gpsd_json(5)

NAME

gpsd_json - gpsd request/response protocol

OVERVIEW

gpsd is a service daemon that can be used to monitor GPSes, DGPS receivers, Marine AIS broadcasts, and various other location-related and kinematic sensors.

Clients may communicate with **gpsd** via textual requests and responses over a socket. It is a bad idea for applications to speak the protocol directly: rather, they should use the **libgps** client library (for C; bindings also exist for other languages) and take appropriate care to check in their code for the expected major and minor protocol versions.

The GPSD protocol is built on top of JSON, JavaScript Object Notation, as specified in [RFC-7159]: The JavaScript Object Notation (JSON) Data Interchange Format. Similar to ECMA 404.

GPSD's use of JSON is restricted in some ways that make parsing it in fixed-extent languages (such as C) easier.

A request line is introduced by "?" and may include multiple commands. Commands begin with a command identifier, followed either by a terminating ';' or by an equal sign "=" and a JSON object treated as an argument. Any ';' or newline indication (either LF or CR-LF) after the end of a command is ignored. All request lines must be composed of US-ASCII characters and may be no more than 80 characters in length, exclusive of the trailing newline.

Responses are single JSON objects that have a "class" attribute the value of which is the object type . Object types include, but are not limited to: "TPV", "SKY", "DEVICE", and "ERROR". Objects are sent both in response to commands, and periodically as gpsd sends reports. Each object is terminated by a carriage return and a new line (CR-NL).

The order of JSON attributes within a response object is never significant, and you may specify command attributes in any order. Responses never contain the special JSON value null; instead, attributes with empty or undefined values are omitted. The length limit for responses and reports is currently 10240 characters, including the trailing CR-NL. Longer responses will be truncated, so client code must be prepared for the possibility of invalid JSON fragments.

The default maximum message length is set by GPS_JSON_RESPONSE_MAX in **include/gpsd_json.h**. at compile time.

In JSON reports, if an attribute is present only if the parent attribute is present or has a particular range, then the parent attribute is emitted first.

There is one constraint on the order in which attributes will be omitted. If an optional attribute is present only when a parent attribute has a specified value or range of values, the parent attribute

will be emitted first to make parsing easier.

The next subsection section documents the core GPSD protocol. Extensions are documented in the following subsections. The extensions may not be supported in your **gpsd** instance if it has been compiled with a restricted feature set.

The protocol was designed and documented by Eric S. Raymond.

CORE PROTOCOL RESPONSES

Here are the core-protocol responses.

TPV

A TPV object is a time-position-velocity report. The "class" and "mode" fields will reliably be present. When "mode" is 0 (Unknown) there is likely no usable data in the sentence. The remaining fields are optional, their presence depends on what data the GNSS receiver has sent, and what **gpsd** may calculate from that data.

A TPV object will usually be sent at least once for every measurement epoch as determined by the "time" field. Unless the receiver has a solid fix, and knows the current leap second, the time may be random.

Multiple TPV objects are often sent per epoch. When the receiver dribbles data to **gpsd**, then **gpsd** has no choice but to dribble it to the client in multiple TPV messages.

The optional "status" field (aka fix type), is a modifier (adjective) to mode. It is not a replacement for, or superset of, the "mode" field. It is almost, but not quite, the same as the NMEA 4.x xxGGA GPS Quality Indicator Values. Many GNSS receivers do not supply it. Those that do interpret the specification in various incompatible ways. To save space in the output, and avoid confusion, the JSON never includes status values of 0 or 1.

All error estimates (epc, epd, epe, eph, ept, epv, epx, epy) are guessed to be 95% confidence, may also be 50%, one sigma, or two sigma confidence. Many GNSS receivers do not specify a confidence level. None specify how the value is calculated. Use error estimates with caution, and only as relative "goodness" indicators. If the GPS reports a value to **gpsd**, then **gpsd** will report that value. Otherwise **gpsd** will try to compute the value from the skyview.

See the file **include/gps.h**, especially **struct gps_data_t**, for expanded notes on the items and values in the TPV message.

Table 1. TPV object

Name	Always?	Type	Description
class	Yes	string	Fixed: "TPV"
device	No	string	Name of the originating device.

mode	Yes	numeric	NMEA mode: 0=unknown, 1=no fix, 2=2D, 3=3D.
status	No	numeric	GPS fix status: 0=Unknown, 1=Normal, 2=DGPS, 3=RTK Fixed, 4=RTK Floating, 5=DR, 6=GNSSDR, 7=Time (surveyed), 8=Simulated, 9=P(Y)
time	No	string	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision. May be absent if the mode is not 2D or 3D. May be present, but invalid, if there is no fix. Verify 3 consecutive 3D fixes before believing it is UTC. Even then it may be off by several seconds until the current leap seconds is known.
altHAE	No	numeric	Altitude, height above ellipsoid, in meters. Probably WGS84.
altMSL	No	numeric	MSL Altitude in meters. The geoid used is rarely specified and is often inaccurate. See the comments below on geoidSep. altMSL is altHAE minus geoidSep.
alt	No	numeric	Deprecated. Undefined. Use altHAE or altMSL.

climb	No	numeric	Climb (positive) or sink (negative) rate, meters per second.
datum	No	string	Current datum. Hopefully WGS84.
depth	No	numeric	Depth in meters. Probably depth below the keel…
dgpsAge	No	numeric	Age of DGPS data. In seconds
dgpsSta	No	numeric	Station of DGPS data.
epc	No	numeric	Estimated climb error in meters per second. Certainty unknown.
epd	No	numeric	Estimated track (direction) error in degrees. Certainty unknown.
eph	No	numeric	Estimated horizontal Position (2D) Error in meters. Also known as Estimated Position Error (epe). Certainty unknown.
eps	No	numeric	Estimated speed error in meters per second. Certainty unknown.
ept	No	numeric	Estimated time stamp error in seconds. Certainty unknown.
epx	No	numeric	Longitude error estimate in meters. Certainty unknown.
epy	No	numeric	Latitude error estimate in meters. Certainty unknown.
epv	No	numeric	Estimated vertical error in meters. Certainty unknown.

