

izibuddy CLI reference

izibuddy CLI reference

InZeMobile

TABLE DES MATIÈRES

Resources	5
Partie 1 — Getting started	
Chapter I — What the CLI is	9
Chapter II — Install and configure	11
Chapter III — Finding your way.....	13
Partie 2 — Using it	
Chapter IV — Calling any operation	17
Chapter V — Output and piping.....	19
Chapter VI — The shortcuts.....	21
Partie 3 — In a pipeline	
Chapter VII — Exit codes and quality gates	25
Chapter VIII — Recipes.....	27
Chapter IX — The packaged check.....	29
Appendices	
Appendix A — Every global option.....	35

Resources

The CLI lives in `src/sdk/python/izibuddy/cli.py`, alongside the SDK it wraps. It has no dependencies beyond the standard library, and it declares the `izibuddy` entry point in `src/sdk/python/pyproject.toml`.

It writes no list of operations: the parsers are built by introspecting the SDK's generated signatures, so the 252 operations reach the terminal through the same regeneration that reaches the library.

The SDK is `src/sdk/python/izibuddy/`, documented in book 11. Its TypeScript twin is book 12.

The guard is `tests/test_cli.js`. It asks the SDK what it exposes and the CLI to route each one — all 252 answer — and it holds the four exit codes apart, which is what makes the tool usable in a pipeline:

```
node tests/test_cli.js
```

The backend it talks to is `src/api/`, documented in book 09.

Télécharger

- Format PDF (13-CLI-reference_v1.7_en_20261008.pdf)
- Format EPUB (13-CLI-reference_v1.7_en_20261008.epub)
- Format Markdown (13-CLI-reference_v1.7_en_20261008.md)

Partie 1

Getting started

Chapter I — What the CLI is

`izibuddy` is a thin layer over the Python SDK. Every one of the backend's **252 operations** is callable from a shell, and none of them was written by hand.

1.1 — Thin means thin

```
from izibuddy import IziBuddy
izi = IziBuddy(os.environ["IZIBUDDY_URL"], token=os.environ.get("IZIBUDDY_TOKEN"))
r = izi.templates.render_template_by_id(uid, delivery="split")
```

That is the whole of what the CLI does per call. What it adds is the part a shell needs and a library does not: argument parsing, output that pipes, credentials that persist between invocations, and **exit codes a pipeline can read**.

1.2 — Why there is no list of commands

252 subcommands, hand-written, would be a second surface — and a second surface drifts from the first. The parsers are built instead by **introspecting the SDK's signatures**: path parameters, body, query parameters, docstring and route all come from the generated code.

The consequence is worth stating plainly: a route added to the backend reaches the CLI through the same regeneration that reaches the SDK. Nobody has to remember a third place.

`tests/test_cli.js` checks the promise operation by operation — it asks the SDK what it exposes and asks the CLI to document each one. All 252 answer.

1.3 — What it is for

Two audiences, and they want different things.

At a terminal: explore an API you do not know by heart. `izibuddy resources`, `izibuddy methods templates`, `izibuddy help templates render_template_by_id` — three commands and you are looking at the route and its arguments.

In a pipeline: render a prompt, fail the build when the render is not clean, and know from the exit code which of those two happened. That is what `--strict` and the separation of exit code 1 from exit code 2 exist for.

Skills and publication

The CLI exposes the generated API surface without an extra language parser:

```
izibuddy templates render_template_by_id UUID --target skill
izibuddy help templates publish_document
```

The skill emission is JSON containing `text`, `name` and `files` (relative paths plus base64 bytes). The caller writes these files in its chosen folder. Publication requires a stable document UUID, a UUID retry token and base revision. A conflict or pending push is not a successful remote publication. Keep local working data until completion. `--strict` remains an explicit CLI quality gate on warnings; it is distinct from the server's mixed-inheritance policy.

