LOGZILLA DOCUMENTATION

Reports API

Create report templates from a preset, a dashboard, or an AI prompt, schedule them, generate reports, and download the stored files through the REST API

LogZilla API · Generated November 6, 2026 · logzilla.ai/docs/logzilla-api/reports-api

© LogZilla Corporation · logzilla.ai

Reports API

A report is produced from a report template. The template names the source of the report, its file type, and for some sources a preset code, a dashboard, or a prompt. A schedule attached to the template runs it on a recurring or one-time basis and mails the result; a template can also be run on demand. Each run stores a report file that stays downloadable.

The Reports permission is required for the template and schedule endpoints. For the full request and response schemas, use the interactive docs at `/api/docs`; see Getting Started (https://www.logzilla.ai/docs/logzilla-api/getting-started) for token handling.

Report Templates

`POST /api/reports-templates` creates a template. `GET`, `PUT`, `PATCH`, and `DELETE /api/reports-templates/{id}` read, replace, update, and remove it. A `PATCH` reads the fields it does not send from the stored template, so the rules below apply to the combined result.

Field	Meaning
`name`	Display name; also the name of every report the template produces
`source`	`preset`, `dashboard`, or `ai`
`preset_code`	The preset to run when `source` is `preset`; `GET /api/report-presets` lists the codes, their `formats`, and the apps they require
`dashboard`	The dashboard to export when `source` is `dashboard`
`prompt`	The prompt text when `source` is `ai`; see AI Reports below
`file_type`	The stored file's type; the allowed values depend on the source
`is_public`	Whether the template is listed for users other than its owner; administrators see every template

Source	Allowed `file_type` values	Default
`preset`	The preset's `formats`, `json` and `xlsx` for the shipped presets	`json`
`dashboard`	`json`, `xlsx`	`json`

Source	Allowed `file_type` values	Default
`ai`	`htmldoc`	`htmldoc`

Creating or updating a template returns 400 with the offending field as the key when:

- `preset_code` is not one of the available presets;
- `file_type` is not offered by the source, for example `csv` for a dashboard, or `htmldoc` for any source other than `ai`;
- `source` is `ai` and `prompt` is empty, or `prompt` is sent for a source other than `ai` (send `null` to clear it when switching a template away from `ai`, together with a `file_type` the new source offers);
- `source` is `ai` while LogZilla AI is disabled; the message says to enable AI features in Settings.

Generating a Report

`POST /api/reports-templates/{id}/generate` runs the template at once and answers 202 with the task:

```
{"task": "0b7f7b4e-4b7a-4a1c-9a4f-2d1c0e3f5a6b", "status": "STARTED"}
```

Poll `GET /api/async-results/{task}`; when its `status` is `SUCCESS`, `results` holds the report as returned by `GET /api/reports/{id}`. The request is refused with 400 before any task starts when the template cannot run: a preset whose required apps are not installed, or an AI template while LogZilla AI is disabled.

The request may carry a JSON body naming the recipients of the result:

```
{"email_to": "a@example.com,b@example.com"}
```

`email_to` is one string of addresses separated by commas, the shape a schedule stores. Each address is validated and a bad one is refused with 400 keyed `email_to` before any task starts. When recipients are named, the report is mailed as a scheduled run of the same template would mail it, with the same subject, body, and attachment. Without a body, or with `email_to` absent, null, or empty, the run stores the report and sends no mail.

Schedules

`POST /api/reports-schedules` attaches a schedule to a template. A template has one schedule; posting again for the same template updates it. `PUT`, `PATCH`, and `DELETE /api/reports-schedules/{id}` change or remove it, and `GET /api/reports-schedules/{id}/reports` lists the runs it produced.

Field	Meaning
`template`	The template id
`schedule_type`	`c` for a recurring schedule, `t` for one run at a set time, `a` for one run at the next check
`schedule`	`{"cron": {"minute": "0", "hour": "6", "day_of_week": "1"}}`, `{"timestamp": "2026-09-15T06:00:00Z"}`, or `{"adhoc": true}`
`email_to`	Required. Recipient addresses separated by commas; each is validated and the stored value has no spaces
`is_active`	Whether the schedule fires; on by default

Cron fields are wall-clock values in the server time zone. Due schedules are picked up within a minute. A run that fails leaves no report and, apart from the notice an AI report sends when it reaches its run time limit (see Caveats), sends no mail; a recurring schedule keeps its next run, and a one-time schedule is consumed.

