decimal separator, even when PWI4 is running on a computer where the locale settings might designate that numbers should use a comma as the decimal separator.

Boolean: Can be either "true" or "false"

Integer: A simple series of digits with an optional sign, such as "123" or "-1".

String: An arbitrary sequence of printable characters, such as "PlaneWave CDK700" or "-12:34:56.7".

Timestamp: A string of the form:

yyyy-MM-dd HH:mm:ss.ffffff

yyyy: The 4-digit year (e.g. 2021)
MM: The 2-digit month (00 – 12)
dd: The 2-digit day (00 – 31)
HH: The 2-digit hour (00 – 23)
mm: The 2-digit minute (00 – 59)
ss: The 2-digit second (00 – 59)
ffffff: The fractional part of a second (000000 – 999999)

Numeric values typically have their units expressed in the keyword. For example, "mount.axis0.dist_to_target_arcsec" represents a value in arcseconds, while "mount.azimuth_degs" represents a value in degrees.

HTTP Response Status Codes

The following HTTP status codes are used by the PWI4 API:

200: OK

Status code 200 indicates that the request was successfully processed by the HTTP server and a status response could be generated. Note however that this does NOT mean that the requested action was successfully completed. To determine successful completion of an action, fields within the status response must be inspected. For example, issuing a /mount/connect request only enqueues a command to connect to the mount. The actual state of the connection (which could take several seconds to complete on some hardware) needs to be checked via the "mount.is_connected" status field that is retrieved in future calls to /status.

400: Bad Request

Status code 400 indicates that the requested endpoint was found, but one or more parameters had errors. For example, a required parameter may be missing, or a numeric parameter may be improperly formatted. The response body will contain text/plain content describing the error.

Examples:

This will produce an Error 400 due to the value of "target" being invalid:
http://localhost:8220/focuser/goto?target=asdf

This will produce an Error 400 due to the required "type" parameter not being specified:
http://localhost:8220/mount/goto_coord_pair?c0=10&c1=20

404: Not Found

Status code 404 indicates that the requested endpoint was not found. The response body will contain text/plain content with the text "404 NotFound".

Examples:

This will produce an Error 404 due to /unknown/endpoint not being a valid command:

http://localhost:8220/unknown/endpoint

500: Internal Server Error

Status code 500 indicates that there was an internal problem processing the request or generating a response. This may indicate that something abnormal has happened within PWI4, and may be due to a bug or an error case that is not being properly handled within PWI4. Commands that produce this result should be reported to PlaneWave Instruments for inspection. The response will include a "text/plain" body containing some text (typically an Exception traceback) that may be useful in identifying the problem.

Examples:

The /internal/crash endpoint can be used to intentionally produce an Error 500 response. It can be used to test how your HTTP library reacts to this error code. The "crash" that occurs is isolated to just this request; the rest of PWI4 and the HTTP server should be unaffected.

http://localhost:8220/internal/crash

Using mount motion commands

Numerous commands are available for commanding the mount to follow a particular target. A few of the more common targeting commands include:

- /mount/goto_ra_dec_j2000
- /mount/follow_tle
- /mount/custom_path/apply

These commands set a new "baseline target" within PWI4, and the mount will try to acquire and track the specified target. In response to one of these commands, the following should occur:

`mount.is_slewing` will read `true` while the mount is acquiring the target, and will transition to `false` once the mount is "on target" (as defined below).

`mount.is_tracking` will read `true` while the mount has a moving target that it is trying to acquire or follow, and will transition to `false` once the mount is "stopped" (no longer as an active target to follow).

`mount.axisN.dist_to_target_arcsec` and `mount.axisN.rms_error_arcsec` will typically become large as the two mount axes are moving a potentially large distance to acquire the target. As the mount slews closer to the target, these numbers should continuously decrease in size. Once the mount is on target, these numbers should remain consistently small (typically below 0.2 arcseconds for sidereal targets) and will vary somewhat depending on servo performance and communication jitter.

Currently the `mount.is_slewing` value will transition to `true` once the first commands to follow a new target have been sent to the mount controller, and will transition to `false` once the RMS value of the distance to target (measured over the past 1 second of collected measurements) drops below 2 arcseconds.

Note that you can use your own criteria for deciding that a mount has “settled adequately” by monitoring the values of `mount.axisN.dist_to_target_arcsec`. For example:

- A sidereal astrophotography target may benefit from a tighter criteria for “settled”
- A fast-moving satellite may require a looser criteria
- A time-critical observation of a transient event (for example, a Gamma Ray Burst) may consider anything within the camera’s field of view to be “close enough” to start the exposure.

As of this writing, PWI4’s standard policy for slewing to targets beyond the telescope limits is to get “as close as possible” in each axis and wait for the target to move within the telescope’s observable limits. This is mostly useful for observing low-earth-orbit satellites that move rapidly across the sky. For example, consider an Alt-Az telescope with a lower altitude limit of 10 degrees. If a satellite is just starting to rise above the horizon and PWI4 is commanded to follow that satellite, it will slew to the current Azimuth of the satellite, but wait at the 10 degree Altitude limit. As soon as the satellite rises above 10 degrees, the altitude axis will begin moving upward to track the satellite.

The option to command other policies for out-of-limit scenarios (for example, to return an error) will be included in future versions of PWI4.

Python reference implementation

A reference implementation for communicating with PWI4 via the HTTP interface can be found here:

http://planewave.com/files/software/PWI4/python/pwi4_client.py

The code should be compatible with Python 2.7 and Python 3.x.

An example of how to use this Python module can be found here:

http://planewave.com/files/software/PWI4/python/pwi4_client_demo.py

The commands in pwi4_client.py are generally very thin wrappers around the URLs described in this document. For example, if we look at the implementation of the mount_connect() function:

```
def mount_connect(self):
    return self.request_with_status("/mount/connect")
```

we can see that it is simply a wrapper around the HTTP request:

http://localhost:8220/mount/connect

Similarly, if we look at a function that takes some arguments:

```
def mount_goto_alt_az(self, alt_degs, az_degs):
    return self.request_with_status(
                "/mount/goto_alt_az",
                alt_degs=alt_degs,
                az_degs=az_degs)
```

this gets converted into a URL such as the following:

http://localhost:8220/mount/goto_alt_az?alt_degs=45.123&az_degs=315.987

All status values are parsed in the PWI4Status class, which returns a structure that mirrors the structure of the standard status response. For example, by looking at the following section of code:

```
self.mount.axis0 = Section()
    self.mount.axis0.is_enabled =
        self.get_bool("mount.axis0.is_enabled")
    self.mount.axis0.rms_error_arcsec =
        self.get_float("mount.axis0.rms_error_arcsec")
    self.mount.axis0.dist_to_target_arcsec =
        self.get_float("mount.axis0.dist_to_target_arcsec")
    self.mount.axis0.servo_error_arcsec =
        self.get_float("mount.axis0.servo_error_arcsec")
    self.mount.axis0.position_degs =
        self.get_float("mount.axis0.position_degs")
    self.mount.axis0.position_timestamp_str =
        self.get_string("mount.axis0.position_timestamp")
```

we can see that the following values would be accessible in a resulting “status” variable that gets returned by the status() function:

Name	**Datatype**
status.mount.axis0.is_enabled	boolean
status.mount.axis0.rms_error_arcsec	floating-point
status.mount.axis0.dist_to_target_arcsec	floating-point
status.mount.axis0.servo_error_arcsec	floating-point
status.mount.axis0.position_degs	floating-point
status.mount.axis0.position_timestamp	string

Sample Python Scripts

A number of samples are available that demonstrate how to use pwi4_client.py and the features of the PWI4 HTTP API.

http://planewave.com/files/software/PWI4/python/pwi4_client_demo.py
A very simple introductory script demonstrating how to use pwi4_client.py

http://planewave.com/files/software/PWI4/python/pwi4_startup.py
Demonstrates a script that might be used to start up the observatory each evening, performing tasks like connecting to the mount, enabling the motors, and finding the home position.

http://planewave.com/files/software/PWI4/python/follow_tle_offset_demo.py
Demonstrates how to follow a satellite using TLE data, and how to perform offsets around that satellite using the **/mount/offset** command. A video is also available demonstrating this script:
https://www.youtube.com/watch?v=6f52QBC5k9k

http://planewave.com/files/software/PWI4/python/pwi4_build_model.py
depends on: http://planewave.com/files/software/PWI4/python/platesolve.py
Demonstrates how to build a pointing model using the scripting interface. In the script, the **/virtualcamera/take_image** command is used to produce a simulated image that can be platesolved, but this can be replaced by the code that is needed to take an image with your actual camera. Note that this script requires the "ps3cli" platesolve module. Please contact PlaneWave Instruments for more details.

http://planewave.com/files/software/PWI4/python/log_pwi4_values.py
A simple example showing how to query selected status values and log them to a file.

http://planewave.com/files/software/PWI4/python/custom_path_test.py
Demonstrates building and executing a custom path made from timestamped positions.

Commands

The commands available in the PWI4 API are described below.

/status

Returns a Status Response containing a snapshot of the system state, as described in the section **Status Response Format**. Although most commands also return a status response, this command is used to explicitly request the status without having any other side-effects on the system.

Parameters

None

Example

Retrieve the status from a default instance of PWI4 running on the same machine:
http://localhost:8220/status

/mount/connect

Establish a connection from PWI4 to the configured mount controller. Some mounts may take several seconds to connect; the response body will not be returned from the server until the connection process has completed.

The value of `mount.is_connected` should be checked after this call. If it is `true`, the connection was successfully established. If it is `false`, there was an error establishing the connection.

If the connection to the mount is later lost, or if a disconnect is requested outside of the current API session (for example, by clicking the Disconnect button in the GUI, or by another API session calling `/mount/disconnect`), the `mount.is_connected` field will transition to `false`. It is advisable to monitor this field during the API session.

If PWI4 already has an active connection to the mount, this call will return immediately indicating success.

Note that in some hardware configurations, other devices may share the same controller as the mount. For example, in the PW1000 telescope system, the integrated focusers and rotators run on the same controller as the mount. As a result, connecting to the mount will also cause these devices to be connected as well.

Parameters

None

Example

Connect to the configured mount:
http://localhost:8220/mount/connect

/mount/disconnect

Close the connection from PWI4 to the configured mount controller, if one exists.

After this call, the value of `mount.is_connected` should be `false`.

Note that in some hardware configurations, other devices may share the same controller as the mount. For example, in the PW1000 telescope system, the integrated focusers and rotators run on the same controller as the mount. As a result, disconnecting from the mount will cause these other devices to be disconnected as well.

Parameters

None

Example

Disconnect the mount:
http://localhost:8220/mount/disconnect

/mount/enable

Begin enabling servo control on one of the mount motors.

The motors on PlaneWave mounts can be either in the Enabled or Disabled state. In the Disabled state, there is typically no voltage going to the motor. In the case of a direct-drive axis, the axis can typically be rotated freely by hand. When the motor is in the Enabled state, the axis is under servo control and motion can be commanded electronically.