geoidSep	No	numeric	Geoid separation is the difference between the WGS84 reference ellipsoid and the geoid (Mean Sea Level) in meters. Almost no GNSS receiver specifies how they compute their geoid. **gpsd** interpolates the geoid from a 5x5 degree table of EGM2008 values when the receiver does not supply a geoid separation. The **gpsd** computed geoidSep is usually within one meter of the "true" value, but can be off as much as 12 meters.
lat	No	numeric	Latitude in degrees: +/- signifies North/South.
leapseconds	No	integer	Current leap seconds.
lon	No	numeric	Longitude in degrees: +/- signifies East/West.
track	No	numeric	Course over ground, degrees from true north.
magtrack	No	numeric	Course over ground, degrees magnetic.
magvar	No	numeric	Magnetic variation, degrees. Also known as the magnetic declination (the direction of the horizontal component of the magnetic field measured clockwise from north) in degrees, Positive is West variation. Negative is East variation.
speed	No	numeric	Speed over ground, meters per second.

ecefx	No	numeric	ECEF X position in meters.
ecefy	No	numeric	ECEF Y position in meters.
ecefz	No	numeric	ECEF Z position in meters.
ecefpAcc	No	numeric	ECEF position error in meters. Certainty unknown.
ecefvx	No	numeric	ECEF X velocity in meters per second.
ecefvy	No	numeric	ECEF Y velocity in meters per second.
ecefvz	No	numeric	ECEF Z velocity in meters per second.
ecefvAcc	No	numeric	ECEF velocity error in meters per second. Certainty unknown.
sep	No	numeric	Estimated Spherical (3D) Position Error in meters. Guessed to be 95% confidence, but many GNSS receivers do not specify, so certainty unknown.
relD	No	numeric	Down component of relative position vector in meters.
relE	No	numeric	East component of relative position vector in meters.
relN	No	numeric	North component of relative position vector in meters.
velD	No	numeric	Down velocity component in meters.
velE	No	numeric	East velocity component in meters.
velN	No	numeric	North velocity component in meters.

wanglem	No	numeric	Wind angle magnetic in degrees.
wangler	No	numeric	Wind angle relative in degrees.
wanglet	No	numeric	Wind angle true in degrees.
wspeedr	No	numeric	Wind speed relative in meters per second.
wspeedt	No	numeric	Wind speed true in meters per second.
wtemp	No	numeric	Water temperature in degrees Celsius.

When the C client library parses a response of this kind, it will assert validity bits in the top-level set member for each field received; see gps.h for bitmask names and values.

Invalid or unknown floating-point values will be set to NAN. Always check floating point values with isfinite() before use. isnan() is not sufficient.

Here's an example TPV sentence:

```
{"class":"TPV","device":"/dev/pts/1",
    "time":"2005-06-08T10:34:48.283Z","ept":0.005,
    "lat":46.498293369,"lon":7.567411672,"alt":1343.127,
    "eph":36.000,"epv":32.321,
    "track":10.3788,"speed":0.091,"climb":-0.085,"mode":3}
```

SKY

A SKY object reports a sky view of the GPS satellite positions. If there is no GPS device available, or no skyview has been reported yet, only the "class" field will reliably be present.

Table 2. SKY object

Name	Always?	Type	Description
class	Yes	string	Fixed: "SKY"
device	No	string	Name of originating device
nSat	No	numeric	Number of satellite objects in "satellites" array.

Name	Always?	Type	Description
gdop	No	numeric	Geometric (hyperspherical) dilution of precision, a combination of PDOP and TDOP. A dimensionless factor which should be multiplied by a base UERE to get an error estimate.
hdop	No	numeric	Horizontal dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get a circular error estimate.
pdop	No	numeric	Position (spherical/3D) dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get an error estimate.
prRes	No	numeric	Pseudorange residue in meters.
qual	No	numeric	Quality Indicator 0=no signal 1=searching signal 2=signal acquired 3=signal detected but unusable 4=code locked and time synchronized 5, 6, 7=code and carrier locked and time synchronized
satellites	No	list	List of satellite objects in skyview

Name	Always?	Type	Description
tdop	No	numeric	Time dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get an error estimate.
time	No	string	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision.
uSat	No	numeric	Number of satellites used in navigation solution.
vdop	No	numeric	Vertical (altitude) dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get an error estimate.
xdop	No	numeric	Longitudinal dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get an error estimate.
ydop	No	numeric	Latitudinal dilution of precision, a dimensionless factor which should be multiplied by a base UERE to get an error estimate.

Many devices compute dilution of precision factors but do not include them in their reports. Many that do report DOPs report only HDOP, two-dimensional circular error. **gpsd** always passes through whatever the device reports, then attempts to fill in other DOPs by calculating the appropriate determinants in a covariance matrix based on the satellite view. DOPs may be missing if some of these determinants are singular. It can even happen that the device reports an error estimate in meters when the corresponding DOP is unavailable; some devices use more sophisticated error modeling than the covariance calculation.

The satellite list objects have the following elements:

Table 3. Satellite object

Name	Always?	Type	Description
PRN	Yes	numeric	PRN ID of the satellite. 1-63 are GNSS satellites, 64-96 are GLONASS satellites, 100-164 are SBAS satellites
az	No	numeric	Azimuth, degrees from true north.
el	No	numeric	Elevation in degrees.
ss	No	numeric	Signal to Noise ratio in dBHz.
used	Yes	boolean	Used in current solution? (SBAS/WAAS/EGNOS satellites may be flagged used if the solution has corrections from them, but not all drivers make this information available.)
gnssid	No	numeric	The GNSS ID, as defined by u-blox, not NMEA. 0=GPS, 2=Galileo, 3=Beidou, 5=QZSS, 6-GLONASS.
svid	No	numeric	The satellite ID within its constellation. As defined by u-blox, not NMEA).
sigid	No	numeric	The signal ID of this signal. As defined by u-blox, not NMEA. See u-blox doc for details.
freqid	No	numeric	For GLONASS satellites only: the frequency ID of the signal. As defined by u-blox, range 0 to 13. The freqid is the frequency slot plus 7.

Name	Always?	Type	Description
health	No	numeric	The health of this satellite. 0 is unknown, 1 is OK, and 2 is unhealthy.

Note that satellite objects do not have a "class" field, as they are never shipped outside of a SKY object.

When the C client library parses a SKY response, it will assert the SATELLITE_SET bit in the top-level set member.

