

TypeScript SDK for izibuddy

TypeScript SDK for izibuddy

InZeMobile

TABLE DES MATIÈRES

Resources	5
Partie 1 — Getting started	
Chapter I — What this SDK is.....	9
Chapter II — Install, build and first call	11
Chapter III — Authentication and sessions.....	13
Partie 2 — Using the client	
Chapter IV — Types, resources and method names	17
Chapter V — Errors	19
Chapter VI — Working with templates	21
Chapter VII — Runtimes, bytes and slashes	23
Partie 3 — Keeping it true	
Chapter VIII — Regenerating after an API change.....	27
Chapter IX — Publishing and consuming	29
Appendices	
Appendix A — The twenty-six resources.....	33

Resources

The SDK lives in `src/sdk/typescript/` of the IziBuddy repository :

- `src/client.ts` — transport, session, typed errors, pagination, and the two path encoders. Written by hand ; the generator never touches it.
- `src/generated.ts` — the 252 operations and 94 interfaces, produced from the backend's OpenAPI schema.
- `src/index.ts` — wires the resources onto the client and merges their types.
- `package.json` — no runtime dependencies ; Node 18 or later, or a browser.

The generator is `src/sdk/generate.py`. It boots the backend in-process, reads its schema, and rewrites the generated file of this SDK and of its Python twin — one reading, two clients :

```
python3 src/sdk/generate.py
npm --prefix src/sdk/typescript run typecheck
```

The guards are `tests/test_sdk_drift.js`, which compares the committed files with a fresh generation byte for byte, and `tsc`, which caught a real name collision between a backend model and one of the client's error classes.

The Python twin is documented in book 11 ; the `izibuddy` command that wraps it, in book 13.

Télécharger

- Format PDF (12-SDK-typescript_v1.2_en_20261008.pdf)
- Format EPUB (12-SDK-typescript_v1.2_en_20261008.epub)
- Format Markdown (12-SDK-typescript_v1.2_en_20261008.md)

Partie 1

Getting started

Chapter I — What this SDK is

`@izibuddy/sdk` is a TypeScript client for the IziBuddy backend — **252 operations** across **26 resource groups**, fully typed, with no runtime dependencies.

1.1 — Generated where it should be, written where it matters

The surface is not hand-written. It is **generated from the backend's OpenAPI schema**, which FastAPI derives from its route decorators and Pydantic models. One generator (`src/sdk/generate.py`) emits both this SDK and its Python twin, from the same reading of the same schema — which is why the two never disagree about what the API offers.

`src/generated.ts` holds the 252 methods and 94 interfaces. `src/client.ts` holds the transport, the session, the error classes and pagination, and is **written by hand**. The generator never touches it.

That split is what makes generation bearable. A route added to `src/api/` appears here by regenerating; a decision about how a 422 should feel is made once and stays made.

1.2 — Types that come from the server, not from a guess

The 94 interfaces are the backend's own Pydantic models, transcribed. Where a route declares a response model, the method's return type is that interface; where it returns a bare dictionary, the type is `unknown` — which is the honest answer and forces the caller to narrow rather than to hope.

They are **types, not validators**. Nothing is checked at runtime: the backend already validated, and validating twice means two answers to keep in agreement.

1.3 — What keeps it honest

`tests/test_sdk_drift.js` regenerates both SDKs into a throwaway directory and compares byte for byte with what is committed. If the API moved and the SDK did not, that test is red and names the first differing line.

`npm run typecheck` is the second guard, and it earns its place: the first build failed because a backend model is called `ValidationError` and so is one of the client's error classes. The generator now renames a colliding model to `ApiValidationError` — the hand-written name wins, and both SDKs apply the same rule so the two books can say the same thing.

1.4 — What this SDK is not

It is not a re-implementation of the backend. It does not parse POO²L, resolve includes, or know what a section is: the engine does that, server-side.

It is not an abstraction over HTTP either. Method names are the backend's handler names, camel-cased — `renderTemplateById` is `render_template_by_id` in `src/api/templates.py`. You can grep from one to the other.

Skills and publication

Generated `TemplateRequest / TemplateUpdate` accept `document_type: "skill"`. Conversions retain identity. `templates.publishDocument` accepts the stable UUID, a UUID token, base revision and document. Keep draft saves in local client storage; use publication explicitly. Review HTTP 409 without automatic overwrite, and retain the draft while `publication_pending` is true. Dependencies are resolved from published states, never published implicitly.

Render with `target: "skill"` to obtain the emission: SKILL.md text and relative files with base64 bytes. Saving or installing those files belongs to the host. No update of a source automatically replaces an installed skill.