After this call, monitor the value of `mount.axisN.is_enabled.` Once the axis is fully enabled, the value should be `true`. Some axes may take several seconds to fully enable.

Axes can become disabled during operation, either as a reaction to an error condition in order to protect the equipment, or due to an external request to disable the motors; for example, by clicking the Disable button on the GUI.

Parameters

axis (integer; required): The axis number to enable. Either 0 for the “primary” axis (Azimuth or RA), or 1 for the “secondary” axis (Altitude or Declination).

Example

Enable axis 0:
http://localhost:8220/mount/enable?axis=0

Enable axis 1:
http://localhost:8220/mount/enable?axis=1

/mount/disable

Disable servo control on one of the mount motors.

After this call, `mount.axisN.is_enabled` should be `false`. When an axis is disabled, no voltage will flow to the motor. In the case of a direct-drive axis, the axis can typically be rotated freely by hand.

Parameters

axis (integer; required): The axis number to enable. Either 0 for the “primary” axis (Azimuth or RA), or 1 for the “secondary” axis (Altitude or Declination).

Example

Disable axis 0:
http://localhost:8220/mount/disable?axis=0

Disable axis 1:
http://localhost:8220/mount/disable?axis=1

/mount/stop

Stop following any moving targets. The mount will decelerate to a stop and then attempt to hold its final position. Under normal circumstances, the mount motors should remain enabled.

If this command is called when the mount is already stopping or stopped, it will have no effect.

Parameters
None

Example
Stop the mount:
http://localhost:8220/mount/stop

/mount/goto_ra_dec_apparent
Begin following the moving target described by the provided Right Ascension and Declination coordinates, given in the Apparent Geocentric coordinate frame.

See the section **Using mount motion commands** for additional information.

Parameters
ra_hours (number; required): The Right Ascension of the target, in decimal hours.
dec_degs (number; required): The Declination of the target, in decimal degrees.

Example
Slew to a target at Apparent coordinates RA = 21.5 hours, Dec = 85.4 degrees
http://localhost:8220/mount/goto_ra_dec_apparent?ra_hours=21.5&dec_degs=85.4

/mount/goto_ra_dec_j2000
Begin following the moving target described by the provided Right Ascension and Declination coordinates, given in the J2000 coordinate frame.

See the section **Using mount motion commands** for additional information.

Parameters
ra_hours (number; required): The Right Ascension of the target, in decimal hours.
dec_degs (number; required): The Declination of the target, in decimal degrees.

Example
Slew to a target at J2000 coordinates RA = 21.5 hours, Dec = 85.4 degrees
http://localhost:8220/mount/goto_ra_dec_j2000?ra_hours=21.5&dec_degs=85.4

/mount/goto_alt_az
Begin moving to the specified Azimuth and Altitude coordinates.

The Altitude is specified as an unrefracted (physical) value. For example, specifying an Altitude of 0 degrees should leave the OTA pointed parallel to the ground (as opposed to about 0.5 degrees above the horizon, where a stellar target that is geometrically at 0 degrees altitude would appear to be due to atmospheric refraction).

See the section **Using mount motion commands** for additional information. However, unlike the commands that follow moving targets, the mount will be "stopped" (`mount.is_tracking=false`) once this move is complete.

Parameters

alt_degs (number; required): The Altitude (unrefracted) of the target, in decimal degrees.
az_degs (number; required): The Azimuth of the target, in decimal degrees.

Example

Slew to a target at 90 degrees Azimuth, 45 degrees Altitude
http://localhost:8220/mount/goto_alt_az?alt_degs=45&az_degs=90

/mount/goto_coord_pair

Begin moving to the specified pair of coordinates with the specified coordinate type.

The following coordinate types are supported:

- **raw**: Motor positions (expressed in degrees) without any pointing model, refraction, or mount geometry adjustments, where c0 is for the "primary" axis (Azimuth or RA) and c1 is for the "secondary" (Altitude or Dec) axis.
- **altaz_observed**: Alt-az coordinates where the altitude is the physical orientation of the telescope's optical axis above the horizon. This specifies the altitude where a telescope would actually need to point to observe a target outside Earth's atmosphere. This could also be used to specify the geometric direction of a nearby terrestrial target that is not significantly affected by refraction. c0 refers to Azimuth, and c1 refers to Altitude.
- **altaz_topocentric**: Alt-az coordinates of a target outside Earth's atmosphere, expressed without considering the effect of refraction. For a value of 0 degrees altitude, the telescope should actually be pointed about 0.5 degrees above the horizon to account for atmospheric refraction changing the apparent direction of the incoming light. c0 refers to Azimuth, and c1 refers to Altitude.

See the section **Using mount motion commands** for additional information.

Parameters

c0 (number or sexagesimal; required): The first coordinate of the target, in decimal degrees or as a sexagesimal DD:MM:SS.sss string
c1 (number or sexagesimal; required): The second coordinate of the target, in decimal degrees or as a sexagesimal DD:MM:SS.sss string
type (string; required): The type of coordinates specified by c0 and c1. May be "raw", "altaz_observed", or "altaz_topocentric". See descriptions above.

Example

Slew to raw motor coordinates Primary=90 degrees, Secondary=45 degrees

http://localhost:8220/mount/goto_coord_pair?c0=90&c1=45&type=raw

/mount/park

Begin moving to the configured park position of the telescope. Once the telescope reaches the park position, it will be in the "stopped" state (`mount.is_tracking=false`).

Parameters

None

Example

Begin moving to the park position

http://localhost:8220/mount/park

/mount/set_park_here

Set the configured park position of the mount to the current position.

Parameters

None

Example

Set the park position to the current position

http://localhost:8220/mount/set_park_here

/mount/offset

Modify one or more of the offsets that are dynamically applied on top of the "baseline" coordinates that the mount is currently following.

The baseline coordinates are calculated based on the last target that was sent to the mount; for example, if the mount is following a satellite specified via a TLE, the baseline coordinates are calculated based on where the TLE predicts the satellite will appear at the current time.

On top of this, one or more position or rate offsets can be applied to the following axes before the final motion commands are sent to the mount on the next update cycle:

- **ra**: Offset along the Right Ascension direction
- **dec**: Offset along the Declination direction
- **axis0**: Offset along the mount's primary axis position (roughly equivalent to Azimuth on an Alt-Az mount, or RA in an equatorial mount)
- **axis1**: Offset along the mount's secondary axis position (roughly equivalent to Altitude on an Alt-Az mount, or Declination in an equatorial mount)
- **path**: Offset along the direction of travel for a moving target

- **transverse**: Offset perpendicular to the direction of travel for a moving target

The following actions may be performed on each of these axis offsets:

- **reset**: Clear all position and rate offsets for this axis. Set the parameter to any value to issue this command.
- **stop_rate**: Set any active offset rate to zero. Set the parameter to any value to issue this command.
- **add_arcsec**: Increase the current offset by the specified amount
- **set_rate_arcsec_per_sec**: Continually increase the offset at the specified rate
- **stop**: Stop both the offset rate and any gradually-applied commands
- **stop_gradual_offset**: Stop only the gradually-applied offset, and maintain the current rate
- **set_total_arsec**: Set the total accumulated offset at the time the command is received to the specified value. Any in-progress rates or gradual offsets will continue to be applied on top of this.
- **add_gradual_offset_arcsec**: Gradually add the specified value to the total accumulated offset. Must be paired with AXIS_gradual_offset_rate or AXIS_gradual_offset_seconds to determine the timeframe over which the gradual offset is applied:
    - **gradual_offset_rate**: Specifies the rate at which a gradual offset should be applied. For example, if an offset of 10 arcseconds is to be applied at a rate of 2 arcsec/sec, then it will take 5 seconds for the offset to be applied.
    - **gradual_offset_seconds**: Specifies the time it should take to apply the gradual offset. For example, if an offset of 10 arcseconds is to be applied over a period of 2 seconds, then the offset will be increasing at a rate of 5 arcsec/sec.

These offsets are specified by issuing one or more parameters that combine an axis and a command, separated by an underscore (_), as follows:

AXIS_COMMAND

For example, to set the RA rate to 1 arcsecond per second, the following parameter would be used:

ra_set_rate_arcsec_per_sec=1

Or to move the telescope 30 arcseconds ahead of where the moving target is predicted to be:

path_set_total_arcsec=30

Any number of these axis+command parameters may be combined into a single /mount/offset command. See the examples below.

When a new moving target is specified, all offsets reset back to 0.

The current state of these offsets can be queried in the status response under the **mount.offsets**.[axis]_* subcategory.

Parameters

See the description above for an illustration of how to form parameters for this command. All parameters are optional, and take a decimal number as a value.

Examples
Offset the current target by 1 arcsecond in RA and 2 arcseconds in Dec (for example, in response to a guide star measurement):
http://localhost:8220/mount/offset?ra_add_arcsec=1&dec_add_arcsec=2

Set the target rate to 0.2 arcsec/sec in RA, and 0.1 arcsec/sec in Dec (for example, to follow a comet):
http://localhost:8220/mount/offset?ra_set_rate_arcsec_per_sec=0.1&dec_set_rate_arcsec_per_sec=0.2

Reset all pointing offsets in the RA and Dec directions:
http://localhost:8220/mount/offset?ra_reset=0&dec_reset=0

Offset the axis0 motor by 30 arcseconds, but gradually apply the offset over the course of 5 seconds:
http://localhost:8220/mount/offset?axis0_add_gradual_offset_arcsec=30&axis0_gradual_offset_seconds=5

/mount/spiral_offset/new
Begin a new spiral offset, resetting any existing spiral offset to a grid position of X=0, Y=0.

The spiral grid consists of an integer X and Y position that can be advanced in discrete steps. Each step in X and Y will offset the pointing target by a multiple of x_step_arcsec or y_step_arcsec. The offsets are applied in the Apparent RA and Dec directions.

For example, if you have a camera with a field of view of 300 x 200 arcseconds and you are trying to locate a bright target without a pointing model, you might begin by slewing to the target and creating a new spiral offset grid with x_step_arcsec = 250 and y_step_arcsec = 150. Then, repeatedly call **/mount/spiral_offset/next**, taking an exposure each time after the mount has settled, until the target is seen by the camera. If you accidentally move past the target, you can go to previous positions using **/mount/spiral_offset/previous**.

As with **/mount/offset**, these offsets are applied relative to the baseline trajectory that the mount is following. As a result, you can perform a spiral search even around a fast-moving target such as a low-Earth-orbit satellite. This can be useful in instances where the satellite orbit is not well characterized and initial pointing may be off by several fields of view.