Here's an example:

```
{"class":"SKY","device":"/dev/pts/1",
    "time":"2005-07-08T11:28:07.114Z",
    "xdop":1.55,"hdop":1.24,"pdop":1.99,
    "satellites":[
        {"PRN":23,"el":6,"az":84,"ss":0,"used":false},
        {"PRN":28,"el":7,"az":160,"ss":0,"used":false},
        {"PRN":8,"el":66,"az":189,"ss":44,"used":true},
        {"PRN":29,"el":13,"az":273,"ss":0,"used":false},
        {"PRN":10,"el":51,"az":304,"ss":29,"used":true},
        {"PRN":4,"el":15,"az":199,"ss":36,"used":true},
        {"PRN":2,"el":34,"az":241,"ss":43,"used":true},
        {"PRN":27,"el":71,"az":76,"ss":43,"used":true}]}
```

GST

A GST object is a pseudorange noise report.

Table 4. GST object

Name	Always?	Type	Description
class	Yes	string	Fixed: "GST"
device	No	string	Name of originating device
time	No	string	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision.

Name	Always?	Type	Description
rms	No	numeric	Value of the standard deviation of the range inputs to the navigation process (range inputs include pseudoranges and DGPS corrections).
major	No	numeric	Standard deviation of semi-major axis of error ellipse, in meters.
minor	No	numeric	Standard deviation of semi-minor axis of error ellipse, in meters.
orient	No	numeric	Orientation of semi-major axis of error ellipse, in degrees from true north.
lat	No	numeric	Standard deviation of latitude error, in meters.
lon	No	numeric	Standard deviation of longitude error, in meters.
alt	No	numeric	Standard deviation of altitude error, in meters.

Here's an example:

```
{"class":"GST","device":"/dev/ttyUSB0",
        "time":"2010-12-07T10:23:07.096Z","rms":2.440,
        "major":1.660,"minor":1.120,"orient":68.989,
        "lat":1.600,"lon":1.200,"alt":2.520}
```

ATT

An ATT object is a vehicle-attitude report. It is returned by digital-compass and gyroscope sensors; depending on device, it may include: heading, pitch, roll, yaw, gyroscope, and magnetic-field readings. Because such sensors are often bundled as part of marine-navigation systems, the ATT response may also include water depth.

The "class" and "mode" fields will reliably be present. Others may be reported or not depending on the specific device type.

The ATT object is synchronous to the GNSS epoch. Some devices report attitude information with arbitrary, even out of order, time scales. **gpsd** reports those in an IMU object. The ATT and IMU objects have the same fields, but IMU objects are output as soon as possible. Some devices output both types with arbitrary interleaving.

Table 5. ATT object

Name	Always?	Type	Description
class	Yes	string	Fixed: "ATT"
device	Yes	string	Name of originating device
time	No	string	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision.
timeTag	No	string	Arbitrary time tag of measurement.
heading	No	numeric	Heading, degrees from true north.
mag_st	No	string	Magnetometer status.
mheading	No	numeric	Heading, degrees from magnetic north.
pitch	No	numeric	Pitch in degrees.
pitch_st	No	string	Pitch sensor status.
rot	No	numeric	Rate of Turn in dgrees per minute.
yaw	No	numeric	Yaw in degrees
yaw_st	No	string	Yaw sensor status.
roll	No	numeric	Roll in degrees.
roll_st	No	string	Roll sensor status.
dip	No	numeric	Local magnetic inclination, degrees, positive when the magnetic field points downward (into the Earth).
mag_len	No	numeric	Scalar magnetic field strength.
mag_x	No	numeric	X component of magnetic field strength.

Name	Always?	Type	Description
mag_y	No	numeric	Y component of magnetic field strength.
mag_z	No	numeric	Z component of magnetic field strength.
acc_len	No	numeric	Scalar acceleration.
acc_x	No	numeric	X component of acceleration (m/s^2).
acc_y	No	numeric	Y component of acceleration (m/s^2).
acc_z	No	numeric	Z component of acceleration (m/s^2).
gyro_x	No	numeric	X component of angular rate (deg/s)
gyro_y	No	numeric	Y component of angular rate (deg/s)
gyro_z	No	numeric	Z component of angular rate (deg/s)
depth	No	numeric	Water depth in meters.
temp	No	numeric	Temperature at the sensor, degrees centigrade.

The heading, pitch, and roll status codes (if present) vary by device. For the TNT Revolution digital compasses, they are coded as follows:

Table 6. Device flags

Code	Description
C	magnetometer calibration alarm
L	low alarm
M	low warning
N	normal
O	high warning
P	high alarm
V	magnetometer voltage level alarm

When the C client library parses a response of this kind, it will assert ATT_IS.

Here’s an example:

```
{"class":"ATT","time":1270938096.843,
    "heading":14223.00,"mag_st":"N",
    "pitch":169.00,"pitch_st":"N", "roll":-43.00,"roll_st":"N",
    "dip":13641.000,"mag_x":2454.000}
```

IMU

The IMU object is asynchronous to the GNSS epoch. It is reported with arbitrary, even out of order, time scales.

The ATT and IMU objects have the same fields, but IMU objects are output as soon as possible.

Seee the ATT onject description for field details.

TOFF

This message is emitted on each cycle and reports the offset between the host's clock time and the GPS time at top of the second (actually, when the first data for the reporting cycle is received).

This message exactly mirrors the PPS message.

The TOFF message reports the GPS time as derived from the GPS serial data stream. The PPS message reports the GPS time as derived from the GPS PPS pulse.

A TOFF object has the following elements:

Table 7. TOFF object

Name	Always?	Type	Description
class	Yes	string	Fixed: "TOFF"
device	Yes	string	Name of the originating device
real_sec	Yes	numeric	seconds from the GPS clock
real_nsec	Yes	numeric	nanoseconds from the GPS clock
clock_sec	Yes	numeric	seconds from the system clock
clock_nsec	Yes	numeric	nanoseconds from the system clock

This message is emitted once per second to watchers of a device and is intended to report the timestamps of the in-band report of the GPS and seconds as reported by the system clock (which may be NTP-corrected) when the first valid time stamp of the reporting cycle was seen.

The message contains two second/nanosecond pairs: real_sec and real_nsec contain the time the

GPS thinks it was at the start of the current cycle; clock_sec and clock_nsec contain the time the system clock thinks it was on receipt of the first timing message of the cycle. real_nsec is always to nanosecond precision. clock_nsec is nanosecond precision on most systems.

