LOGZILLA DOCUMENTATION

Query API Parameters

Common LogZilla Query API parameters for time_range, filter expressions, sorting, pagination, and timezone handling across event and system query types

LogZilla API · Generated April 29, 2026 · logzilla.ai/docs/logzilla-api/query-api-parameters

Query API Parameters

LogZilla Query API requests use common parameters for time ranges, filtering options, and result formats. These parameters provide flexible control over data retrieval and analysis operations across different query types.

Prerequisites

- Valid LogZilla API authentication token (see Getting Started (https://www.logzilla.ai/docs/logzilla-api/getting-started))
- Understanding of query creation workflow (see Making Queries (https://www.logzilla.ai/docs/logzilla-api/making-queries))
- JSON data format familiarity
- Boolean logic operators

Common Query Parameters

Although every query type defines its own list of parameters there are some parameters used by most of them:

time_range

For every query, the start and end time period of the desired data must be provided. For some queries, sub-periods within the given period are calculated based on the step parameter - i.e. when requesting event rates, results will contain values for each sub-period, such as all minutes in the last hour, or last 30 days, etc.

The time_range parameter is an object with the following fields:

ts_from
Timestamp (number of seconds from epoch) defining the beginning of the period, for which 0 (zero) can be used to use the current time, or a negative number to specify time relative to the current time

ts_to
Timestamp defining the end of the period. 0 or a negative number can be provided to get time relative to current

step
A single number defining the length of sub-periods in seconds; for example, 60 will create 1-minute periods, 900 will create 15-minute periods, and so on. The default is set heuristically according to ts_from and ts_to - i.e. when a 1 hour time range is requested `step` will be set to 1 minute, for the range of 1 minute or less `step` will be one second, and so on.

preset

As an alternative to ts_from and ts_to, the preset parameter can be used. Based on the timezone, it determines the start of the day and uses appropriate ts_from and ts_to values. Available presets include 'today', 'yesterday', or relative time patterns such as 'last_5_minutes', 'last_1_hour', 'last_7_days', 'last_2_weeks'

timezone

Determines the beginning of the day for preset parameter; by default, GLOBAL_TZ config value is used

Periods are always rounded to the nearest multiple of `step`. Rounding is always up so the last period is often partially in the future, such as if a step of 1 hour is requested and it is now 13:21 then the last period will be 13:00 - 14:00. This then results in results for the current hour being received despite the hour indicated not yet being complete.

For query types that do not use subperiods (such as "LastN") only ts_from and ts_to are important, but `step` and `round_to_step` to round can still be used. (Note that in earlier versions there was an option to provide `round_to_step` and `periods` parameters, which are now unsupported).

filter

By default, every query operates on all data (according to the given time range), but for each, a compound parameter "filter" can be provided which will filter results by selected fields (including as desired message text). This parameter is an array of filter conditions that are always "AND"-ed, meaning that each record must match all of the given conditions to be included in the final results. Filtering is always done before aggregating so if for example the event rate is queried and filtering is specified by hostname then only the number of events with this host name will be reported as the result count.

Every filter condition is an object with following fields:

name	description
`field`	name of the field to filter by, as it appears in results
`value`	actual value to filter by. for fields other than timestamp this can also be a list of possible values (only for "eq" comparison)
`op`	if type is numeric (this includes timestamps) this defines the type of comparison. see immediately below
`ignore_case`	determines whether text comparisons are case sensitive or not. Defaults to True, meaning all comparisons are case insensitive. To force case sensitive mode set ignore_case=False

operator	description
`eq`	value is an exact value to be found. this is the default when no comparison operator is specified. you can also specify list of possible values
`lt`	match only records with field less than the given value
`le`	match only records with field less than or equal to the given value
`gt`	match only records with field greater than the given value
`ge`	match only records with field greater than or equal to the given value
`qp`	special operator for message boolean syntax

Examples

Return only events with counter greater than 5:

```
[ { "field": "counter", "op": "gt", "value": 5 } ]
```

Return events from host 'fileserver23' with severity 'ERROR' or higher:

```
[ { "field": "severity", "value": [0, 1, 2, 3] },
  { "field": "host", "value": "fileserver23" } ]
```

Return events from hosts "alpha" and "beta" matching "power failure" in event message text:

```
[ { "field": "message", "value": "power failure" },
  { "field": "host", "value": ["alpha", "beta"] } ]
```

Message Boolean Syntax

Boolean logic expressions enable complex filtering of log message content using logical operators. These expressions allow combining multiple search terms and conditions to create precise filters for message text matching.

Supported Operators

AND

which is also implicitly used between two words/expressions if there is no other operator specified

NOT

shortcut: use either **'!'** or **'-'**

OR

shortcut: **'|'**

Operators are case-insensitive, so: 'AND', 'and', 'AnD' are correct

Examples of forbidden expressions:

```
-Foobar1
Foobar1 | -Foobar2
```

Examples of correct expressions:

Return events containing words 'Foobar1' or 'Foobar2' and not 'Foobar3':

```
[ { "field": "message", "op": "qp", "value": "Foobar1 | Foobar2 !Foobar3" } ]
```

Return events containing words ('Foobar1' or 'Foobar2') and ('Foobar3' or 'Foobar4'):

```
[ { "field": "message", "op": "qp", "value": "(Foobar1 | Foobar2) (Foobar3 | Foobar4)" } ]
```

Common Results Format

The "results" container is always an object with one or several fields, usually containing "totals" and/or "details". The former contains results for the whole period whereas the latter is an array of values for subperiods. Both total and subperiod usually contain "ts_from" and "ts_to" timestamps, to show exact time range for the data retrieved, and then the result "values" or just "count".

See the description of the particular query type in:

- Event Queries (https://www.logzilla.ai/docs/logzilla-api/event-query-types)
- System/Admin Queries (https://www.logzilla.ai/docs/logzilla-api/system-query-types)

for details on result formats and examples.

Generic Results Format for System Queries

System queries return data collected by the telegraf system, for different system parameters, and are used for displaying system widgets (that can be used later on for diagnostic monitoring of system performance).

All these queries return "totals" and "details". For details the objects are similar to data for EventRateQuery but there are more keys with different values. An example from *System_CPUQuery*:

```
{
  "details": [
    {
      "ts_from": 1416231300,
      "ts_to": 1416231315,
      "softirq": 0,
      "system": 8.400342,
      "idle": 374.946619,
      "user": 16.067144,
      "interrupt": 0.20001199999999997,
      "nice": 0,
      "steal": 0,
      "wait": 0.20001199999999997
    },
    "..."
  ]
}
```

For totals instead of an array we have a single object with keys like above, but rather than a single value there is a set of values:

```
{
  "system": {
    "count": 236,
    "sum": 1681.6008720000007,
    "min": 5.2671220000000005,
    "max": 9.599976,
    "avg": 7.125427423728817,
    "last": 6.400112999999999,
    "last_ts": 1416234840
  }
}
```

So here there are different kinds of aggregates for the selected time period:

type	description
`count`	number of known values for the given time period

type	description
`sum`	total of those values (used for calculating avg)
`min`	minimum value
`max`	maximum value
`avg`	average value (sum / count)
`last`	last known value from given period
`last_ts`	timestamp when last known value occurred

Related Documentation

- Making Queries (https://www.logzilla.ai/docs/logzilla-api/making-queries) - Creating queries and handling asynchronous requests
- Event Queries (https://www.logzilla.ai/docs/logzilla-api/event-query-types) and System/Admin Queries (https://www.logzilla.ai/docs/logzilla-api/system-query-types) - Detailed documentation for all query types