Generated Reports

`GET /api/reports` lists the stored reports, filtered by `template`, `schedule`, `source`, `file_type`, or `document_id`. `GET /api/reports/{id}` returns the metadata and `DELETE` removes the report.

Each report carries `ready`, `true` once its file has been written. A preset or AI run lists its report before it writes the file, so a report with `ready: false` is still being generated (an AI run can take minutes); poll `GET /api/reports/{id}` or the list until `ready` is `true` before requesting the export. A run that fails removes its report. A report whose run was interrupted, for example by a restart of the worker, keeps `ready: false` and its export answers 400; an hourly check removes it once it is older than the Agent Max Run Time setting plus one hour (see Caveats), and `DELETE /api/reports/{id}` removes it sooner.

`GET /api/reports/{id}/export` downloads the file. The response carries the content type of the file type and a `Content-Disposition` header with a download name of the form `report_<date>.<extension>`:

`file_type`	Content type	Download extension
`json`	`application/json`	`.json`
`csv`	`text/csv`	`.csv`

file_type	Content type	Download extension
xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	.xlsx
htmldoc	text/html	.html

AI Reports

A template with `source` set to `ai` stores a prompt and runs it through the LogZilla AI model configured under Settings > System Settings > LogZilla AI. LogZilla AI must be enabled; see Installation of the LogZilla AI Module (https://www.logzilla.ai/docs/logzilla-ai/installation).

```
curl -H "Authorization: token YOUR_TOKEN" \
     -H "Content-Type: application/json" \
     -X POST "http://your-logzilla-server/api/reports-templates" \
     -d '{"name": "Daily critical summary", "source": "ai",
          "prompt": "Summarize the critical events of the last 24 hours by host."}'
```

The template stores the prompt text itself, not a reference to it. Changing the prompt with `PATCH` replaces the stored text; runs that already happened keep the report they produced.

Each run is one conversation between the configured model and the LogZilla query tools, the same tools the LogZilla AI chat uses: TopN, LastN, Search, EventRate, StorageStats, and a lookup of the fields an installed app defines. The model receives the current time in the instance time zone, the query rules, a summary of the installed apps, and the prompt text as stored with leading and trailing whitespace removed, and it queries the stored events before it answers. The queries run with the permissions of the report's owner: the user who scheduled the template, or the user who ran it on demand. When RBAC is enabled, host permissions filter every query, and an owner without the Search permission gets a report built from counts and top values without event details. See Role Based Access Control (https://www.logzilla.ai/docs/administration/role-based-access-control). Query results, including event text the owner may read, are sent to the model. The instance name, the owner's name, the schedule, and the recipients are not.

The answer is rendered as an HTML document with a header that names the template, the instance, the generation time, and the prompt, and a footer that names the software version and the model and links to the stored report. Headings, paragraphs, lists, tables, and code blocks in the answer are rendered; raw HTML, links, and images in the answer appear as text. When the schedule, or the run-now request, names recipients, the document is the body of the mail, with the subject `[LogZilla] AI Report: <template name>, <instance>, <date>` and no attachment. The same document is stored as the report file with `file_type` `htmldoc` and downloaded through `GET /api/reports/{id}/export` as `text/html`.

While LogZilla AI is disabled, creating an AI template or changing one is refused with 400, `POST /api/reports-templates/{id}/generate` is refused with 400, and a scheduled run produces no report and no mail and records an error naming the template in the celeryworker log. A model error or an empty answer likewise leaves no report.

Caveats

- A run is bounded by the Agent Max Run Time setting of the LogZilla AI settings group (`AGENT_MAX_RUN_TIME`, default 900 seconds, range 60 to 7200); see Installation of the LogZilla AI Module (https://www.logzilla.ai/docs/logzilla-ai/installation). A report that needs more fails, records an error naming the template in the celeryworker log, and mails the recipients named by the schedule or by the run-now request, or the owner of a run-now without recipients, a notice naming the template and the limit in place of the report; narrow the prompt's time range or filters, or raise the limit.
- The model decides which queries to run from the prompt; a prompt that names the time range, the hosts, or the severities it wants produces a more focused report.
- The prompt is sent as written; it supports no variables or templates.
- The answer is rendered as text and tables only. Links and images in the answer are shown as text, and no chart or PDF output is produced.
- The Reports page of the legacy interface lists AI templates and can run them but cannot create or edit them; AI templates are created and changed through the API.
- Switching a template away from `ai` needs `prompt` set to `null` and a `file_type` the new source offers in the same request; switching to `ai` needs a `prompt` and takes `htmldoc` without naming it.