Here's an example:

```
{"class":"TOFF","device":"/dev/ttyUSB0",
     "real_sec":1330212592, "real_nsec":343182,
     "clock_sec":1330212592,"clock_nsec":343184,
     "precision":-2}
```

PPS

This message is emitted each time the daemon sees a valid PPS (Pulse Per Second) strobe from a device.

This message exactly mirrors the TOFF message.

The TOFF message reports the GPS time as derived from the GPS serial data stream. The PPS message reports the GPS time as derived from the GPS PPS pulse.

There are various sources of error in the reported clock times. The speed of the serial connection between the GPS and the system adds a delay to the start of cycle detection. An even bigger error is added by the variable computation time inside the GPS. Taken together the time derived from the start of the GPS cycle can have offsets of 10 milliseconds to 700 milliseconds and combined jitter and wander of 100 to 300 milliseconds.

See the NTP documentation for their definition of precision.

A PPS object has the following elements:

Table 8. PPS object

Name	Always?	Type	Description
class	Yes	string	Fixed: "PPS"
device	Yes	string	Name of the originating device
real_sec	Yes	numeric	seconds from the PPS source
real_nsec	Yes	numeric	nanoseconds from the PPS source
clock_sec	Yes	numeric	seconds from the system clock
clock_nsec	Yes	numeric	nanoseconds from the system clock

Name	Always?	Type	Description
precision	Yes	numeric	NTP style estimate of PPS precision
shm	Yes	string	shm key of this PPS
qErr	No	numeric	Quantization error of the PPS, in picoseconds. Sometimes called the "sawtooth" error.

This message is emitted once per second to watchers of a device emitting PPS, and reports the time of the start of the GPS second (when the 1PPS arrives) and seconds as reported by the system clock (which may be NTP-corrected) at that moment.

The message contains two second/nanosecond pairs: real_sec and real_nsec contain the time the GPS thinks it was at the PPS edge; clock_sec and clock_nsec contain the time the system clock thinks it was at the PPS edge. real_nsec is always to nanosecond precision. clock_nsec is nanosecond precision on most systems.

There are various sources of error in the reported clock times. For PPS delivered via a real serial-line strobe, serial-interrupt latency plus processing time to the timer call should be bounded above by about 10 microseconds; that can be reduced to less than 1 microsecond if your kernel supports [RFC-2783]. USB1.1-to-serial control-line emulation is limited to about 1 millisecond. seconds.

Here's an example:

```
{"class":"PPS","device":"/dev/ttyUSB0",
     "real_sec":1330212592, "real_nsec":343182,
     "clock_sec":1330212592,"clock_nsec":343184,
     "precision":-3}
```

OSC

This message reports the status of a GPS-disciplined oscillator (GPSDO). The GPS PPS output (which has excellent long-term stability) is typically used to discipline a local oscillator with much better short-term stability (such as a rubidium atomic clock).

An OSC object has the following elements:

Table 9. OSC object

Name	Always?	Type	Description
class	Yes	string	Fixed: "OSC"
device	Yes	string	Name of the originating device.

Name	Always?	Type	Description
running	Yes	boolean	If true, the oscillator is currently running. Oscillators may require warm-up time at the start of the day.
reference	Yes	boolean	If true, the oscillator is receiving a GPS PPS signal.
disciplined	Yes	boolean	If true, the GPS PPS signal is sufficiently stable and is being used to discipline the local oscillator.
delta	Yes	numeric	The time difference (in nanoseconds) between the GPS-disciplined oscillator PPS output pulse and the most recent GPS PPS input pulse.

Here's an example:

```
{"class":"OSC","running":true,"device":"/dev/ttyUSB0",
    "reference":true,"disciplined":true,"delta":67}
```

CORE PROTOCOL COMMANDS

And here are the commands you can send to **gpsd**.

?VERSION;

Returns an object with the following attributes:

Table 10. VERSION object

Name	Always?	Type	Description
class	Yes	string	Fixed: "VERSION"
release	Yes	string	Public release level
rev	Yes	string	Internal revision-control level.

Name	Always?	Type	Description
proto_major	Yes	numeric	API major revision level.
proto_minor	Yes	numeric	API minor revision level.
remote	No	string	URL of the remote daemon reporting this version. If empty, this is the version of the local daemon.

The daemon ships a VERSION response to each client when the client first connects to it.

When the C client library parses a response of this kind, it will assert the VERSION_SET bit in the top-level set member.

Here's an example:

```
{"class":"VERSION","version":"2.40dev",
    "rev":"06f62e14eae9886cde907dae61c124c53eb1101f",
    "proto_major":3,"proto_minor":1
}
```

?DEVICES;

Returns a device list object with the following elements:

Table 11. DEVICES object

Name	Always?	Type	Description
class	Yes	string	Fixed: "DEVICES"
devices	Yes	list	List of device descriptions
remote	No	string	URL of the remote daemon reporting the device set. If empty, this is a DEVICES response from the local daemon.

When the C client library parses a response of this kind, it will assert the DEVICELIST_SET bit in the top-level set member.

Here's an example:

```
{"class"="DEVICES","devices":[
```

```
    {"class":"DEVICE","path":"/dev/pts/1","flags":1,"driver":"SiRF binary"},
    {"class":"DEVICE","path":"/dev/pts/3","flags":4,"driver":"AIVDM"}]}
```

The daemon occasionally ships a bare DEVICE object to the client (that is, one not inside a DEVICES wrapper). The data content of these objects will be described later as a response to the ?DEVICE command.

?WATCH;

This command sets watcher mode. It also sets or elicits a report of per-subscriber policy and the raw bit. An argument WATCH object changes the subscriber's policy. The response describes the subscriber's policy. The response will also include a DEVICES object.

A WATCH object has the following elements:

Table 12. WATCH object

Name	Always?	Type	Description
class	Yes	string	Fixed: "WATCH"
enable	No	boolean	Enable (true) or disable (false) watcher mode. Default is true.
json	No	boolean	Enable (true) or disable (false) dumping of JSON reports. Default is false.
nmea	No	boolean	Enable (true) or disable (false) dumping of binary packets as pseudo-NMEA. Default is false.