Parameters
x_step_arcsec (float; required): The size of each spiral grid step in the X direction, in arcseconds
y_step_arcsec (float; required): The size of each spiral grid step in the Y direction, in arcseconds

Example
Begin a new spiral grid with an X size of 250 arcseconds and a Y size of 150 arcseconds:
http://localhost:8220/mount/spiral_offset/new?x_step_arcsec=250&y_step_arcsec=150

/mount/spiral_offset/next

Advance to the next spiral grid position. The telescope will move in either X or Y by either x_step_arcsec or y_step_arcsec, as specified in **/mount/spiral_offset/new**. The following table shows the order of X and Y gridpoints in the first 16 successive calls to /mount/spiral_offset/next, as well as the effective offset that would be applied if **/mount/spiral_offset/new** had been initialized with x_step_arcsec=250 and y_step_arcsec=150:

Iteration	X	Y	X total arcsec	Y total arcsec
0	0	0	0	0
1	1	0	250	0
2	1	-1	250	-150
3	0	-1	0	-150
4	-1	-1	-250	-150
5	-1	0	-250	0
6	-1	1	-250	150
7	0	1	0	150
8	1	1	250	150
9	2	1	500	150
10	2	0	500	0
11	2	-1	500	-150
12	2	-2	500	-300
13	1	-2	250	-300
14	0	-2	0	-300
15	-1	-2	-250	-300
16	-2	-2	-500	-300

Here are the first 25 offsets visualized as a grid:

		X				
		-2	**-1**	**0**	**1**	**2**
Y	**-2**	16	15	14	13	12
	-1	17	4	3	2	11
	0	18	5	0	1	10
	1	19	6	7	8	9
	2	20	21	22	23	24

When the grid offset is changed, **mount.axisN.dist_to_target_arcsec** will spike and then gradually come back towards zero as the mount moves to the new position.

Parameters

None

Example

Advance to the next spiral position:

http://localhost:8220/mount/spiral_offset/next

/mount/spiral_offset/previous

Move back to the previous spiral position. This can be useful if a “next” command was sent before registering that the target was seen in the previous exposure.

If the spiral grid is at position X=0, Y=0, then this call has no effect.

Refer to the tables found under **/mount/spiral_offset/next**. This command will move backwards through the table from the current X,Y position.

Parameters

None

Example

Revert to the previous spiral position:
http://localhost:8220/mount/spiral_offset/previous

/mount/tracking_on

Begin following the sidereal (RA/Dec) target that is currently passing in front of the telescope. This can be thought of as “turning sidereal tracking on”.

If the mount is already following another (potentially non-sidereal) target, this command will stop following that target and begin following current sidereal position instead.

Parameters

None

Example

Begin tracking the current sidereal position:
http://localhost:8220/mount/tracking_on

/mount/tracking_off

Stop following any active targets. This is essentially equivalent to **/mount/stop**.

Parameters

None

Example

Stop following any active targets:
http://localhost:8220/mount/tracking_off

/mount/follow_tle

Begin following a using coordinates calculated from the specified two-line element (TLE) set.

TLE data is available from a variety of sources, including Celestrak (https://celestrak.org) and Space-Track (https://www.space-track.org).

PWI4 uses the SGP4 propagator for satellites with an orbital period shorter than 225 minutes. For satellites with a longer orbital period, the SDP4 propagator is used.

Although the format is called **two**-line elements, TLE data commonly includes a first “title” line containing the name of the object. This three-line format, sometimes referred to as 3LE, is what is recognized by PWI4.

NOTE: The TLE format has its origins with 80-column punched cards, and is very strict in which columns are used for which purposes. When formatting the line2 and line3 parameters to this command, it is critical that all spaces between parameters are preserved, and are not automatically consolidated into individual spaces or otherwise transformed.

See the section **Using mount motion commands** for additional information.

Parameters

line1 (string; required): The name of the satellite. Can generally be set arbitrarily
line2 (string; required): The first line of the TLE set
line3 (string; required): The second line of the TLE set

Example

Slew to geosynchronous satellite AMC-1, using TLE data downloaded from Celestrak on 2022-10-03. Note that the %20 URL code is used to encoder a “space” character to reduce ambiguity about how many spaces are included between parameters.

http://localhost:8220/mount/follow_tle?line1=AMC-1%20(GE-1)&line2=1%2024315U%2096054A%20%20%2022275.63417522%20%20.00000060%20%2000000%2B0%20%2000000%2B0%200%20%209991&line3=2%2024315%20%20%205.8687%20%2072.2153%200003221%20109.2822%20287.2529%20%201.00270763%2032751

/mount/find_home

Begin moving to the home position, if available.

Some mounts (such as the L-series mounts) use position encoders that lose their position after a power cycle. The position readouts can be re-initialized by locating a “home” position that is at a known encoder location.

Other mounts that use absolute encoders (such as the RC700 and PW1000) do not need to be homed, and this command will be ignored.

Parameters

None

Example

Begin homing the mount
http://localhost:8220/mount/find_home

/mount/set_slew_time_constant

Set the slew time constant value, which dictates how aggressively the mount will approach its target. More details can be found under the description of the **mount.slew_time_constant** status value.

Parameters
value (float; required): The new value to set as the slew time constant, in seconds

Example
Set the slew time constant to 0.3 seconds
http://localhost:8220/mount/set_slew_time_constant?value=0.3

/mount/set_axis0_wrap_range_min
Set the minimum wrap range value for Axis 0 (Azimuth / RA), which dictates how the mount will unwind a multi-turn axis to reach a target. More details can be found under the description of the **mount.axis0_wrap_range_min_degs** status value.

Parameters
degs (float; required): The new value to set as the minimum wrap range, in degrees.

Example
Set the minimum wrap range to -180 degrees:
http://localhost:8220/mount/set_axis0_wrap_range_min?degs=-180

/mount/model/add_point
Add a point to the pointing model.

The current position of the mount will be mapped to the provided J2000 RA/Dec parameters. The provided parameters are typically determined by taking an image and using a PlateSolve star-matching routine to find the center coordinates of the image, or by manually centering the telescope on a bright star and providing the coordinates of that star.

When a new point is added, the model will be re-fit to all enabled calibration points and new correction terms will immediately be applied.

Note: Any time the pointing model is modified, the telescope will come to a stop.

Parameters
ra_j2000_hours (float or sexagesimal; required): The J2000 Right Ascension that the telescope is actually pointing at, in hours
dec_j2000_degs (float or sexagesimal; required): The J2000 Declination that the telescope is actually pointing at, in degrees

Example
Add a point to the pointing model that maps the current encoder position to RA 10h20m30.456s, Dec 15d30m45.6s, using sexagesimal format:
http://localhost:8220/mount/model/add_point?ra_j2000_hours=10:20:30.456&dec_j2000_degs=15:30:45.6

Add a point to the pointing model using RA 10.5 hours, Dec -20.4 degrees, in decimal format:
http://localhost:8220/mount/model/add_point?ra_j2000_hours=10.5&dec_j2000_degs=-20.4

/mount/model/clear_points

Remove all calibration points from the pointing model. All pointing model correction terms will reset to 0, and the reported and commanded telescope positions will be based only on raw motor coordinates (plus atmospheric refraction correction if enabled).

Note: Any time the pointing model is modified, the telescope will come to a stop.

Parameters

None

Example

Clear all calibration points:
http://localhost:8220/mount/model/clear_points

/mount/model/delete_point

Remove one or more points from the list of calibration points in the model. If any of the removed points were previously enabled, the model will be re-fit with the remaining set of enabled calibration points.

Note that the points are re-indexed after this command completes. Therefore, if this command is called repeatedly with point index "0", eventually all calibration points will be removed.

If a specified index does not exist, the server will return a 400 BadResponse.

Note: Any time the pointing model is modified, the telescope will come to a stop.

Parameters

index (integer or list of integers; required): A number or a comma-separated list of numbers indicating the indexes of the points to delete. The first point is at index 0. If index is "2", then the third calibration point will be removed. If index is "0,1,2,3", then the first four calibration points will be removed.

Example

Remove the first calibration point:
http://localhost:8220/mount/model/delete_point?index=0

Remove the first 3 calibration points:
http://localhost:8220/mount/model/delete_point?index=0,1,2

/mount/model/enable_point

Mark one or more specified calibration points as being "enabled" in the model, meaning that the points will contribute to the fitting of the model.

If the specified index does not exist, the server will return a 400 BadResponse.

Note: Any time the pointing model is modified, the telescope will come to a stop.

Parameters
index (integer or list of integers; required): A number or a comma-separated list of numbers indicating the indexes of the points to enable. The first point is at index 0. If index is “2”, then the third calibration point will be enabled. If index is “0,1,2,3”, then the first four calibration points will be enabled.

Example
Enable the first calibration point:
http://localhost:8220/mount/model/enable_point?index=0

Enable the first 3 calibration points:
http://localhost:8220/mount/model/enable_point?index=0,1,2

/mount/model/disable_point
Mark one or more specified calibration points as being “disabled” in the model, meaning that the points will still be stored in the model but will not contribute to the fitting of the terms.

If a point that was previously added to the model is seen to be an outlier or to have a highly negative effect on the RMS fit of the model, it may be worth disabling the point temporarily to see what effect its removal has. If the point is determined to be unnecessary or harmful, it can then be deleted completely.

If a specified index does not exist, the server will return a 400 BadResponse.

Note: Any time the pointing model is modified, the telescope will come to a stop.

Parameters
index (integer or list of integers; required): A number or a comma-separated list of numbers indicating the indexes of the points to disable. The first point is at index 0. If index is “2”, then the third calibration point will be disabled. If index is “0,1,2,3”, then the first four calibration points will be disabled.

Example
Disable the first calibration point:
http://localhost:8220/mount/model/disable_point?index=0

Disable the first 3 calibration points:
http://localhost:8220/mount/model/disable_point?index=0,1,2

/mount/model/save_as_default
Save the current pointing model as the default model that will be loaded the next time PWI4 is started. The standard location for the default pointing model is:

```
Documents\PlaneWave Instruments\PWI4\Mount\DefaultModel.pxp (under Windows)
~/PlaneWave Instruments/PWI4/Mount/DefaultModel.pxp (under Linux)
```

Saving the model will not affect telescope pointing or tracking.

Parameters
None

Example
Save the current pointing model as the default model:
http://localhost:8220/mount/model/save_as_default

/mount/model/save
Save the current pointing model with a specified filename. The file will be saved under the following directory:

Documents\PlaneWave Instruments\PWI4\Mount\ (under Windows)
~/PlaneWave Instruments/PWI4/Mount/ (under Linux)

For security reasons, other directories may not be specified as part of the filename, and only a limited set of characters may be used as part of the filename. See the parameter description below for details. If invalid characters are used, the server will respond with a 400 Bad Request error.

The saved model may later be loaded using the **/mount/model/load** command.

Saving the model will not affect telescope pointing or tracking.

Parameters
filename (string; required): The filename to use when saving the pointing model. This parameter may only consist of letters, numbers, period (.), and underscore (_) characters. Directories may not be specified as part of the filename.

