API server

This document describes Klipper's Application Programmer Interface (API). This interface enables external applications to query and control the Klipper host software.

Enabling the API socket

In order to use the API server, the klippy.py host software must be started with the `-a` parameter. For example:

```
~/klippy-env/bin/python ~/klipper/klippy/klippy.py ~/printer.cfg -a
/tmp/klippy_uds -l /tmp/klippy.log
```

This causes the host software to create a Unix Domain Socket. A client can then open a connection on that socket and send commands to Klipper.

See the Moonraker project for a popular tool that can forward HTTP requests to Klipper's API Server Unix Domain Socket.

Request format

Messages sent and received on the socket are JSON encoded strings terminated by an ASCII 0x03 character:

```
<json_object_1><0x03><json_object_2><0x03>...
```

Klipper contains a `scripts/whconsole.py` tool that can perform the above message framing. For example:

```
~/klipper/scripts/whconsole.py /tmp/klippy_uds
```

This tool can read a series of JSON commands from stdin, send them to Klipper, and report the results. The tool expects each JSON command to be on a single line, and it will automatically append the 0x03 terminator when transmitting a request. (The Klipper API server does not have a newline requirement.)

API Protocol

The command protocol used on the communication socket is inspired by json-rpc.

A request might look like:

```
{"id": 123, "method": "info", "params": {}}
```

and a response might look like:

{"id": 123, "result": {"state_message": "Printer is ready", "klipper_path": "/home/pi/klipper", "config_file": "/home/pi/printer.cfg", "software_version": "v0.8.0-823-g883b1cb6", "hostname": "octopi", "cpu_info": "4 core ARMv7 Processor rev 4 (v7l)", "state": "ready", "python_path": "/home/pi/klippy-env/bin/python", "log_file": "/tmp/klippy.log"}}

Each request must be a JSON dictionary. (This document uses the Python term "dictionary" to describe a "JSON object" - a mapping of key/value pairs contained within `{}`.)

The request dictionary must contain a "method" parameter that is the string name of an available Klipper "endpoint".

The request dictionary may contain a "params" parameter which must be of a dictionary type. The "params" provide additional parameter information to the Klipper "endpoint" handling the request. Its content is specific to the "endpoint".

The request dictionary may contain an "id" parameter which may be of any JSON type. If "id" is present then Klipper will respond to the request with a response message containing that "id". If "id" is omitted (or set to a JSON "null" value) then Klipper will not provide any response to the request. A response message is a JSON dictionary containing "id" and "result". The "result" is always a dictionary - its contents are specific to the "endpoint" handling the request.

If the processing of a request results in an error, then the response message will contain an "error" field instead of a "result" field. For example, the request: `{"id": 123, "method": "gcode/script", "params": {"script": "G1 X200"}}` might result in an error response such as: `{"id": 123, "error": {"message": "Must home axis first: 200.000 0.000 0.000 [0.000]", "error": "WebRequestError"}}`

Klipper will always start processing requests in the order that they are received. However, some request may not complete immediately, which could cause the associated response to be sent out of order with respect to responses from other requests. A JSON request will never pause the processing of future JSON requests.

Subscriptions

Some Klipper "endpoint" requests allow one to "subscribe" to future asynchronous update messages.

For example:

```
{"id": 123, "method": "gcode/subscribe_output", "params": {"response_template": {"key": 345}}}
```

may initially respond with:

```
{"id": 123, "result": {}}
```

and cause Klipper to send future messages similar to:

```
{"params": {"response": "ok B:22.8 /0.0 T0:22.4 /0.0"}, "key": 345}
```

A subscription request accepts a "response_template" dictionary in the "params" field of the request. That "response_template" dictionary is used as a template for future asynchronous messages - it may contain arbitrary key/value pairs. When sending these future asynchronous messages, Klipper will add a "params" field containing a dictionary with "endpoint" specific contents to the response template and then send that template. If a "response_template" field is not provided then it defaults to an empty dictionary (`{}`).

Available "endpoints"

By convention, Klipper "endpoints" are of the form `<module_name>/<some_name>`. When making a request to an "endpoint", the full name must be set in the "method" parameter of the request dictionary (eg, `{"method"="gcode/restart"}`).

info

The "info" endpoint is used to obtain system and version information from Klipper. It is also used to provide the client's version information to Klipper. For example: `{"id": 123, "method": "info", "params": { "client_info": { "version": "v1"}}}`

If present, the "client_info" parameter must be a dictionary, but that dictionary may have arbitrary contents. Clients are encouraged to provide the name of the client and its software version when first connecting to the Klipper API server.

emergency_stop

The "emergency_stop" endpoint is used to instruct Klipper to transition to a "shutdown" state. It behaves similarly to the G-Code `M112` command. For example: `{"id": 123, "method": "emergency_stop"}`

register_remote_method

This endpoint allows clients to register methods that can be called from klipper. It will return an empty object upon success.