Name	Always?	Type	Description
raw	No	integer	Controls 'raw' mode. When this attribute is set to 1 for a channel, **gpsd** reports the unprocessed NMEA or AIVDM data stream from whatever device is attached. Binary GPS packets are hex-dumped. RTCM2 and RTCM3 packets are not dumped in raw mode. When this attribute is set to 2 for a channel that processes binary data, **gpsd** reports the received data verbatim without hex-dumping.
scaled	No	boolean	If true, apply scaling divisors to output before dumping; default is false.
split24	No	boolean	If true, aggregate AIS type24 sentence parts. If false, report each part as a separate JSON object, leaving the client to match MMSIs and aggregate. Default is false. Applies only to AIS reports.
pps	No	boolean	If true, emit the TOFF JSON message on each cycle and a PPS JSON message when the device issues 1PPS. Default is false.

Name	Always?	Type	Description
device	No	string	If present, enable watching only of the specified device rather than all devices. Useful with raw and NMEA modes in which device responses aren't tagged. Has no effect when used with enable:false.
remote	No	string	URL of the remote daemon reporting the watch set. If empty, this is a WATCH response from the local daemon.

There is an additional boolean "timing" attribute which is undocumented because that portion of the interface is considered unstable and for developer use only.

In watcher mode, GPS reports are dumped as TPV and SKY responses. AIS, Subframe and RTCM reporting is described in the next section.

When the C client library parses a response of this kind, it will assert the POLICY_SET bit in the top-level set member.

Here's an example:

```
{"class":"WATCH", "raw":1,"scaled":true}
```

?POLL;

The POLL command requests data from the last-seen fixes on all active GPS devices. Devices must previously have been activated by ?WATCH to be pollable.

Polling can lead to possibly surprising results when it is used on a device such as an NMEA GPS for which a complete fix has to be accumulated from several sentences. If you poll while those sentences are being emitted, the response will contain only the fix data collected so far in the current epoch. It may be as much as one cycle time (typically 1 second) stale.

The POLL response will contain a timestamped list of TPV objects describing cached data, and a timestamped list of SKY objects describing satellite configuration. If a device has not seen fixes, it will be reported with a mode field of zero.

Table 13. POLL object

Name	Always?	Type	Description
class	Yes	string	Fixed: "POLL"
time	Yes	Numeric	Timestamp in ISO 8601 format. May have a fractional part of up to .001sec precision.
active	Yes	Numeric	Count of active devices.
tpv	Yes	JSON array	Comma-separated list of TPV objects.
sky	Yes	JSON array	Comma-separated list of SKY objects.

Here's an example of a POLL response:

```
{"class":"POLL","time":"2010-06-04T10:31:00.289Z","active":1,
    "tpv":[{"class":"TPV","device":"/dev/ttyUSB0",
            "time":"2010-09-08T13:33:06.095Z",
            "ept":0.005,"lat":40.035093060,
            "lon":-75.519748733,"track":99.4319,"speed":0.123,"mode":2}],
    "sky":[{"class":"SKY","device":"/dev/ttyUSB0",
            "time":1270517264.240,"hdop":9.20,
            "satellites":[{"PRN":16,"el":55,"az":42,"ss":36,"used":true},
                          {"PRN":19,"el":25,"az":177,"ss":0,"used":false},
                          {"PRN":7,"el":13,"az":295,"ss":0,"used":false},
                          {"PRN":6,"el":56,"az":135,"ss":32,"used":true},
                          {"PRN":13,"el":47,"az":304,"ss":0,"used":false},
                          {"PRN":23,"el":66,"az":259,"ss":0,"used":false},
                          {"PRN":20,"el":7,"az":226,"ss":0,"used":false},
                          {"PRN":3,"el":52,"az":163,"ss":32,"used":true},
                          {"PRN":31,"el":16,"az":102,"ss":0,"used":false}
]}]}
```

NOTE Client software should not assume the field inventory of the POLL response is fixed for all time. As **gpsd** collects and caches more data from more sensor types, those data are likely to find their way into this response.

?DEVICE; ?DEVICE=

This command reports (when followed by ';') the state of a device, or sets (when followed by '=' and a DEVICE object) device-specific control bits, notably the device's speed and serial mode and the native-mode bit. The parameter-setting form will be rejected if more than one client is attached to the channel and a device path is not specified.

Pay attention to the response, because it is possible for this command to fail if the GPS does not support a command or only supports some combinations of modes. In case of failure, the daemon

and GPS will continue to communicate at the old speed.

Use the parameter-setting form with caution. On USB and Bluetooth GPSes it is also possible for serial mode setting to fail either because the serial adaptor chip does not support non-8N1 modes or because the device firmware does not properly synchronize the serial adaptor chip with the UART on the GPS chipset when the speed changes. These failures can hang your device, possibly requiring a GPS power cycle or (in extreme cases) physically disconnecting the NVRAM backup battery.

A DEVICE object has the following elements:

Table 14. DEVICE object

Name	Always?	Type	Description
class	Yes	string	Fixed: "DEVICE"
activated	No	string	Time the device was activated as an ISO8601 time stamp. If the device is inactive this attribute is absent.
bps	No	integer	Device speed in bits per second.
cycle	No	real	Device cycle time in seconds.
driver	No	string	GPSD's name for the device driver type. Won't be reported before **gpsd** has seen identifiable packets from the device.
flags	No	integer	Bit vector of property flags. Currently defined flags are: describe packet types seen so far (GPS, RTCM2, RTCM3, AIS). Won't be reported if empty, e.g. before **gpsd** has seen identifiable packets from the device.
hexdata	No	string	Data, in bare hexadecimal, to send to the GNSS receiver.

Name	Always?	Type	Description
mincycle	No	real	Device minimum cycle time in seconds. Reported from ?DEVICE when (and only when) the rate is switchable. It is read-only and not settable.
native	No	integer	0 means NMEA mode and 1 means alternate mode (binary if it has one, for SiRF and Evermore chipsets in particular). Attempting to set this mode on a non-GPS device will yield an error.
parity	No	string	N, O or E for no parity, odd, or even.
path	No	string	Name the device for which the control bits are being reported, or for which they are to be applied. This attribute may be omitted only when there is exactly one subscribed channel.
readonly	No	boolean	True if device is read-only.
stopbits	Yes	integer	Stop bits (1 or 2).
subtype	No	string	Whatever version information the device driver returned.
subtype1	No	string	More version information the device driver returned.

The serial parameters will (bps, parity, stopbits) be omitted in a response describing a TCP/IP source such as an Ntrip, DGPSIP, or AIS feed; on a serial device they will always be present.