Templates and document nature

A **template** is the common reusable entity. Its **Nature** is `prompt`, `skill` or a custom identifier such as `mod` or `acme.review`. Create **New prompt** or **New skill**, then edit Nature in Properties when needed. Changing nature preserves the UUID, references, parameters and history. The leaf template owns its type; includes never change it. Mixed inheritance can be allowed with warnings or denied.

```
@@@template review_assistant
@@@type skill
@@@description Reusable review instructions
@@@section rules
Separate observations from assumptions.
```

Use `type:acme.review` to search a custom nature. The **Example** flag replaces the old Template flag; `flag:example` finds these starting examples. `flag:draft` continues to find private drafts. A prompt copies the rendered text, a skill exports SKILL.md and resources, and other types export composed text without claiming a specialized package format.

Chapter II — Install and configure

2.1 — Install

The CLI ships with the Python SDK and has **no dependencies** beyond the standard library.

```
pip install -e src/sdk/python      # depuis une copie du dépôt
izibuddy --version
```

The command ships **with** the Python SDK — one package, `izibuddy`, declaring the console script. Installing the SDK installs the command, and there is nothing else to fetch.

Once the package is published, that becomes `pip install izibuddy`. It is not on PyPI yet; book 11 says where that stands and what publishing takes.

Without installing :

```
PYTHONPATH=src/sdk/python python3 -m izibuddy.cli --version
```

2.2 — Where the URL comes from

```
export IZIBUDDY_URL=http://127.0.0.1:8765
izibuddy resources
```

or per call :

```
izibuddy --url http://127.0.0.1:8765 resources
```

With neither, the CLI refuses and says which environment variable to set. It does not guess a default: a client that silently talks to `localhost` is one that silently talks to the wrong instance.

2.3 — Where the token comes from

Three sources, in order — `--token`, then `IZIBUDDY_TOKEN`, then a saved file :

```
izibuddy login alice --save
# token saved to ~/.config/izibuddy/token
izibuddy whoami --field username --raw
# alice
```

`--save` writes the JWT to `~/.config/izibuddy/token` with mode `600`. A bearer token is not a hint about who you are — whoever reads it **is** you — so the file is created narrow.

Change the location with `IZIBUDDY_TOKEN_FILE`, which is what a CI runner should do to keep the token inside the job's workspace.

2.4 — Logging in without a terminal

`login` templates for the password when it is not given. In a pipeline, where there is nobody to prompt:

```
IZIBUDDY_PASSWORD="$SECRET" izibuddy login "$USER" --save --quiet
```

The password never appears in `argv`, so it does not show up in `ps` or in a shell history.

Chapter III — Finding your way

Three commands, and you never need to read the source.

3.1 — What exists

```
$ izibuddy resources
admin                8 operation(s)
admin_alerts         7 operation(s)
analytics            12 operation(s)
...
templates            57 operation(s)
...
26 resources, 252 operations
```

`--json` gives the same as a mapping, for a script.

3.2 — What a resource can do

```
$ izibuddy methods authentication
auth_status          GET /api/auth/status
change_password       POST /api/auth/change-password
check_auth           GET /api/auth/check
complete_password_reset POST /api/auth/password-reset/complete
...
```

Each line is the method name and the route it calls. Naming methods after the backend's own handlers is what makes that second column useful: you can grep the route in `src/api/` and land on the function.

Typing a resource with no method does the same thing, so `izibuddy templates` is a listing rather than an error.

3.3 — How to call one

```
$ izibuddy help templates render_template_by_id
izibuddy templates render_template_by_id <template_id> [--delivery ...]

Render Template By Id.

Rend un prompt avec résolution des variables

`GET /api/templates/render/{template_id}`
```

The usage line is built from the signature, so it cannot describe arguments the method does not take. The prose is the backend developer's docstring, carried through unchanged — including its language.

3.4 — When you nearly get it right

```
$ izibuddy templates render_template
izibuddy: unknown operation: templates render_template – did you mean: render_template_
```

Exit code 3. The near misses are computed from the resource's own methods, which is why they are always real.