Example
Save the model as "camera1_port2.pxp":
http://localhost:8220/mount/model/save?filename=camera1_port2.pxp
Try to save the model with an invalid filename (produces as 400 Bad Request error):
http://localhost:8220/mount/model/save?filename=/invalid/characters.pxp

/mount/model/load
Load the pointing model with the specified filename. The filename and directory limitations described under **/mount/model/save** apply to this command as well.

If the file does not exist or the model cannot be loaded, an HTTP error response code will be returned.

The mount will come to a stop when a pointing model is successfully loaded.

Parameters
filename (string; required): The filename of the pointing model to load. This parameter may only consist of letters, numbers, period (.), and underscore (_) characters. Directories may not be specified as part of the filename.

Example
Load the default pointing model:
http://localhost:8220/mount/model/load?filename=DefaultModel.pxp

Load a model that was previously saved as “camera1_port2.pxp”:
http://localhost:8220/mount/model/load?filename=camera1_port2.pxp

/mount/radecpath/new

Begin a new custom RA/Dec path.

This command can be followed by a series of calls to **/mount/radecpath/add_point** to build a list of timestamped RA/Dec points that the mount should pass through. After the list is built, a call to **/mount/radecpath/apply** will begin the process of following this interpolated path.

Note: If possible, the commands under /mount/custom_path/ are preferred. For large point lists, the repeated calls to **/mount/radecpath/add_point** can be quite slow. The improved **/mount/custom_path/add_point_list** interface allows for bulk upload of multiple points in a single call and is significantly faster.

Parameters

None

Example

Begin a new custom RA/Dec path:
http://localhost:8220/mount/radecpath/new

/mount/radecpath/show

Return the list of points that currently exist in the radecpath buffer. The response is of type text/plain, and consists of one point per line with 3 comma-separated fields:

[Julian date], [J2000 RA Hours (decimal)], [J2000 Dec Degrees (decimal)]

Parameters

None

Example

Show the current list of points in the RA/Dec path:
http://localhost:8220/mount/radecpath/show

/mount/radecpath/add_point

Add a point to the list of points that will be used to build an interpolated path for the mount to follow. Times are specified using the Julian Date. For reference, the current Julian Date may be found in the status response under **mount.julian_date**.

Parameters

jd (float; required): The Julian Date at which the mount should pass through the specified point
ra_j2000_hours (float or sexagesimal; required): The J2000 Right Ascension that the mount should pass through at the specified time, in hours
dec_j2000_degs (float or sexagesimal; required): The J2000 Declination that the mount should pass through at the specified time, in degrees

Example
Add RA 12:45:00, Dec 15:30:45 at Julian Date 2451545.6, using sexagesimal format:
http://localhost:8220/mount/radecpath/add_point?ra_j2000_hours=12:45:00&dec_j2000_degs=-15:30:45&jd=2451545.6
Add RA 13.2, Dec -10.8 at Julian Date 2451545.7, using decimal format:
http://localhost:8220/mount/radecpath/add_point?ra_j2000_hours=13.2&dec_j2000_degs=-10.8&jd=2451545.7

/mount/radecpath/apply
Command the mount to begin following the interpolated path specified by the list of points added to the custom RA/Dec path. Any other targets the mount is following, including other custom RA/Dec paths, will be immediately overridden by this command.

If the path buffer has fewer than 2 points, an error 400 Bad Request will be returned.

Parameters
None

Example
Begin following the custom RA/Dec path:
http://localhost:8220/mount/radecpath/apply

/mount/custom_path/new
Begin building a new custom path using the specified coordinate type.

This command can be followed by one or more calls to **/mount/custom_path/add_point_list** to build a list of timestamped coordinates that the mount should pass through. After the list is built, a call to **/mount/custom_path/apply** will begin the process of following this interpolated path.

Parameters
type (string; required): The type of coordinates to use for the path. See **/mount/goto_coord_pair** for a description of supported coordinate types.

Example
Begin a new custom path that will use topocentric Alt-Az coordinates:
http://localhost:8220/mount/custom_path/new?type=altaz_topocentric

Begin a new custom path that will use raw motor coordinates:
http://localhost:8220/mount/custom_path/new?type=raw

/mount/custom_path/add_point_list
Add a collection of points to the custom path.

To accommodate the potentially large set of data that will be sent with this command, the HTTP POST method is used instead of HTTP GET.

Here is an example of the HTTP request that adds the following two points:
(2451545.5, 1.2, 3.4),
(2451545.6, 1.3, 3.5)

```
POST /mount/custom_path/add_point_list? HTTP/1.1
Accept-Encoding: identity
Content-Type: application/x-www-form-urlencoded
Content-Length: 68
Host: localhost:8220
User-Agent: Python-urllib/3.7
Connection: close
data=2451545.5000000000%2C1.2%2C3.4%0A2451545.6000000001%2C1.3%2C3.5
```

Notice that:

- The `POST` method is used
- The content-type is `application/x-www-form-urlencoded`
- The points are included with body of the request using the keyword “data”, followed by a URL-encoded version of the string:
  ```
  2451545.5000000,1.2,3.4
  2451545.6000000,1.3,3.5
  ```
- In the URL-encoding scheme, comma characters are replaced by “%2C”, and newline (\n) characters are replaced by (%0A)

Most HTTP libraries will have a built-in mechanism for including POST or “form” data.

Parameters

data (string; required): The URL-encoded string containing a list of comma-separated (JD,Coord0,Coord1) values. This must be included in the body of the POST request, rather than in the URL.

Example

See `mount_custom_path_add_point_list()` in pwi4_client.py for an example of how to assemble a list of points and send them via a POST request in Python.

/mount/custom_path/apply

Command the mount to begin following the interpolated path specified by the list of points added to the custom path. Any other targets the mount is following, including other custom paths, will be immediately overridden by this command.

If the path buffer has fewer than 2 points, or if **/mount/custom_path/new** has never been called. an error 400 Bad Request will be returned.

Parameters

None

Example

Begin following the custom path:
http://localhost:8220/mount/custom_path/apply

/focuser/enable
Begin enabling servo control on the motor for the active focuser.

The motors on some PlaneWave focusers can be either in the Enabled or Disabled state. In the Disabled state, there is typically no voltage going to the motor. When the motor is in the Enabled state, the axis is under servo control and motion can be commanded electronically.

For focusers that do not support enabling/disabling servo control, this command will have no effect.

After this call, monitor the value of **focuser.is_enabled**. Once the axis is fully enabled, the value should be `true`. Focuser motors should typically take less than 1 second to enable.

Axes can become disabled during operation, either as a reaction to an error condition in order to protect the equipment, or due to an external request to disable the motors; for example, by clicking the Disable button on the GUI.

Parameters
None

Example
Enable the focuser motor:
http://localhost:8220/focuser/enable

/focuser/disable
Disable servo control on the motor for the active focuser.

For focusers that do not support enabling/disabling servo control, this command will have no effect.

After this call, the value of **focuser.is_enabled** should transition to `false`.

Parameters
None

Example
Disable the focuser motor:
http://localhost:8220/focuser/disable

/focuser/goto
Begin moving the active focuser to the specified target position, in microns.

While the focuser is moving to the target, **focuser.is_moving** should return `true`.

Parameters
target (float; required): The target position, in microns

Example
Move the focuser to 5000 microns:
http://localhost:8220/focuser/goto?target=5000

/focuser/stop
Stop any movement on the active focuser.

Parameters
None

Example
Stop the focuser:
http://localhost:8220/focuser/stop

/rotator/enable
Begin enabling servo control on the motor for the active rotator.

The motors on some PlaneWave rotators can be either in the Enabled or Disabled state. In the Disabled state, there is typically no voltage going to the motor. When the motor is in the Enabled state, the axis is under servo control and motion can be commanded electronically.

For rotators that do not support enabling/disabling servo control, this command will have no effect.

After this call, monitor the value of **rotator.is_enabled**. Once the axis is fully enabled, the value should be `true`. Rotator motors should typically take less than 1 second to enable.

Axes can become disabled during operation, either as a reaction to an error condition in order to protect the equipment, or due to an external request to disable the motors; for example, by clicking the Disable button on the GUI.

Parameters
None

Example
Enable the rotator motor:
http://localhost:8220/rotator/enable

/rotator/disable
Disable servo control on the motor for the active rotator.

For rotators that do not support enabling/disabling servo control, this command will have no effect.

After this call, the value of **rotator.is_enabled** should transition to `false`.

Parameters
None

Example
Disable the rotator motor:
http://localhost:8220/rotator/disable

/rotator/goto_mech

Move the active rotator to the specified location, expressed in mechanical coordinates.

The mechanical position is useful for determining where a rotator will be within its total range of motion, and may be useful for taking flat-field images at a consistent orientation regardless of where in the sky the telescope is pointing. Mechanical positions can also be useful for parking the rotator at a consistent position; for example, so that the camera cables are hanging down.

Parameters

degs (float; required): The target mechanical position, in decimal degrees

Example

Move the rotator to a mechanical position of 45.5 degrees:
http://localhost:8220/rotator/goto_mech?degs=45.5

/rotator/goto_field

Move the active rotator to the specified location, expressed in on-sky "field" coordinates.

The field position takes into account both the mechanical position of the rotator and the amount of on-sky rotation induced by the geometry of the mount. For an equatorial mount, the field position and mechanical position are roughly equivalent (perhaps with a constant offset). For an alt-az mount, the field position will depend on where in the sky the telescope is pointing.

The "field angle" is similar to the position angle (PA) that the camera would observe on sky, but there may be a constant offset between the reported field angle and the true PA that would need to be accounted for.

Parameters

degs (float; required): The target field position, in decimal degrees

Example

Move the rotator to a field position of 12.3 degrees:
http://localhost:8220/rotator/goto_field?degs=12.3

/rotator/offset

Move the active rotator by the specified amount relative to the current location. For example, if the rotator is currently at a mechanical position of 90 degrees, sending an offset of -30 degrees will result in the rotator moving to a position of 60 degrees.

Parameters

degs (float; required): The distance to move, in decimal degrees

Example

Move the rotator by 10 degrees relative to the current position:
http://localhost:8220/rotator/offset?degs=10

/rotator/stop

Stop any movement on the active rotator.

Parameters
None

Example
Stop the rotator motor:
http://localhost:8220/rotator/stop

/fans/on
Turn on all fans that are connected and controlled by PWI4

Parameters
None

Example
Turn on all fans
http://localhost:8220/fans/on

/fans/off
Turn off all fans that are connected and controlled by PWI4

Parameters
None

Example
Turn off all fans
http://localhost:8220/fans/off

/m3/goto
Begin moving the M3 port selector mirror to the specified port.

This command only applies to telescope systems that have an M3 port selector, such as the CDK700, RC700, and PW1000.

Parameters
port (integer; required): The target Nasmyth port. May be either “1” or “2”.

Example
Begin moving M3 to Port 1:
http://localhost:8220/m3/goto?port=1

/m3/stop
Stop any in-progress moves on the M3 port selector.