For example: `{"id": 123, "method": "register_remote_method", "params": {"response_template": {"action": "run_paneldue_beep"}, "remote_method": "paneldue_beep"}}` will return: `{"id": 123, "result": {}}`

The remote method `paneldue_beep` may now be called from Klipper. Note that if the method takes parameters they should be provided as keyword arguments. Below is an example of how it may called from a gcode_macro:

```
[gcode_macro PANELDUE_BEEP]
gcode:
  {action_call_remote_method("paneldue_beep", frequency=300,
duration=1.0)}
```

When the PANELDUE_BEEP gcode macro is executed, Klipper would send something like the following over the socket: `{"action": "run_paneldue_beep", "params": {"frequency": 300, "duration": 1.0}}`

objects/list

This endpoint queries the list of available printer "objects" that one may query (via the "objects/query" endpoint). For example: `{"id": 123, "method": "objects/list"}` might return: `{"id": 123, "result": {"objects": ["webhooks", "configfile", "heaters", "gcode_move", "query_endstops", "idle_timeout", "toolhead", "extruder"]}}`

objects/query

This endpoint allows one to query information from printer objects. For example: `{"id": 123, "method": "objects/query", "params": {"objects": {"toolhead": ["position"], "webhooks": null}}}` might return: `{"id": 123, "result": {"status": {"webhooks": {"state": "ready", "state_message": "Printer is ready"}, "toolhead": {"position": [0.0, 0.0, 0.0, 0.0]}}, "eventtime": 3051555.377933684}}`

The "objects" parameter in the request must be a dictionary containing the printer objects that are to be queried - the key contains the printer object name and the value is either "null" (to query all fields) or a list of field names.

The response message will contain a "status" field containing a dictionary with the queried information - the key contains the printer object name and the value is a dictionary containing its fields. The response message will also contain an "eventtime" field containing the timestamp from when the query was taken.

Available fields are documented in the Status Reference document.

objects/subscribe

This endpoint allows one to query and then subscribe to information from printer objects. The endpoint request and response is identical to the "objects/query" endpoint. For example: `{"id": 123, "method": "objects/subscribe", "params": {"objects":{"toolhead": ["position"], "webhooks": ["state"]}, "response_template":{}}}` might return: `{"id": 123, "result": {"status": {"webhooks": {"state": "ready"}, "toolhead": {"position": [0.0, 0.0, 0.0, 0.0]}}, "eventtime": 3052153.382083195}}` and result in subsequent asynchronous messages such as: `{"params": {"status": {"webhooks": {"state": "shutdown"}}, "eventtime": 3052165.418815847}}`

gcode/help

This endpoint allows one to query available G-Code commands that have a help string defined. For example: `{"id": 123, "method": "gcode/help"}` might return: `{"id": 123, "result": {"RESTORE_GCODE_STATE": "Restore a previously saved G-Code state", "PID_CALIBRATE": "Run PID calibration test", "QUERY_ADC": "Report the last value of an analog pin", ...}}`

gcode/script

This endpoint allows one to run a series of G-Code commands. For example: `{"id": 123, "method": "gcode/script", "params": {"script": "G90"}}`

If the provided G-Code script raises an error, then an error response is generated. However, if the G-Code command produces terminal output, that terminal output is not provided in the response. (Use the "gcode/subscribe_output" endpoint to obtain G-Code terminal output.)

If there is a G-Code command being processed when this request is received, then the provided script will be queued. This delay could be significant (eg, if a G-Code wait for temperature command is running). The JSON response message is sent when the processing of the script fully completes.

gcode/restart

This endpoint allows one to request a restart - it is similar to running the G-Code "RESTART" command. For example: `{"id": 123, "method": "gcode/restart"}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.

gcode/firmware_restart

This is similar to the "gcode/restart" endpoint - it implements the G-Code "FIRMWARE_RESTART" command. For example: `{"id": 123, "method": "gcode/firmware_restart"}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.

gcode/subscribe_output

This endpoint is used to subscribe to G-Code terminal messages that are generated by Klipper. For example: `{"id": 123, "method": "gcode/subscribe_output", "params": {"response_template":{}}}` might later produce asynchronous messages such as: `{"params": {"response": "// Klipper state: Shutdown"}}`

This endpoint is intended to support human interaction via a "terminal window" interface. Parsing content from the G-Code terminal output is discouraged. Use the "objects/subscribe" endpoint to obtain updates on Klipper's state.

motion_report/dump_stepper

This endpoint is used to subscribe to Klipper's internal stepper queue_step command stream for a stepper. Obtaining these low-level motion updates may be useful for diagnostic and debugging purposes. Using this endpoint may increase Klipper's system load.