The contents of the flags field should be interpreted as follows:

Table 15. Device flags

C #define	Value	Description
SEEN_GPS	0x01	GPS data has been seen on this device
SEEN_RTCM2	0x02	RTCM2 data has been seen on this device
SEEN_RTCM3	0x04	RTCM3 data has been seen on this device
SEEN_AIS	0x08	AIS data has been seen on this device

When the C client library parses a response of this kind, it will assert the DEVICE_SET bit in the top-level set member.

Here's an example:

```
{"class":"DEVICE","bps":4800,"parity":"N","stopbits":1,"native":0}
```

When a client is in watcher mode, the daemon will ship it DEVICE notifications when a device is added to the pool or deactivated.

When the C client library parses a response of this kind, it will assert the DEVICE_SET bit in the top-level set member.

Examples

A notice of a deactivated device:

```
{"class":"DEVICE","path":"/dev/pts1","activated":0}
```

A send a u-blox receiver at /dev/ttyUSB2 a request for a UBX-MON-VER message:

```
?DEVICE={"path":"/dev/ttyUSB2","hexdata":"b5620a0400000e34"}
```

The *gpsd* daemon will respond with an ACK on success:

```
{"class":"ACK"}
```

ERROR

The daemon may ship an error object in response to a syntactically invalid command line or unknown command. It has the following elements:

Table 16. ERROR notification object

Name	Always?	Type	Description
class	Yes	string	Fixed: "ERROR"
message	Yes	string	Textual error message

Here's an example:

```
{"class":"ERROR","message":"Unrecognized request '?FOO'"}
```

When the C client library parses a response of this kind, it will assert the ERR_SET bit in the top-level set member.

RTCM2

RTCM-104 is a family of serial protocols used for broadcasting pseudorange corrections from differential-GPS reference stations. Many GPS receivers can accept these corrections to improve their reporting accuracy.

RTCM-104 comes in two major and incompatible flavors, 2.x and 3.x. Each major flavor has minor (compatible) revisions.

The applicable standard for RTCM Version 2.x is RTCM Recommended Standards for Differential NAVSTAR GPS Service RTCM Paper 194-93/SC 104-STD. For RTCM 3.1 it is RTCM Paper 177-2006-SC104-STD. Ordering instructions for both standards are accessible from the website of the Radio Technical Commission for Maritime Services under "Publications".

RTCM WIRE TRANSMISSIONS

Differential-GPS correction stations consist of a GPS reference receiver coupled to a low frequency (LF) transmitter. The GPS reference receiver is a survey-grade GPS that does GPS carrier tracking and can work out its position to a few millimeters. It generates range and range-rate corrections and encodes them into RTCM104. It ships the RTCM104 to the LF transmitter over serial rs-232 signal at 100 baud or 200 baud depending on the requirements of the transmitter.

The LF transmitter broadcasts the approximately 300khz radio signal that differential-GPS radio receivers pick up. Transmitters that are meant to have a higher range will need to transmit at a slower rate. The higher the data rate the harder it will be for the remote radio receiver to receive with a good signal-to-noise ration. (Higher data rate signals can't be averaged over as long a time frame, hence they appear noisier.)

RTCM WIRE FORMATS

An RTCM 2.x message consists of a sequence of up to 33 30-bit words. The 24 most significant bits of each word are data and the six least significant bits are parity. The parity algorithm used is the same ISGPS-2000 as that used on GPS satellite downlinks. Each RTCM 2.x message consists of two header words followed by zero or more data words, depending upon the message type.

An RTCM 3.x message begins with a fixed leader byte 0xD3. That is followed by six bits of version information and 10 bits of payload length information. Following that is the payload; following the payload is a 3-byte checksum of the payload using the Qualcomm CRC-24Q algorithm.

RTCM2 JSON FORMAT

Each RTCM2 message is dumped as a single JSON object per message, with the message fields as attributes of that object. Arrays of satellite, station, and constellation statistics become arrays of JSON sub-objects. Each sentence will normally also have a "device" field containing the pathname of the originating device.

All attributes other than the device field are mandatory. Header attributes are emitted before others.

Header portion

Table 17. SKY object

Name	Type	Description
class	string	Fixed: "RTCM2".
type	integer	Message type (1-9).
station_id	integer	The id of the GPS reference receiver. The LF transmitters also have (different) id numbers.
zcount	real	The reference time of the corrections in the message in seconds within the current hour. Note that it is in GPS time, which is some seconds ahead of UTC (see the U.S. Naval Observatory's table of leap second corrections).
seqnum	integer	Sequence number. Only 3 bits wide, wraps after 7.
length	integer	The number of words after the header that comprise the message.

Name	Type	Description
station_health	integer	Station transmission status. Indicates the health of the beacon as a reference source. Any nonzero value means the satellite is probably transmitting bad data and should not be used in a fix. 6 means the transmission is unmonitored. 7 means the station is not working properly. Other values are defined by the beacon operator.

<message type> is one of

1

full corrections — one message containing corrections for all GPS satellites in view. This is not common.

3

reference station parameters — the position of the reference station GPS antenna.

4

datum — the datum to which the DGPS data is referred.

5

constellation health — information about the satellites the beacon can see.

6

null message — just a filler.

7

radio beacon almanac — information about this or other beacons.

9

subset corrections — a message containing corrections for only a subset of the GPS satellites in view.

16

special message — a text message from the beacon operator.

31

GLONASS subset corrections — a message containing corrections for a set of the GLONASS satellites in view.

Type 1 and 9: Correction data

One or more satellite objects follow the header for type 1 or type 9 messages. Here is the format:

Table 18. Satellite object

Name	Type	Description
ident	integer	The PRN number of the satellite for which this is correction data.
udre	integer	User Differential Range Error (0-3). See the table following for values.
iod	integer	Issue Of Data, matching the IOD for the current ephemeris of this satellite, as transmitted by the satellite. The IOD is a unique tag that identifies the ephemeris; the GPS using the DGPS correction and the DGPS generating the data must use the same orbital positions for the satellite.
prc	real	The pseudorange error in meters for this satellite as measured by the beacon reference receiver at the epoch indicated by the z_count in the parent record.
rrc	real	The rate of change of pseudorange error in meters/sec for this satellite as measured by the beacon reference receiver at the epoch indicated by the z_count field in the parent record. This is used to calculate pseudorange errors at other epochs, if required by the GPS receiver.