Parameters
None

Example
Bring M3 to a stop:
http://localhost:8220/m3/stop

/virtualcamera/take_image
Generate a simulated FITS images of a starfield centered on the current telescope pointing position.

This feature requires that the Kepler/UC4 catalog is installed. This is the same catalog that is used by PWI4 to platesolve images when building a pointing model.

These simulated images can be used to simulate building a pointing model with the API interface. The generated images can be platesolved, and the resulting RA/Dec can be added to the model.

The response is a file of type "image/fits" with the default filename "image.fits".

Parameters
None

Example
Generate and download a FITS image of a simulated starfield:
http://localhost:8220/virtualcam/take_image

/temperatures/pw1000
For instances of PWI4 that communicate with the PW1000 telescope system, this command returns a report of the current temperature sensor values from that telescope.

A sample response is:

```
temperature.primary=-999.000
temperature.secondary=-999.000
temperature.m3=-999.000
temperature.ambient=-999.000
```

This interface will likely be deprecated soon in favor of a more generalized temperature reporting interface in the standard status response.

Parameters
None

Example
Request the temperatures from a PW1000 telescope system:
http://localhost:8220/temperatures/pw1000

/autofocus/start

Begin an autofocus sequence. PWI4 will command the focuser to move to a series of positions and take images. The diameters of all detected stars from each image is measured, a function relating focus position to star diameter is fit, and the focus is commanded to move to the position that should yield the smallest star diameter.

Progress and results from this procedure can be retrieved in the status response under the “autofocus.*” section; for example, **autofocus.is_running** can be used to monitor completion of the process.

Parameters

client_id (string; optional): An optional ID that identifies the client that issued the command. The ID can be any unique string. As the autofocus process produces log output that can be collected via /autofocus/log, each client can retrieve its own set of pending log messages.

Example

Start a new autofocus run using a unique client ID:
http://localhost:8220/autofocus/start?client_id=myclientname_1A2B3C4D

/autofocus/stop

Stop any in-progress autofocus sequence.

Parameters

None

Example

Stop the autofocus sequence:
http://localhost:8220/autofocus/stop

/autofocus/log

Retrieve a list of enqueued log messages from the autofocus sequence.

If client_id is specified, that client will have its own queue of messages. This way, if two clients are both trying to read the log messages, they won’t interfere with each other.

Once a message has been read from the client_id’s queue using this command (or the default queue if no client_id is specified), the message is removed from that queue and will not be included in future requests to /autofocus/log.

The response consists of a “text/plain” body with one log message per line, separated by newlines (\n).

Parameters

client_id (string; optional): An optional ID that identifies the client that issued the command. The ID can be any unique string as long as it matches a client_id that was used in a call to **/autofocus/start**.

Example

Read the enqueued autofocus log messages for a specific client ID:
http://localhost:8220/autofocus/log?client_id=myclientname_1A2B3C4D

/internal/crash

The /internal/crash endpoint can be used to intentionally produce an Error 500 response. It can be used to test how your HTTP library reacts to this error code. The "crash" that occurs is isolated to just this request; the rest of PWI4 and the HTTP server should be unaffected.

Parameters

None

Example

Produce an Error 500 response:
http://localhost:8220/internal/crash

Status values

The values included in a status response are described below.

pwi4.version

Type: string

The version of PWI 4 that is currently running. Version numbers are structured as follows:

[major].[minor].[revision] [beta_num_or_final]

where:

[major] will always be 4 for the PWI 4 series of control software

[minor] is incremented when substantial new features or hardware support are added

[revision] is incremented when smaller additions or bugfixes are made

[beta_num] may **optionally** contain the word "beta " followed by a number; for example: "beta 13".

Versions with features that may not have been extensively tested yet typically have the "beta" tag. Eventually these versions get promoted to "final" releases, which do not contain the "beta" tag.

Sample values:
"4.0.8"
"4.0.9 beta 13"

pwi4.version_field[0]

Type: int

The "major" portion of the PWI4 version. For PWI 4, this should always be "4".

pwi4.version_field[1]
Type: int

The “minor” portion of the PWI4 version.

pwi4.version_field[2]
Type: int

The “revision” portion of the PWI4 version

pwi4.version_field[3]
Type: int

The “beta” iteration of the PWI4 version. If this release is a “final” (non-beta) version, this field will have the value 99.

response.timestamp_utc
Type: timestamp

The UTC time, according to the PC clock, at which the text of this status response was generated.

This value can be compared to the time at which the client initiated the request or received the response (if those times are derived from the same PC clock, or another clock that is synchronized to the same source as the PC clock) to measure the overhead in transmitting the request or response.

This value can also be compared to other timestamps included in a status response to determine the age of various telemetry values reported by the system.

site.latitude_degs
Type: float

The configured latitude of the telescope, in degrees. Negative values are south of the equator. The expected range is from -90 to +90 degrees.

site.longitude_degs
Type: float

The configured longitude of the telescope, in degrees. Negative values are west of the prime meridian. The typical range will be from -180 to +180 degrees, although values from -360 to +360 are allowed.

site.height_meters
Type: float

The configured height (elevation) of the telescope above sea level, in meters.

site.lmst_hours

Type: float

The local mean sidereal time at the telescope, in decimal hours. The expected range is from 0 to 24 hours.

mount.is_connected

Type: boolean

true if PWI4 has an active connection to the mount hardware.

mount.geometry

Type: int

The geometry of configured mount. Possible values:

- 0: Alt-Az
- 1: Equatorial Fork
- 2: German Equatorial

mount.timestamp_utc

Type: timestamp

The UTC time (according to the PC clock) at which the status values of the mount were last updated. If PWI4 is not connected to the mount, the value will be 0001-01-01 00:00:00.0000

mount.julian_date

Type: float

The Julian day (JD) at the time the mount's position was last sampled. The Julian day is a continuous count of whole and fractional days, where a JD of 2451544.5 corresponds to midnight on January 1, 2000.

mount.slew_time_constant

Type: float

The current value of the slew time constant used to calculate trajectory intercept velocities for the mount. For most PWI4 mount drivers, at each iteration of the communication loop a velocity is sent that will cause the mount to intersect the target trajectory in S seconds, where S is the slew time constant. The value of S should be at least twice the communication loop interval, or else the mount will have gone past its target position before a refined velocity can be sent.

Smaller slew time constant values lead to more aggressive approaches to a target, but if the value is too small it can cause instability and overshoot. The final approach to a target tends to look like an exponential decay, with a half life that is related to the slew time constant.

mount.ra_apparent_hours

Type: float

The telescope's current position in hours of apparent right ascension. Pointing model corrections (if present) have been applied. Relative to a J2000 target, apparent coordinates account for precession, nutation, and annual aberration, but not refraction.

If PWI4 is not connected to the mount, the value will be 0.

mount.dec_apparent_degs

Type: float

The telescope's current position in degrees of apparent declination. Pointing model corrections (if present) have been applied. Relative to a J2000 target, apparent coordinates account for precession, nutation, and annual aberration, but not refraction.

If PWI4 is not connected to the mount, the value will be 0.

mount.ra_j2000_hours

Type: float

The telescope's current position in hours of J2000 right ascension. Pointing model corrections (if present) have been applied.

If PWI4 is not connected to the mount, the value will be 0.

mount.dec_j2000_degs

Type: float

The telescope's current position in degrees of J2000 declination. Pointing model corrections (if present) have been applied.

If PWI4 is not connected to the mount, the value will be 0.

mount.target_ra_apparent_hours

Type: float

The position of the target the telescope is trying to acquire or follow, in hours of apparent right ascension. As soon as a new target is submitted, this value (and mount.target_dec_apparent_degs) will update to reflect the new target coordinates.

This value includes the baseline target coordinates as well as the total accumulated RA/Dec and Path/Transverse offsets. As of PWI 4.0.9, it does not incorporate the native Axis0/Axis1 offsets, but this may be subject to change in the future.

If PWI4 is not connected to the mount, or if the mount is not currently following a target, this value will be 0.

mount.target_dec_apparent_degs

Type: float

The position of the target the telescope is trying to acquire or follow, in degrees of apparent declination. Refer to mount.target_ra_apparent_hours for more information.

If PWI4 is not connected to the mount, or if the mount is not currently following a target, this value will be 0.

mount.azimuth_degs

Type: float

The telescope's current position in degrees of azimuth. North is defined as 0 degrees, and East is 90 degrees. This value incorporates pointing model corrections. For the raw reading of the Azimuth motor position in an Alt/Az mount, refer to mount.axis0.position_degs.

If PWI4 is not connected to the mount, this value will be 0.

mount.altitude_degs

Type: float

The telescope's current position in degrees of altitude. 0 degrees is the horizon, and 90 degrees is zenith. This value incorporates pointing model corrections. For the raw reading of the Altitude motor position in an Alt/Az mount, refer to mount.axis1.position_degs.

If PWI4 is not connected to the mount, this value will be 0.

mount.is_slewing

Type: boolean

When the mount is first commanded to follow a new target, this flag reports true while the mount is moving to acquire the target. Once the target is acquired, or if the movement is stopped, this flag reports false.

If PWI4 is not connected to the mount, the value will be false.

mount.is_tracking

Type: boolean

When the mount is trying to follow a target, this flag reports true. If mount.is_slewing is also true, then the mount has not yet acquired the target. When the mount is stopped, this flag reports false.

mount.field_angle_here_degs

Type: float

The amount of field rotation induced by the geometry of the mount at the current telescope position.

For equatorial fork mounts with good polar alignment, this number will be close to 0.

For Alt-Az mounts, this number will change depending on where the telescope is pointing. On a well-leveled Alt-Az mount, this number will correlate closely with the parallactic angle at the current pointing position.

Because this number reports the calculated field rotation angle for the last-sampled position of the telescope, this number may change dramatically over the course of a slew to a new target. In many cases, the value mount.field_angle_at_target_degs is more useful since it reports what the field rotation angle will be once the telescope has arrived at its target.

This value incorporates corrections from the pointing model if one is present.

This value only accounts for amount of field rotation caused by the mount. The effects of an instrument rotator are handled separately.

mount.field_angle_at_target_degs

Type: float

The amount of field rotation induced by the geometry of the mount at the target (destination) position of the telescope.

For equatorial fork mounts with good polar alignment, this number will be close to 0.

For Alt-Az mounts, this number will change depending on where the target is located in the sky. On a well-leveled Alt-Az mount, this number will closely correlate with the parallactic angle at the target position.

On an Alt-Az mount, this value will typically jump to a new value when a new target is requested, and then will slowly change as the target moves across the sky.

This value incorporates corrections from the pointing model if one is present.