A request may look like: `{"id": 123, "method":"motion_report/dump_stepper", "params": {"name": "stepper_x", "response_template": {}}}` and might return: `{"id": 123, "result": {"header": ["interval", "count", "add"]}}` and might later produce asynchronous messages such as: `{"params": {"first_clock": 179601081, "first_time": 8.98, "first_position": 0, "last_clock": 219686097, "last_time": 10.984, "data": [[179601081, 1, 0], [29573, 2, -8685], [16230, 4, -1525], [10559, 6, -160], [10000, 976, 0], [10000, 1000, 0], [10000, 1000, 0], [10000, 1000, 0], [9855, 5, 187], [11632, 4, 1534], [20756, 2, 9442]]}}`

The "header" field in the initial query response is used to describe the fields found in later "data" responses.

motion_report/dump_trapq

This endpoint is used to subscribe to Klipper's internal "trapezoid motion queue". Obtaining these low-level motion updates may be useful for diagnostic and debugging purposes. Using this endpoint may increase Klipper's system load.

A request may look like: `{"id": 123, "method": "motion_report/dump_trapq", "params": {"name": "toolhead", "response_template":{}}}` and might return: `{"id": 1, "result": {"header": ["time", "duration", "start_velocity", "acceleration", "start_position", "direction"]}}` and might later produce asynchronous messages such as: `{"params": {"data": [[4.05, 1.0, 0.0, 0.0, [300.0, 0.0, 0.0], [0.0, 0.0, 0.0]], [5.054, 0.001, 0.0, 3000.0, [300.0, 0.0, 0.0], [-1.0, 0.0, 0.0]]]}}`

The "header" field in the initial query response is used to describe the fields found in later "data" responses.

adxl345/dump_adxl345

This endpoint is used to subscribe to ADXL345 accelerometer data. Obtaining these low-level motion updates may be useful for diagnostic and debugging purposes. Using this endpoint may increase Klipper's system load.

A request may look like: `{"id": 123, "method":"adxl345/dump_adxl345", "params": {"sensor": "adxl345", "response_template": {}}}` and might return: `{"id": 123,"result":{"header":["time","x_acceleration","y_acceleration", "z_acceleration"]}}` and might later produce asynchronous messages such as: `{"params": {"overflows":0,"data":[[3292.432935,-535.44309,-1529.8374,9561.4], [3292.433256,-382.45935,-1606.32927,9561.48375]]}}`

The "header" field in the initial query response is used to describe the fields found in later "data" responses.

angle/dump_angle

This endpoint is used to subscribe to angle sensor data. Obtaining these low-level motion updates may be useful for diagnostic and debugging purposes. Using this endpoint may increase Klipper's system load.

A request may look like: `{"id": 123, "method":"angle/dump_angle", "params": {"sensor": "my_angle_sensor", "response_template": {}}}` and might return: `{"id": 123,"result": {"header":["time","angle"]}}` and might later produce asynchronous messages such as: `{"params":{"position_offset":3.151562,"errors":0, "data":[[1290.951905,-5063], [1290.952321,-5065]]}}`

The "header" field in the initial query response is used to describe the fields found in later "data" responses.

pause_resume/cancel

This endpoint is similar to running the "PRINT_CANCEL" G-Code command. For example: `{"id": 123, "method": "pause_resume/cancel"}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.

pause_resume/pause

This endpoint is similar to running the "PAUSE" G-Code command. For example: `{"id": 123, "method": "pause_resume/pause"}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.

pause_resume/resume

This endpoint is similar to running the "RESUME" G-Code command. For example: `{"id": 123, "method": "pause_resume/resume"}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.

query_endstops/status

This endpoint will query the active endpoints and return their status. For example: `{"id": 123, "method": "query_endstops/status"}` might return: `{"id": 123, "result": {"y": "open", "x": "open", "z": "TRIGGERED"}}`

As with the "gcode/script" endpoint, this endpoint only completes after any pending G-Code commands complete.