User Differential Range Error values are as follows:

Table 19. UDRE values

0	1-sigma error ⇐ 1 m
1	1-sigma error ⇐ 4 m
2	1-sigma error ⇐ 8 m

3	1-sigma error > 8 m

Here's an example:

```
{"class":"RTCM2","type":1,
    "station_id":688,"zcount":843.0,"seqnum":5,"length":19,"station_health":6,
    "satellites":[
        {"ident":10,"udre":0,"iod":46,"prc":-2.400,"rrc":0.000},
        {"ident":13,"udre":0,"iod":94,"prc":-4.420,"rrc":0.000},
        {"ident":7,"udre":0,"iod":22,"prc":-5.160,"rrc":0.002},
        {"ident":2,"udre":0,"iod":34,"prc":-6.480,"rrc":0.000},
        {"ident":4,"udre":0,"iod":47,"prc":-8.860,"rrc":0.000},
        {"ident":8,"udre":0,"iod":76,"prc":-7.980,"rrc":0.002},
        {"ident":5,"udre":0,"iod":99,"prc":-8.260,"rrc":0.002},
        {"ident":23,"udre":0,"iod":81,"prc":-8.060,"rrc":0.000},
        {"ident":16,"udre":0,"iod":70,"prc":-11.740,"rrc":0.000},
        {"ident":30,"udre":0,"iod":4,"prc":-18.960,"rrc":-0.006},
        {"ident":29,"udre":0,"iod":101,"prc":-24.960,"rrc":-0.002}
]}
```

Type 3: Reference Station Parameters

Here are the payload members of a type 3 (Reference Station Parameters) message:

Table 20. Reference Station Parameters

Name	Type	Description
x	real	ECEF X coordinate.
y	real	ECEF Y coordinate.
z	real	ECEF Z coordinate.

The coordinates are the position of the station, in meters to two decimal places, in Earth Centred Earth Fixed coordinates. These are usually referred to the WGS84 reference frame, but may be referred to NAD83 in the US (essentially identical to WGS84 for all except geodesists), or some other reference frame in other parts of the world.

An invalid reference message is represented by a type 3 header without payload fields.

Here's an example:

```
{"class":"RTCM2","type":3,
    "station_id":652,"zcount":1657.2,"seqnum":2,"length":4,"station_health":6,
    "x":3878620.92,"y":670281.40,"z":5002093.59
}
```

Type 4: Datum

Here are the payload members of a type 4 (Datum) message:

Table 21. Datum

Name	Type	Description
dgnss_type	string	Either "GPS", "GLONASS", "GALILEO", or "UNKNOWN".
dat	integer	0 or 1 and indicates the sense of the offset shift given by dx, dy, dz. dat = 0 means that the station coordinates (in the reference message) are referred to a local datum and that adding dx, dy, dz to that position will render it in GNSS coordinates (WGS84 for GPS). If dat = 1 then the ref station position is in GNSS coordinates and adding dx, dy, dz will give it referred to the local datum.
datum_name	string	A standard name for the datum.
dx	real	X offset.
dy	real	Y offset.
dz	real	Z offset.

<dx> <dy> <dz> are offsets to convert from local datum to GNSS datum or vice versa. These fields are optional.

An invalid datum message is represented by a type 4 header without payload fields.

Type 5: Constellation Health

One or more of these follow the header for type 5 messages — one for each satellite.

Here is the format:

Table 22. Constellation health

Name	Type	Description
ident	integer	The PRN number of the satellite.

Name	Type	Description
iodl	bool	True indicates that this information relates to the satellite information in an accompanying type 1 or type 9 message.
health	integer	0 indicates that the satellite is healthy. Any other value indicates a problem (coding is not known).
snr	integer	The carrier/noise ratio of the received signal in the range 25 to 55 dB(Hz).
health_en	bool	If set to True it indicates that the satellite is healthy even if the satellite navigation data says it is unhealthy.
new_data	bool	True indicates that the IOD for this satellite will soon be updated in type 1 or 9 messages.
los_warning	bool	Line-of-sight warning. True indicates that the satellite will shortly go unhealthy.
tou	integer	Healthy time remaining in seconds.

Type 6: Null

This just indicates a null message. There are no payload fields.

Unknown message

This format is used to dump message words in hexadecimal when the message type field doesn't match any of the known ones.

Here is the format:

Table 23. Unknown Message

Name	Type	Description
data	list	A list of strings.

Each string in the array is a hex literal representing 30 bits of information, after parity checks and inversion. The high two bits should be ignored.

Type 7: Radio Beacon Almanac

Here is the format:

Table 24. Constellation health

Name	Type	Description
lat	real	Latitude in degrees, of the LF transmitter antenna for the station for which this is an almanac. North is positive.
lon	real	Longitude in degrees, of the LF transmitter antenna for the station for which this is an almanac. East is positive.
range	integer	Published range of the station in km.
frequency	real	Station broadcast frequency in kHz.
health	integer	<health> is the health of the station for which this is an almanac. If it is non-zero, the station is issuing suspect data and should not be used for fixes. The ITU and RTCM104 standards differ about the mode detailed interpretation of the <health> field and even about its bit width.
station_id	integer	The id of the transmitter. This is not the same as the reference id in the header, the latter being the id of the reference receiver.
bitrate	integer	The transmitted bitrate.

Here's an example:

```
{"class":"RTCM2","type":9,"station_id":268,"zcount":252.6,
        "seqnum":4,"length":5,"station_health":0,
        "satellites":[
            {"ident":13,"udre":0,"iod":3,"prc":-25.940,"rrc":0.066},
            {"ident":2,"udre":0,"iod":73,"prc":0.920,"rrc":-0.080},
            {"ident":8,"udre":0,"iod":22,"prc":23.820,"rrc":0.014}
]}
```

Type 13: GPS Time of Week

Here are the payload members of a type 13 (Groumf Tramitter Parameters) message:

Table 25. Ground Transmitter Parameters

Name	Type	Description
status	bool	If True, signals user to expect a type 16 explanatory message associated with this station. Probably indicates some sort of unusual event.
rangeflag	bool	If True, indicates that the estimated range is different from that found in the type 7 message (which contains the beacon's listed range). Generally indicates a range reduction due to causes such as poor ionospheric conditions or reduced transmission power.
lat	real	Degrees latitude, signed. Positive is N, negative is S.
lon	real	Degrees longitude, signed. Positive is E, negative is W.
range	integer	Transmission range in km (1-1024).