Partie 2

Using it

Chapter IV — Calling any operation

4.1 — The shape

```
izibuddy <resource> <method> [<path parameters>...] [--body ...] [--<query> ...]
```

Path parameters are positional, in the order the route states them. Query parameters are flags named after themselves. A body is `--body`.

```
izibuddy templates render_template_by_id 9e0fe140-... --delivery split
izibuddy tags_v2 merge_tags ancien nouveau
izibuddy users get_user_web_settings alice
```

4.2 — Bodies come from three places

```
izibuddy templates create_template --body '{"name":"Note","content":"..."}'
izibuddy templates create_template --body @prompt.json
cat prompt.json | izibuddy templates create_template --body -
```

Inline for a quick hand-typed call, `@file` for a payload a repository should version, `-` for a pipeline that composes. Anything that is not valid JSON fails with exit code 3 and the parser's message, before any request is sent.

4.3 — Two identifiers, and the CLI cannot save you from mixing them

A prompt has a storage `id` — a uid — and a POO²L `template_id`. **Most routes want the uid.**

```
$ izibuddy templates list_all_templates --field templates
```

```
> { "id": "a48abd91-a04f-41f4-8a99-fbf2179e6625", "template_id": "CLI_NOTE", "name":
```

Passing `CLI_NOTE` where a uid belongs produces a not-found that reads oddly, because the backend looked up exactly what it was handed. Nothing in the URL distinguishes the two, so nothing in the CLI can.

4.4 — Paths that contain slashes

Seven routes take a parameter that may contain `/` — a nested collection, a hierarchical tag. Pass it as-is :

```
izibuddy templates update_collection 'Marketing/2026' --body @collection.json
```

The SDK knows which parameters those are and does not percent-encode their slashes. Quote the argument so the shell does not touch it either.

Chapter V — Output and piping

5.1 — By default, JSON on stdout

Pretty-printed, keys sorted — so a diff between two runs shows what changed rather than what moved.

```
$ izibuddy authentication auth_status
{
  "auth_required": true,
  "has_users": true,
  "show_register": true
}
```

5.2 — One key, and one value

```
$ izibuddy whoami --field username --raw
sdk_demo
```

`--field` picks a key from the response; `--raw` prints a string without JSON quotes. Together they produce something a shell can assign:

```
NAME=$(izibuddy whoami --field username --raw)
```

Without `--raw` the same call prints `"sdk_demo"`, quotes included — correct JSON, and wrong for a variable.

A `--field` naming a key the response does not have is exit code 3, not an empty line. An empty line is indistinguishable from a legitimately empty value, and a pipeline would carry it forward.

5.3 — Bytes stay bytes

Attachments, CSV exports and guide pages are not JSON. They go to stdout untouched, so a redirect is enough:

```
izibuddy templates get_template_attachment "$UID" nda.pdf > nda.pdf
izibuddy analytics export_analytics_csv > analytics.csv
```

Nothing is decoded «just in case» — that would corrupt the only case that matters.

5.4 — Where the noise goes

Everything that is not the answer goes to `stderr`: warnings, error messages, the confirmation that a token was saved. So this is safe :

```
izibuddy render "$UID" > prompt.txt
```

`prompt.txt` holds the rendered prompt and nothing else, even when the render carried warnings.

`--quiet` suppresses the answer entirely and keeps only the exit code, for a call made for its effect rather than its result.

Chapter VI — The shortcuts

Three commands are hand-written rather than introspected. Each earns it by doing something the generic form cannot.

6.1 — login

```
izibuddy login alice --save
izibuddy login alice --password "$SECRET" --quiet
IZIBUDDY_PASSWORD="$SECRET" izibuddy login alice --save --quiet
```

The generated `izibuddy authentication login --body '{...}'` would work and would print the token, leaving you to carry it. `login` **remembers** it — that is the whole difference, and it is what turns a sequence of calls into a session.

Without `--password` and without `IZIBUDDY_PASSWORD`, it templates. The prompt does not echo, and a password given through the environment never reaches `argv`, so it stays out of `ps` and out of shell history.