This value only accounts for amount of field rotation caused by the mount. The effects of an instrument rotator are handled separately.

mount.field_angle_rate_at_target_degs_per_sec

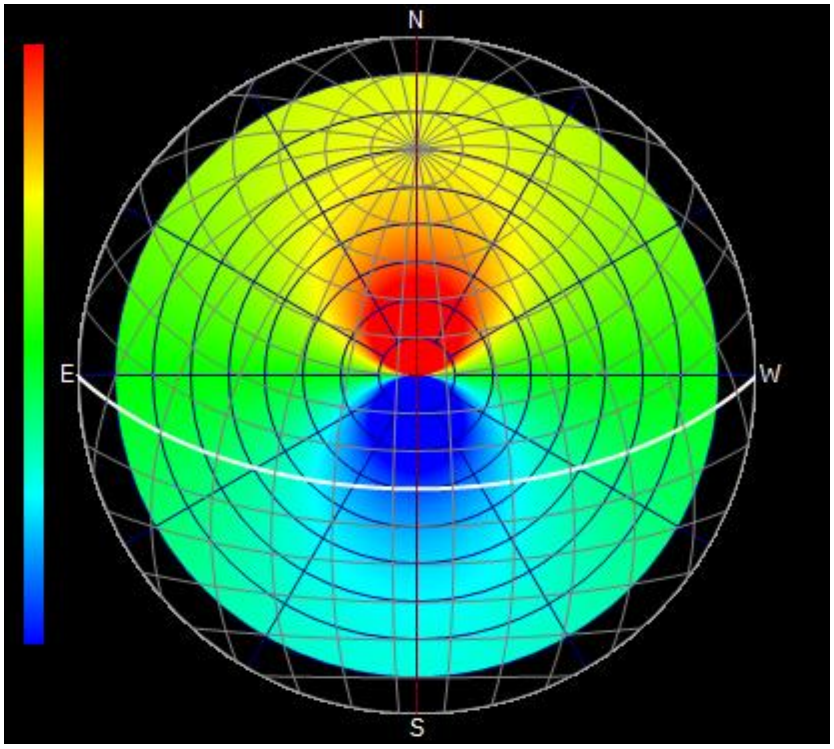
Type: float

The rate at which **mount.field_angle_at_target_degs** is changing, in degrees per second. For an Alt-Az mount, a rotator could periodically be commanded to move at this rate to compensate for field rotation when imaging a sidereal target. This rate will change gradually and continuously as an Alt-Az telescope tracks across the sky.

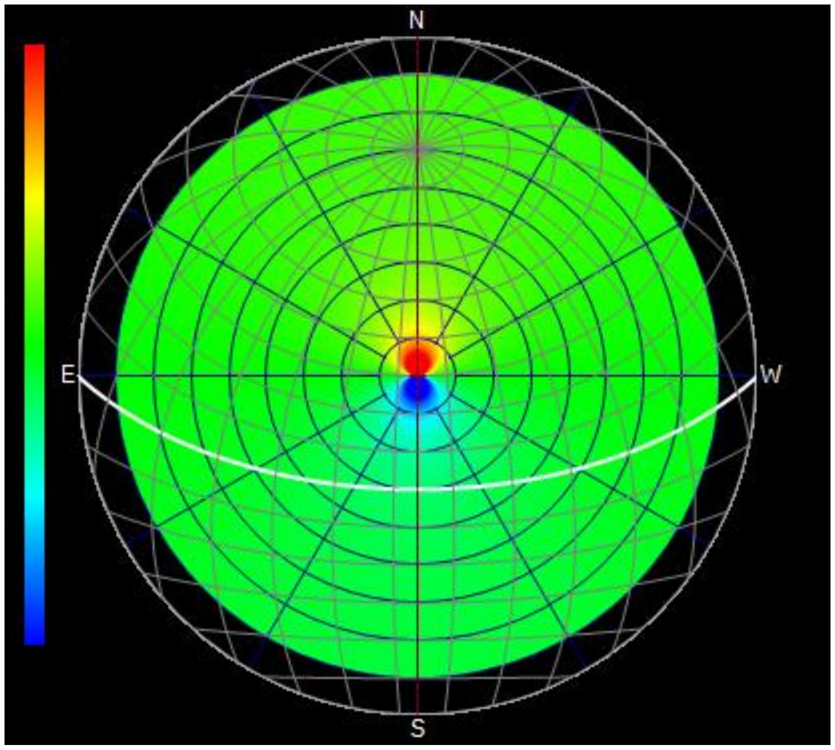
When the mount is commanded to move to a new position, the field rotation rate is calculated for that target position. The rotator can begin moving at the reported rate so that the system is ready to begin imaging as soon as the mount arrives at the target. In addition, the sign of the reported rate along with the reported mechanical position could be used to determine if the rotator should be commanded to move away from a hardstop prior to derotating and starting the exposure so that derotation can run for a longer time before the rotator needs to unwrap.

For the RC700 and PW1000 telescopes where the Nasmyth-mounted rotator is not directly coupled to the telescope, it will also be necessary to compensate for the rate of Altitude motion, as reported by **mount.axisN.measured_velocity_degs_per**_sec.

The following figure shows field rotation rates for an Alt-Az telescope located at 30 degrees latitude. In the color map, blue represents a rate <= -30 arcseconds/sec, and red represents a rate >= +30 arcseconds/sec. This range of rates covers most parts of the sky except for regions at high altitudes near the meridian.



Re-scaling the color map so that blue is <= -120 arcseconds/sec and red is >= +120 arcseconds/sec, you can get a sense of how rapidly the rates increase as you approach zenith:



mount.path_angle_at_target_degs

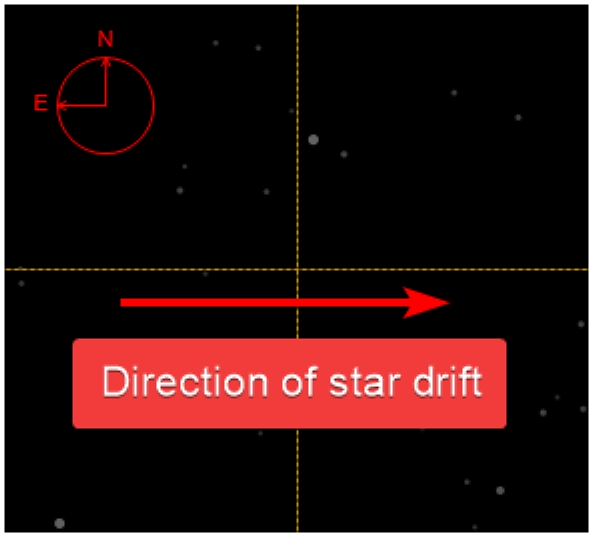
Type: float

Reports the angle (in degrees) at which the target followed by the mount is moving in the imaging plane. This is primarily useful when tracking fast-moving non-sidereal targets, such as satellites.

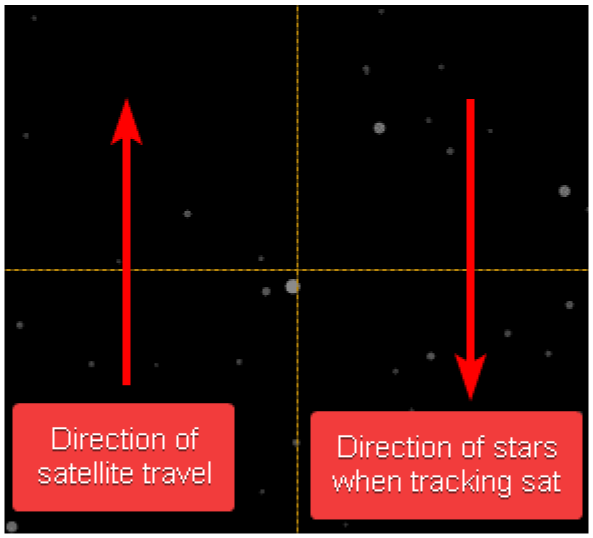
Imagine a camera mounted at a fixed orientation on the telescope so that movement of Axis0 (Azimuth / RA) corresponds to the left-right direction in the image, and movement of Axis1 (Altitude / Dec) corresponds to the up-down direction in the image. If the telescope were stopped and the moving target were allowed to pass through the image, then:

- A target with a path angle of 0 would enter the bottom of the image and exit the top of the image
- A target with a path angle of 90 would enter the left side of the image and exit the right side of the image.

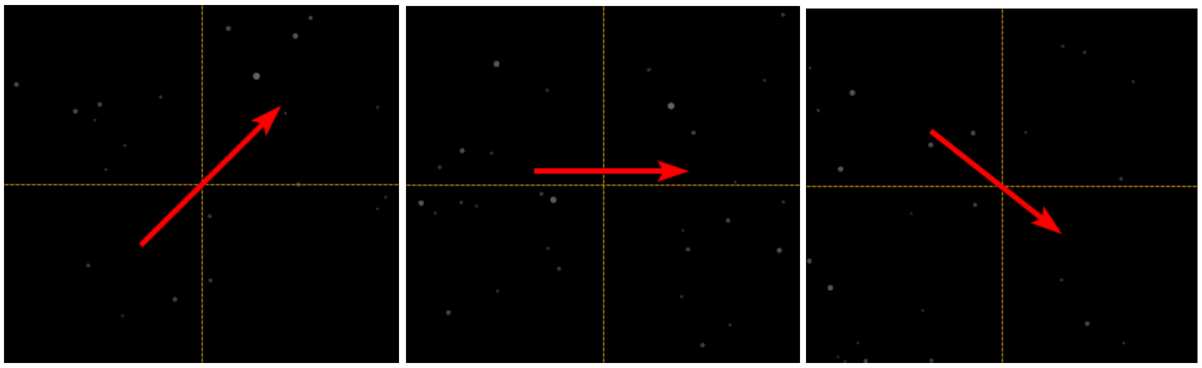
For example, consider an equatorially-mounted telescope tracking a sidereal target. In this case, the left-right direction in the ideal camera described above would correspond to the direction of Right Ascension. If the telescope were stopped, stars would drift to the right in this image, so when tracking a star the path angle would be 90 degrees. The same would be true for a perfect Alt-Az telescope in the Northern hemisphere at the moment it tracks through the southern meridian (Azimuth = 180 degrees):



Now imagine tracking a satellite that is currently moving primarily in the Altitude direction of an Alt-Az telescope. In this case, the reported path angle would be roughly 0 degrees:


Direction of satellite travel
Direction of stars when tracking sat

As the satellite moves to a new part of the sky, the direction of satellite travel within the camera (and, equivalently, the direction of the streaks produced by background stars while following the satellite) will gradually change direction. In the sequence below, we see the direction of travel gradually move to 45 degrees, 90 degrees, and eventually 135 degrees. Note that for satellites this sort of “rotation of direction” occurs on both Alt-Az and Equatorial telescope.



To compensate for this effect (e.g. to keep the direction of background star streaks consistent in all images while tracking a satellite), the instrument rotator can be moved based on the reported path angle value.

mount.path_angle_rate_at_target_degs_per_sec

Type: float

The rate at which **mount.path_angle_at_target_degs** is changing, in degrees per second.

mount.distance_to_sun_degs

Type: float

The great-circle distance from the current pointing direction of the telescope and the sun, in degrees.