This message type replaces message type 3 (Reference Station Parameters) in RTCM 2.3.

Type 14: GPS Time of Week

Here are the payload members of a type 14 (GPS Time of Week) message:

Table 26. Reference Station Parameters

Name	Type	Description
week	integer	GPS week (0-123).
hour	integer	Hour of week (0-167).
leapsecs	integer	Leap Seconds (0-63).

Here's an example:

```
{"class":"RTCM2","type":14,"station_id":652,"zcount":1657.2,
        "seqnum":3,"length":1,"station_health":6,"week":601,"hour":109,
        "leapsecs":15}
```

Type 16: Special Message

Table 27. Special Message

Name	Type	Description
message	string	A text message sent by the beacon operator.

Type 31: Correction data

One or more GLONASS satellite objects follow the header for type 1 or type 9 messages. Here is the format:

Table 28. Satellite object

Name	Type	Description
ident	integer	The PRN number of the satellite for which this is correction data.
udre	integer	User Differential Range Error (0-3). See the table following for values.
change	boolean	Change-of-ephemeris bit.
tod	uinteger	Count of 30-second periods since the top of the hour.
prc	real	The pseudorange error in meters for this satellite as measured by the beacon reference receiver at the epoch indicated by the z_count in the parent record.
rrc	real	The rate of change of pseudorange error in meters/sec for this satellite as measured by the beacon reference receiver at the epoch indicated by the z_count field in the parent record. This is used to calculate pseudorange errors at other epochs, if required by the GPS receiver.

Here's an example:

```
{"class":"RTCM2","type":31,"station_id":652,"zcount":1642.2,
    "seqnum":0,"length":14,"station_health":6,
```

```
    "satellites":[
        {"ident":5,"udre":0,"change":false,"tod":0,"prc":132.360,"rrc":0.000},
        {"ident":15,"udre":0,"change":false,"tod":0,"prc":134.840,"rrc":0.002},
        {"ident":14,"udre":0,"change":false,"tod":0,"prc":141.520,"rrc":0.000},
        {"ident":6,"udre":0,"change":false,"tod":0,"prc":127.000,"rrc":0.000},
        {"ident":21,"udre":0,"change":false,"tod":0,"prc":128.780,"rrc":0.000},
        {"ident":22,"udre":0,"change":false,"tod":0,"prc":125.260,"rrc":0.002},
        {"ident":20,"udre":0,"change":false,"tod":0,"prc":117.280,"rrc":-0.004},
        {"ident":16,"udre":0,"change":false,"tod":17,"prc":113.460,"rrc":0.018}
]}
```

RTCM3 DUMP FORMAT

The support for RTCM104v3 dumping is incomplete and buggy. Do not attempt to use it for production! Anyone interested in it should read the source code.

AIS DUMP FORMATS

AIS support is an extension. It may not be present if your instance of **gpsd** has been built with a restricted feature set.

AIS packets are dumped as JSON objects with class "AIS". Each AIS report object contains a "type" field giving the AIS message type and a "scaled" field telling whether the remainder of the fields are dumped in scaled or unscaled form. (These will be emitted before any type-specific fields.) It will also contain a "device" field naming the data source. Other fields have names and types as specified in the AIVDM/AIVDO Protocol Decoding document on the GPSD project website; each message field table may be directly interpreted as a specification for the members of the corresponding JSON object type.

By default, certain scaling and conversion operations are performed for JSON output. Latitudes and longitudes are scaled to decimal degrees rather than the native AIS unit of 1/10000th of a minute of arc. Ship (but not air) speeds are scaled to knots rather than tenth-of-knot units. Rate of turn may appear as "nan" if is unavailable, or as one of the strings "fastright" or "fastleft" if it is out of the AIS encoding range; otherwise it is quadratically mapped back to the turn sensor number in degrees per minute. Vessel draughts are converted to decimal meters rather than native AIS decimeters. Various other scaling conversions are described in "AIVDM/AIVDO Protocol Decoding".

SUBFRAME DUMP FORMATS

Subframe support is always compiled into **gpsd** but many GPSes do not output subframe data or the **gpsd** driver may not support subframes.

Subframe packets are dumped as JSON objects with class "SUBFRAME". Each subframe report object contains a "frame" field giving the subframe number, a "tSV" field for the transmitting satellite number, a "TOW17" field containing the 17 MSBs of the start of the next 12-second message and a "scaled" field telling whether the remainder of the fields are dumped in scaled or unscaled

form. It will also contain a "device" field naming the data source. Each SUBFRAME object will have a sub-object specific to that subframe page type. Those sub-object fields have names and types similar to those specified in the IS-GPS-200 document; each message field table may be directly interpreted as a specification for the members of the corresponding JSON object type.

Table 29. SUBFRAME object

Name	Always?	Type	Description
class	Yes	string	Fixed: "SUBFRAME"
device	Yes	string	Name of the originating device.
gnssId	Yes	integer	Constellation of transmitting satellite
tSV	Yes	integer	ID of transmitting satellite (Not PRN)
TOW17	No	integer	Type 17 bits of the next GPS Time Of Week
frame	No	integer	Frame number
scaled	Yes	boolean	True is values scaled

READING

Reading the raw JSON can be tedious. You can pretty print, and colorize, your JSON with [jq] to make reading easier. Using **jq** ito pretty pring a JSON file can be as simple as:

```
$ jq .  GPSD.json
```

To grab 10 seconds of live **gpsd** JSON, and pretty print it:

```
$ gpspipe -w -x 10 | jq .
```

If you only want to see the TPV messages:

```
$ gpspipe -w -x 10 | fgrep TPV | jq .
```

SEE ALSO

gpsd(8), **libgps**(3), **libgpsmm**(3), **jq**(1)

RESOURCES

Project web site: {gpsdweb}

[RFC 2783]: https://datatracker.ietf.org/doc/html/rfc2783
Pulse-Per-Second API for UNIX-like Operating Systems

[RFC 7159]: https://datatracker.ietf.org/doc/html/rfc7159
The JavaScript Object Notation (JSON) Data Interchange Format

[jq]: https://github.com/stedolan/jq

COPYING