6.2 — whoami

```
$ izibuddy whoami --field username --raw
sdk_demo
```

A shorthand for `authentication get_me` that unwraps the `user` object, because `{"success": true, "user": {...}}` is a shape you would otherwise strip on every call.

6.3 — render

The reason the CLI exists, for most pipelines.

```
$ izibuddy render a48abd91-a04f-41f4-8a99-fbf2179e6625
## Corps

Bonjour depuis le CLI.

izibuddy: warning: MISSING_DESCRIPTION: prompt 'Note CLI' has no @@@description
```

Three things separate it from `templates render_template_by_id`:

- it prints `rendered` rather than the whole JSON envelope, so the output is the prompt and a redirect gives you a usable file;
- it checks `success` and turns a failure reported in the body into exit code 1 — see the next chapter, because this is the trap;

- it carries `--strict` and `--attachments`.

```
izibuddy render "$UID" --delivery split --attachments
```

With `split`, the render returns the text and the attachment block separately; `--attachments` prints the block after the text instead of discarding it.

6.4 — Everything else, and why

There is no `create`, no `delete`, no `ls`. They would be aliases with nothing added — and each one would be a name to learn on top of the name the API already has. `izibuddy templates create_template` is one word longer and needs no translation table.

Partie 3

In a pipeline

Chapter VII — Exit codes and quality gates

7.1 — Four codes, and they mean four different things

code	meaning
0	success
1	failure — HTTP error, network, or the backend reporting failure in the body
2	quality gate — <code>--strict</code> and the response carried warnings
3	usage — unknown resource or method, bad JSON, missing URL

2 is separate from 1 on purpose. A pipeline wants to distinguish *the call failed* from *the call succeeded and the result is not clean enough*. The second is a team's policy ; the first is a fact.

3 is separate because a typo is not a failing build.

7.2 — The trap : a failure that answers 200

Several endpoints report failure **in the body**, with HTTP 200 :

```
{"success": false, "error": "Template with id 'inexistent' not found"}
```

Nothing went wrong at the HTTP level, so a client that looks only at the status would exit 0 — and a CI job would go green on work that was not done.

The CLI checks `success` and exits 1 :

```
$ izibuddy render inexistant
izibuddy: cannot render: Template with id 'inexistent' not found
$ echo $?
1
```

7.3 — `--strict`

```
$ izibuddy render "$UID" --strict > prompt.txt
izibuddy: warning: MISSING_DESCRIPTION: prompt 'Note CLI' has no @@@description
izibuddy: --strict: 1 warning(s), exit 2
$ echo $?
2
```

The render **succeeded** — `prompt.txt` holds the text. `--strict` says that a prompt with warnings is not fit to ship, and the exit code carries that decision to whoever called.

Without `--strict` the warnings still print on `stderr` and the exit code is `0`. Nothing is hidden; only the verdict changes.

7.4 — Reading them in a script

```
izibuddy render "$UID" --strict > prompt.txt
case $? in
  0) echo "clean" ;;
  1) echo "call failed" ; exit 1 ;;
  2) echo "warnings – rendered anyway, in prompt.txt" ; exit 1 ;;
  3) echo "usage error" ; exit 1 ;;
esac
```

That the four are distinct is not an accident of implementation: it was got wrong twice while writing this CLI. `argparse` exits `2` on a usage error, which would have made a typo read as a quality-gate failure. Both parsers now exit `3`, and `tests/test_cli.js` holds the line.

Chapter VIII — Recipes

8.1 — A quality gate in CI

Read the next chapter first. `izibuddy check` and the GitHub Action do this in one line, versioned with the tool. What follows is for the cases they do not cover — another CI, or a loop that needs a decision of its own.

```
#!/usr/bin/env bash
set -euo pipefail
export IZIBUDDY_URL="$IZIBUDDY_URL"
export IZIBUDDY_TOKEN_FILE="$PWD/.izibuddy-token"

IZIBUDDY_PASSWORD="$IZIBUDDY_PASSWORD" izibuddy login "$IZIBUDDY_USER" --save --quiet

failed=0
for uid in $(izibuddy templates list_all_templates --field templates \
            | python3 -c 'import sys,json; [print(p["id"]) for p in json.load(sys.st
if ! izibuddy render "$uid" --strict --quiet; then
    echo "::warning::prompt $uid renders with warnings"
    failed=1
fi
done
exit $failed
```

`set -e` is why `--quiet` matters here: the call is made for its exit code, not its output. And `IZIBUDDY_TOKEN_FILE` keeps the token inside the job's workspace rather than in the runner's home.