Note that this is only valid if:

- The latitude, longitude, and time are correct within PWI4 (for calculating the position of the Sun)
- PWI4 has an accurate notion of where the telescope is physically pointing. Specifically: the telescope has been correctly homed (if needed) and the current pointing model produces relatively accurate pointing.

For mounts that need to be homed: because PWI4 does not have an accurate notion of where the mount is pointing before the axes have been homed, this value will not be correctly calculated. If this value is to be used to perform some sort of solar avoidance, it will be necessary to use a different mechanism while homing.

mount.axis0_wrap_range_min_degs

Type: float

The minimum mechanical angle of the Axis 0 (Azimuth / RA) motor to be used when determining how to unwrap the motor to reach a target.

Most PlaneWave mounts allow more than one full rotation in Axis 0. When slewing to a target, there may be more than one axis position that can achieve the target position. For example, in an Alt-Az mount with a mechanical range from -350 to +350 degrees, pointing to an Azimuth of 90 degrees can be achieved either with a mechanical position of -270 or +90 degrees.

The wrap range minimum defines a 360-degree range of mechanical positions that will be used by PWI4 to follow a target. In the previous example, if axis0_wrap_range_min_degs is -45, then PWI4 will choose the mechanical position that lies between -45 and +315, so to point at Azimuth 90 it will use mechanical position +90. If axis0_wrap_range_min_degs was instead set to -300, then mechanical angles from -300 to +60 would be considered, and -270 would be used to point at Azimuth 90.

This value may be changed automatically by PWI4 when slewing to certain targets. For example, when following a satellite, PWI4 will automatically set this value so that the satellite can be followed from horizon to horizon without the axis needing to unwrap.

Additional unwrapping strategies are planned for future versions of PWI4, and this value may become optional or deprecated at that time.

mount.offsets.[axis]_*

A section containing values that describe the current offset state along all available axes. For more details on mount targeting offsets, refer to **/mount/offset**.

[axis] may be one of:

- ra
- dec
- axis0
- axis1
- path
- transverse

For example, mount.offsets.transverse_arcsec.total reports the current total offset being applied in the transverse direction (perpendicular to the direction of travel for the target).

mount.offsets.[axis]_arcsec.total

Type: float

The total offset currently being applied to the axis, in arcseconds.

This value is affected by offsets set via the **/mount/offset** command. For example:

- If [axis]_set_rate_arcsec_per_sec is set to 1, this value will begin increasing at a rate of 1 arcsecond per second
- If [axis]_set_total_arcsec is set to 10, this value will immediately jump to 10 (but will continue increasing at whatever rate was last set)
- If [axis]_reset is set, this value will immediately jump to 0 and remain until another offset command is sent
- If [axis]_add_gradual_offset_arcsec is set to 5 and [axis]_gradual_offset_seconds is set to 10, then this value will gradually increase by 5 arcseconds over the course of the next 10 seconds and then stop.

mount.offsets.[axis]_arcsec.rate

Type: float

The rate at which mount.offsets.[axis]_arcsec.total is increasing, in arcseconds per second.

This value is affected by the following **/mount/offset** sub-commands:

- [axis]_reset: Resets the rate to 0 (and resets total position to 0)
- [axis]_stop: Resets the rate to 0 (but maintains the total position)
- [axis]_stop_rate: Resets the rate to 0 (but gradual offsets may continue)
- [axis]_set_rate_arcsec_per_sec: Change the rate to the specified value

Note that this rate only refers to continuously-applied commanded rates. The gradual_offset mechanism effectively adds an additional rate, but it is represented separately.

mount.offsets.[axis]_arcsec.gradual_offset_progress

Type: float

A value from 0.0 to 1.0 indicating how much of the most recent gradual_offset has been applied. When the gradual offset is complete, this value will remain at 1.0.

For example, if [axis]_add_gradual_offset_arcsec is set to 5 and [axis]_gradual_offset_seconds is set to 10, then:

- 1 second after sending the offset command, this value will return 0.1, and 0.5 arcseconds will have been added to the total
- After 5 seconds, this value will be 0.5 and 2.5 arcseconds will have been added to the total offset
- After 10 seconds, this value will be 1.0, and 5 arcseconds will have been added to the total. Nothing further will be added to the total (unless a separate rate is active).
- After 20 seconds, this value will still be 1.0

mount.spiral_offset.x

Type: int

The current X position of the spiral offset grid. For a full description of the spiral offset mechanism, see **/mount/spiral_offset/new** and **/mount/spiral_offset/next**.

mount.spiral_offset.y

Type: int

The current Y position of the spiral offset grid. For a full description of the spiral offset mechanism, see **/mount/spiral_offset/new** and **/mount/spiral_offset/next**.

mount.spiral_offset.x_step_arcsec

Type: float

The size of each spiral grid offset in the X direction, in arcseconds. This value is set by **/mount/spiral_offset/new** and affects how much the mount will move with each call to **/mount/spiral_offset/next** or **/mount/spiral_offset/previous**.

mount.spiral_offset.y_step_arcsec

Type: float

The size of each spiral grid offset in the Y direction, in arcseconds. This value is set by **/mount/spiral_offset/new** and affects how much the mount will move with each call to **/mount/spiral_offset/next** or **/mount/spiral_offset/previous**.

mount.axisN.*

Contains status information about the motorized axes controlled by the mount

axis0: The primary axis of rotation. Corresponds to the Azimuth axis in an Alt-Az mount, or the Right Ascension axis in an Equatorial mount.

axis1: The secondary axis of rotation. Corresponds to the Altitude axis in an Alt-Az mount, or the Declination axis in an Equatorial mount.

mount.axisN.is_enabled

Type: boolean

true if the motor for this axis is powered on and under servo control

false if the motor is not under servo control

For a direct-drive motor, an enabled motor will resist being pushed out of position, and a disabled motor can normally be moved freely by hand.

An enabled motor may transition to the disabled state if an error is detected in the control system – for example, if the motor is using too much current, or is too far away from its expected position.

mount.axisN.rms_error_arcsec

Type: float

The root-mean-square value of the recent mount.axisN.dist_to_target_arcsec measurements. Provides a basic estimate of how well the motor is tracking its intended target.

If the PWI4 is not connect to the mount, the motor is disabled, or the motor is not following a target, this value will be 0.

mount.axisN.dist_to_target_arcsec

Type: float

The distance from the motor's last-sampled position and the desired position of the axis at that time. When a new distant target is submitted, this value will be large at first and will gradually towards zero as the motor approaches the target.

mount.axisN.servo_error_arcsec
Type: float

The distance from the desired position of the motor and the sampled position of the encoder, as reported by the servo control electronics.

mount.axisN.min_mech_position_degs
Type: float

The minimum mechanical position that is allowed by PWI4 on this axis, in degrees. If the axis tries to move beyond this position, the target will be clamped to this position. This value can be compared directly to **mount.axisN.position_degs**.

mount.axisN.max_mech_position_degs
Type: float

The maximum mechanical position that is allowed by PWI4 on this axis, in degrees. If the axis tries to move beyond this position, the target will be clamped to this position. This value can be compared directly to **mount.axisN.position_degs**.

mount.axisN.target_mech_position_degs
Type: float

The mechanical position that the PWI4 trajectory generator is currently trying to reach, in degrees.

If, for example, an Altitude axis is trying to slew to a target that is below the horizon and this is the current state:

```
position_degs = 45
min_mech_position_degs = 5
target_mech_position_degs = -3
```

then the axis will move to 5 degrees and then stop. As the target rises, target_mech_position_degs might eventually rise above 5 degrees and the mount will be ready to intercept and follow the target.

mount.axisN.position_degs
Type: float

The raw position of the motor encoder, in degrees. This value is unaffected by the pointing model or refraction calculations.

mount.axisN.position_timestamp
Type: float

The UTC time when the axis telemetry was sampled

mount.axisN.max_velocity_degs_per_sec

Type: float

The maximum velocity that PWI4 will command during axis movement, in degrees per second. If an axis is at 90 degrees an is trying to move to 270 degrees with a maximum allowed velocity of 20 degrees per second, the axis will ramp up to 20 degs/sec with the acceleration reported by **mount.axisN.acceleration_degs_per_sec**, and then will decelerate down to the velocity of the target. For short moves, the axis might not reach this maximum velocity if the motor needs to begin decelerating before it is has finished accelerating.

mount.axisN.setpoint_velocity_degs_per_sec

Type: float

The velocity that the axis is currently trying to achieve, in degrees per second. During the acceleration phase of a movement, this value will increase in magnitude, potentially up to **mount.axisN.max_velocity_degs_per_sec**.

mount.axisN.measured_velocity_degs_per_sec

Type: float

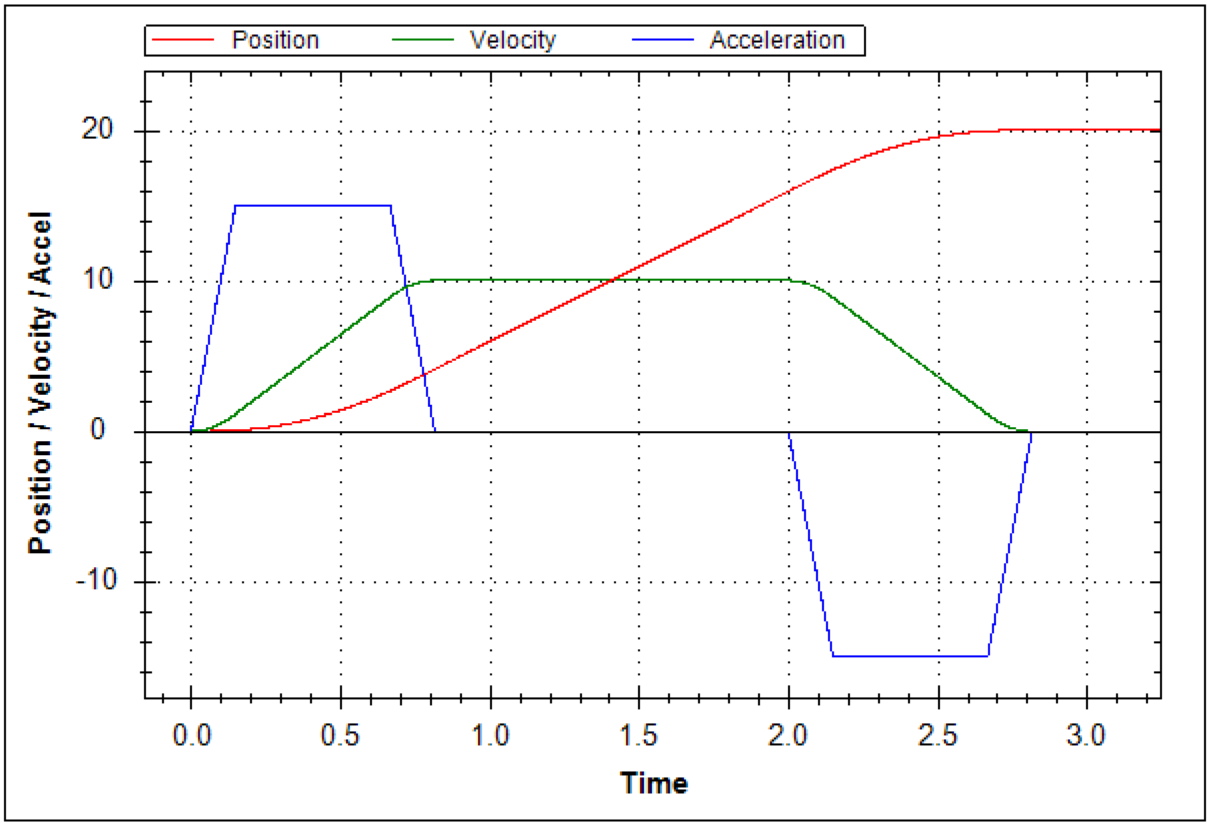
The actual measured velocity of axis movement, in degrees per second. The difference between this value and **mount.axisN.setpoint_velocity_degs_per_sec** is the velocity error of the servo drive. If, for example, the axis is stuck against a hardstop, the setpoint velocity might be several degrees per second (the velocity that the controller is trying to achieve) but the measured velocity might be close to zero.