8.2 — Export every prompt

```
mkdir -p export
izibuddy templates list_all_templates --field templates \
| python3 -c 'import sys,json; [print(p["id"], p["template_id"]) for p in json.load
| while read -r uid name; do
    izibuddy render "$uid" > "export/${name}.txt"
done
```

Warnings go to stderr, so every `.txt` holds a prompt and nothing else.

8.3 — Create from a file, and keep the id

```
uid=$(izibuddy templates create_template --body @prompt.json --quiet \  
    && izibuddy templates list_all_templates --field templates \  
    | python3 -c 'import sys,json; print(json.load(sys.stdin)[0]["id"])')
```

Creation reports the outcome, not the object — no uid comes back — so the id is read from the list afterwards. Worth knowing before writing the pipeline that assumes otherwise.

8.4 — Watch a value

```
watch -n 30 'izibuddy status get_queue_stats --field pending --raw'
```

8.5 — Explore an unfamiliar corner

```
izibuddy methods git  
izibuddy help git commit_changes  
izibuddy git get_git_status
```

Three commands from « what can this do » to a call, without opening the source.

8.6 — Use the SDK when the shell stops helping

The CLI is a layer, not a wall. A loop that reads a value, decides, and calls again is a program — and the SDK is right there :

```
from izibuddy import IziBuddy  
izi = IziBuddy(os.environ["IZIBUDDY_URL"], token=os.environ["IZIBUDDY_TOKEN"])  
for p in izi.pages("GET", "/api/templates/list", cle="templates"):  
    r = izi.templates.render_template_by_id(p["id"])  
    if r.get("warnings"):  
        print(p["template_id"])
```

Book 11 documents it.

Chapter IX — The packaged check

A recipe is a script each team copies into its own repository. It carries its bugs along, and nobody can fix it for everyone — there is nothing to update. This chapter is the same gate as a **name**: one command, versioned with the tool, and a GitHub Action wrapping it.

9.1 — izibuddy check

```
$ izibuddy check
WARN NOISY_ONE MISSING_DESCRIPTION: prompt 'Noisy' has no @@@description
2 prompt(s) · 0 failed · 1 with warnings · 0.0s
$ echo $?
2
```

It lists every prompt, renders each one, and reports. Where `render` answers about one prompt, `check` answers about the collection — which is the question a build asks.

option	
<code>--select GLOB</code>	only templates whose id matches, e.g. <code>--select 'CONTRACT_*'</code>
<code>--fail-on errors\ warnings</code>	what counts as a failing build; <code>warnings</code> by default
<code>--format text\ json\ junit\ github</code>	
<code>--report FILE</code>	also write it; a <code>.xml</code> path writes JUnit, anything else JSON
<code>--delivery attach\ inline\ split</code>	passed through to each render

9.2 — A verdict, and a fact, kept apart

A render that **cannot complete** is exit `1` in every case. That is not a matter of policy: the prompt is broken.

Warnings are policy, and `--fail-on` is where it lives:

```
izibuddy check                # avertissements = échec → 2
izibuddy check --fail-on errors # avertissements tolérés → 0
```

Nothing else decides. `--fail-on errors` still prints every warning — it changes the verdict, never the report.