mount.axisN.acceleration_degs_per_sec

Type: float

The acceleration (and, in most cases, deceleration) used by the axis when trying to achieve a different velocity, in degrees per second squared. A typical slew o a new target will consist of an acceleration phase, a constant-velocity phase (if there is still enough distance to cover before deceleration must begin), and a deceleration phase:

This graph shows the position, velocity, and acceleration that might occur during a 20 degree movement with 15 degree/sec/sec acceleration and a maximum velocity of 10 degrees/sec:



mount.axisN.measured_current_amps

Type: float

The measured motor current, in amps. Motor current can be monitored and potentially used to diagnose issues such as friction and imbalance in the axis.

mount.model.filename

Type: string

The name of the pointing model file that is currently loaded. If no model is loaded, this value will have zero length.

mount.model.num_points.total

Type: int

The total number of calibration points in the active pointing model.

mount.model.num_points.enabled

Type: int

The number of calibration points in the active pointing model that are contributing to the model fit.

mount.model.rms_error_arcsec

Type: float

The root-mean-square error of the calibration points compared to the current best fit of the model

focuser.is_connected

Type: boolean

`true` if PWI4 is connected to a focuser

focuser.is_enabled

Type: boolean

`true` if the active focuser motor is energized and under servo control

focuser.position

Type: float

The position of the active focuser. Currently all supported focusers report mechanical positions in microns. For a focuser that the instrumentation mounts to (e.g. the standard PW1000 instrument focuser, or the Series 5 focuser), this number will be directly related to the position of the instrument relative to the focal plane. For secondary mirror (M2) focusers, this number will need to be scaled by some factor to determine how far the focal plane has moved.

focuser.is_moving

Type: boolean

`true` if the focuser is moving in response to a motion command. Once motion has stopped, this value will transition to `false`.

rotator.is_connected

Type: boolean

`true` if PWI4 is connected to a rotator. In some cases, such as the PW1000, the rotator connection is shared with the mount connection. As a result, connecting to the mount will also cause **rotator.is_connected** to transition to `true`.

If the connection is lost, or if a disconnect is requested, this value will transition to `false`.

rotator.is_enabled

Type: boolean

`true` if the active rotator motor is energized and under servo control.

This value may transition to `false` if an error occurs and the motor needs to shut down for protection – for example, if the motor detects that it is stuck or drawing too much current.

For direct-drive rotators such as the PW1000 Rotator, when the motor is disabled it is possible to move the rotator by hand.

rotator.mech_position_degs

Type: float

The mechanical position of the active rotator, in degrees. The mechanical position can be used to determine where the rotator is in its range of motion and to account for issues such as cable wrap and camera orientation relative to the mechanical telescope system.

When the rotator is commanded to move using **/rotator/goto_mech**, this field should closely match the commanded target position once the rotator finishes moving.

Contrast this field with **rotator.field_angle_degs**, which relates to the rotation angle of the sky as viewed from the camera and takes into account mount geometry as well as rotator position.

rotator.field_angle_degs

Type: float

The orientation of the sky as seen from a camera mounted to the rotator. This value takes into account both the rotator mechanical position and the amount of rotation induced by the mount geometry for the current pointing direction. This value typically equals the Position Angle of the sky as viewed from the camera, plus some constant offset depending on how the camera was mounted to the rotator.

When the rotator is commanded to move using **/rotator/goto_field**, this value should closely match the commanded target orientation once the rotator is finished moving.

rotator.is_moving

Type: boolean

`true` if the active rotator is currently undergoing any electronically-driven movement. This can include a goto command or a jog movement being executed, as well as field derotation being performed.

rotator.is_slewing

Type: boolean

`true` if the active rotator is currently moving to a target position in response to a **/rotator/goto_mech** or **/rotator/goto_field** command.

m3.port

Type: int

For systems that include an M3 Nasmyth port selector, this value indicates which port is currently active. Possible values are:

0: The M3 mirror is between ports

1: M3 is pointed at Nasmyth Port 1

2: M3 is pointed at Nasmyth Port 2

autofocus.is_running

Type: boolean

`true` if an autofocus sequence is currently running, either in response to a **/autofocus/start** command or as a result of the Autofocus button being clicked in the GUI.

Once the autofocus sequence completes, this value transitions to `false` and the results can be inspected in the other **autofocus.*** status fields.

autofocus.success

Type: boolean

`true` if the most recent autofocus sequence successfully determined a best focus position. This value should be inspected only after **autofocus.is_running** transitions from `true` to `false`.

autofocus.best_position

Type: float

If **autofocus.success** is `true`, this value contains the best-focus position that was determined by the autofocus sequence. After a successful autofocus, the focuser will automatically be commanded to move to this position.

autofocus.tolerance

Type: float

If **autofocus.success** is `true`, this value contains an estimate for how far the focuser can move from the best-focus position to produce a 3% increase in the RMS diameter of a star.

Appendix A: Sample status response

Most commands will reply with a **status response** that gives a snapshot of the system at the time the response was generated. The response consists of keyword=value pairs on each line. A sample response is included below:

```
pwi4.version=4.0.14
pwi4.version_field[0]=4
pwi4.version_field[1]=0
pwi4.version_field[2]=14
pwi4.version_field[3]=99
response.timestamp_utc=2022-10-07 07:31:00.388598
site.latitude_degs=33.4999722222222
site.longitude_degs=-118
site.height_meters=50
site.lmst_hours=0
mount.is_connected=false
mount.geometry=0
mount.timestamp_utc=0001-01-01 00:00:00.0000
mount.julian_date=0
mount.slew_time_constant=0.5
mount.ra_apparent_hours=0
mount.dec_apparent_degs=0
mount.ra_j2000_hours=0
mount.dec_j2000_degs=0
mount.target_ra_apparent_hours=0
mount.target_dec_apparent_degs=0
mount.azimuth_degs=0
mount.altitude_degs=0
mount.is_slewing=false
mount.is_tracking=false
mount.field_angle_here_degs=0
mount.field_angle_at_target_degs=0
mount.field_angle_rate_at_target_degs_per_sec=0
mount.path_angle_at_target_degs=0
mount.path_angle_rate_at_target_degs_per_sec=0
mount.distance_to_sun_degs=0
mount.axis0_wrap_range_min_degs=0
mount.offsets.ra_arcsec.total=0
mount.offsets.ra_arcsec.rate=0
mount.offsets.ra_arcsec.gradual_offset_progress=1
mount.offsets.dec_arcsec.total=0
mount.offsets.dec_arcsec.rate=0
mount.offsets.dec_arcsec.gradual_offset_progress=1
mount.offsets.axis0_arcsec.total=0
mount.offsets.axis0_arcsec.rate=0
mount.offsets.axis0_arcsec.gradual_offset_progress=1
mount.offsets.axis1_arcsec.total=0
mount.offsets.axis1_arcsec.rate=0
mount.offsets.axis1_arcsec.gradual_offset_progress=1
mount.offsets.path_arcsec.total=0
mount.offsets.path_arcsec.rate=0
mount.offsets.path_arcsec.gradual_offset_progress=1
mount.offsets.transverse_arcsec.total=0
mount.offsets.transverse_arcsec.rate=0
mount.offsets.transverse_arcsec.gradual_offset_progress=1
mount.spiral_offset.x=0
mount.spiral_offset.y=0
mount.spiral_offset.x_step_arcsec=600
mount.spiral_offset.y_step_arcsec=600
mount.axis0.is_enabled=true
mount.axis0.rms_error_arcsec=0.00965543035281771
mount.axis0.dist_to_target_arcsec=-0.00315739269147343
mount.axis0.servo_error_arcsec=0
mount.axis0.min_mech_position_degs=-120
mount.axis0.max_mech_position_degs=480
mount.axis0.target_mech_position_degs=250.787657165704
mount.axis0.position_degs=250.787651985884
mount.axis0.position_timestamp=2022-10-06 21:17:18.0857
mount.axis0.max_velocity_degs_per_sec=15
mount.axis0.setpoint_velocity_degs_per_sec=0.00308531150221825
mount.axis0.measured_velocity_degs_per_sec=0.00308829694730799
mount.axis0.acceleration_degs_per_sec_sqr=7
mount.axis0.measured_current_amps=0.043664181666445
mount.axis1.is_enabled=true
mount.axis1.rms_error_arcsec=0.0099546319965143
mount.axis1.dist_to_target_arcsec=0.00186475335752145
mount.axis1.servo_error_arcsec=0
mount.axis1.min_mech_position_degs=15
mount.axis1.max_mech_position_degs=89.9
mount.axis1.target_mech_position_degs=34.3945753289812
mount.axis1.position_degs=34.3945804238319
mount.axis1.position_timestamp=2022-10-06 21:17:18.0857
mount.axis1.max_velocity_degs_per_sec=15
mount.axis1.setpoint_velocity_degs_per_sec=-0.00328307040035725
mount.axis1.measured_velocity_degs_per_sec=-0.00328239401250979
mount.axis1.acceleration_degs_per_sec_sqr=7
mount.axis1.measured_current_amps=0.0262907641655176
mount.model.filename=DefaultModel.pxp
mount.model.num_points_total=0
mount.model.num_points_enabled=0
mount.model.rms_error_arcsec=0
focuser.is_connected=false
focuser.is_enabled=false
focuser.position=5000.00143743542
focuser.is_moving=false
rotator.is_connected=false
rotator.is_enabled=false
rotator.mech_position_degs=0
rotator.field_angle_degs=86.5922129806722
rotator.is_moving=false
rotator.is_slewing=false
m3.port=1
autofocus.is_running=false
autofocus.success=false
autofocus.best_position=0
autofocus.tolerance=0
```