9.3 — Formats a CI can read

`--format github` writes annotations, so the failures appear **on** the job rather than inside a log nobody unrolls:

```
::warning title=izibuddy: NOISY_ONE::MISSING_DESCRIPTION: prompt 'Noisy' has no @@@desco
```

`--format junit` writes the format nearly every CI can ingest and display next to its tests:

```
<testsuite name="izibuddy check" tests="2" failures="1" time="0.067">
  <testcase classname="izibuddy.prompt" name="CLEAN_ONE">
  </testcase>
  <testcase classname="izibuddy.prompt" name="NOISY_ONE">
    <failure type="warning">MISSING_DESCRIPTION: prompt 'Noisy' has no @@@description</
  </testcase>
</testsuite>
```

`--format json` gives `{total, failed, warned, seconds, results[]}`, each result carrying `uid`, `template_id`, `status`, `detail` and `warnings`.

9.4 — The GitHub Action

```
- uses: Jacques66/appli-izibuddy/.github/actions/izibuddy-check@main
  with:
    url: ${ secrets.IZIBUDDY_URL }
    token: ${ secrets.IZIBUDDY_TOKEN }
    fail-on: warnings
    report: izibuddy-check.xml

- uses: actions/upload-artifact@v4
  if: always()
  with:
    name: izibuddy-check
    path: izibuddy-check.xml
```

It is a **composite** action: no Docker image to pull, no Node runtime imposed on the runner. It installs the SDK, logs in if you gave a username and password instead of a token, runs the check with `--format github`, and exposes three outputs a later job can read:

```
- id: check
  uses: ./.github/actions/izibuddy-check
  with: { url: '...', token: '...' }
- run: echo "${ steps.check.outputs.warned }" templates carry warnings"
```

9.5 — A token, not a password

Both work — `token`, or `username` and `password` — but prefer the token: it can be scoped and revoked without touching an account, and it never has to be typed by a person.

Either way the action sets `IZIBUDDY_TOKEN_FILE` to the runner's temporary directory rather than letting it default to the home directory. On a shared runner the home survives the job; the workspace does not.

9.6 — When you still want a script

The recipe in the previous chapter is not wrong, and outside GitHub Actions it is what you have. Reach for it when the loop needs a decision the action does not take — checking only what a pull request changed, for instance, or opening an issue on failure. `izibuddy check --format json` is the seam.

Appendices

Appendix A — Every global option

Each of these works **before or after** the subcommand: `izibuddy --json resources` and `izibuddy resources --json` are the same command.

option	effect
<code>--url URL</code>	backend URL ; defaults to <code>IZIBUDDY_URL</code>
<code>--token JWT</code>	bearer token ; defaults to <code>IZIBUDDY_TOKEN</code> , then the saved file
<code>--timeout S</code>	seconds, default 30
<code>--strict</code>	exit 2 when the response carries warnings
<code>--quiet , -q</code>	print nothing on stdout ; keep the exit code
<code>--raw</code>	print a string without JSON quotes
<code>--field CLÉ</code>	print only that key of the response
<code>--json</code>	JSON output for the discovery commands
<code>--version</code>	

A.1 — Environment

variable	
<code>IZIBUDDY_URL</code>	backend URL
<code>IZIBUDDY_TOKEN</code>	bearer token
<code>IZIBUDDY_TOKEN_FILE</code>	where <code>--save</code> writes ; default <code>~/.config/izibuddy/token</code>
<code>IZIBUDDY_PASSWORD</code>	password for a non-interactive login

`XDG_CONFIG_HOME` moves the default token location, following the usual convention.

A.2 — Commands

command	
<code>resources</code>	list the 26 resources with their operation counts
<code>methods <resource></code>	list a resource's operations and their routes

command	
<code>help <resource> <method></code>	signature, documentation and route
<code>login <user></code>	open a session; <code>--save</code> remembers the token
<code>whoami</code>	the authenticated user
<code>render <uid></code>	render a prompt; <code>--delivery</code> , <code>--attachments</code> , <code>--strict</code>
<code><resource> <method> ...</code>	any of the 252 operations

A.3 — Per-operation options

`--body` appears only for operations that take one; each query parameter becomes a flag named after itself. Both come from the SDK signature, so `izibuddy help <resource> <method>` is always exact — it cannot describe an argument the method does not accept.