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, build and first call

2.1 — Install

The package lives at `src/sdk/typescript/`. It has **no runtime dependencies** — it uses the platform's `fetch`, `AbortController` and `URLSearchParams`, all standard since Node 18.

```
cd src/sdk/typescript
npm install      # typescript, uniquement pour compiler
npm run build    # émet dist/
```

Is it on npm? Not yet, and this book will say so until it is. The package is `@izibuddy/sdk`, its tarball is built and verified on every release run — installed into a clean consumer project and imported — and publishing is one tag away (`git tag sdk-v0.1.0`). The release workflow publishes with `--provenance`, which attaches to the package the verifiable record of the commit and workflow that produced it. Until that tag is pushed, `npm install @izibuddy/sdk` fetches nothing.

`typescript` is a `devDependency`: a program consuming the built `dist/` installs nothing.

2.2 — Requirements

Node 18 or later, or any browser. `package.json` declares `"type": "module"` and the build emits ES2022 modules — the imports carry explicit `.js` specifiers, which is what Node's ESM loader requires. Omitting them is the first thing that breaks a TypeScript package at runtime while the compiler stays silent, and it is the mistake this build made once.

2.3 — The first call

```
import { IziBuddy } from '@izibuddy/sdk';

const izi = new IziBuddy('http://127.0.0.1:8765');
console.log((await izi.api.health()).status); // 'healthy'
```

`health` needs no authentication, which makes it the right first call: it proves the base URL, the port and the network path before any question of credentials.

2.4 — Options

```
const izi = new IziBuddy('http://127.0.0.1:8765', {
  token: jetonDejaObtenu, // ou : izi.login(...)
```

```
    timeoutMs: 60_000,          // défaut 30 000
    fetch: monFetch,           // tests, proxy, runtime sans fetch global
  });
```

The `fetch` option is the seam that keeps this package testable and portable: a test double, a proxy-aware wrapper, or a polyfill goes in there. The constructor throws immediately if no `fetch` is available, rather than failing later on the first call.

The timeout is enforced with `AbortController` and cleared on every path, including the failure ones. A timer left running is how a Node process refuses to exit.

2.5 — What the client is

```
izi.baseUrl    // 'http://127.0.0.1:8765'
izi.token       // string | undefined
izi.templates   // TemplatesResource – 57 méthodes
```

The 26 resources are attached in the constructor, so `izi.templates` and its siblings exist immediately, typed, with no lazy loading and nothing to await.

Chapter III — Authentication and sessions

3.1 — The model

The backend authenticates with a **JWT bearer token**. `POST /api/auth/login` returns one; protected routes expect `Authorization: Bearer ...`. There is no server-side session, which is why logging out is a client-side act.

3.2 — Logging in

```
const izi = new IziBuddy('http://127.0.0.1:8765');
await izi.login('alice', 'secret');
const { user } = await izi.authentication.getMe();
console.log(user.username);
```

`login` is hand-written rather than generated, for one reason: it **remembers** the token. The generated `izi.authentication.login(...)` returns the same payload and leaves you to carry it.

```
izi.token;    // 'eyJhbGciOiJIUzI1NiIs...'
izi.logout(); // oublie le jeton, rien de plus
```

3.3 — Carrying a token you already have

A browser application that received a token from its own session, or a worker handed one by a queue, passes it straight in:

```
const izi = new IziBuddy(base, { token: jetonRecu });
```

`token` is a mutable public field, so a refresh flow can replace it in place without rebuilding the client and its 26 resources.

3.4 — Knowing before you ask

Two calls need no token:

```
await izi.authentication.authStatus();
// { auth_required: true, has_users: false, show_register: true }

await izi.authentication.getRegistrationConfig();
// { success: true, self_registration_enabled: true,
//   has_domain_restriction: false, email_enabled: false }
```

`has_users: false` on a fresh instance means the first account still has to be created — and the first account is created as an administrator.

3.5 — When the token is refused

```
import { AuthError } from '@izibuddy/sdk';

try {
  await izi.admin.restartApplication();
} catch (e) {
  if (e instanceof AuthError) console.log(e.status, e.detail); // 403 ...
  else throw e;
}
```

401 and 403 both raise `AuthError` : from the caller's side they are the same problem — *this identity may not do this*. The `status` field keeps the distinction for anyone who needs it.

Partie 2

Using the client

Chapter IV — Types, resources and method names

4.1 — Resources are the backend's tags, camel-cased

backend tag	property
templates	izi.templates
authentication	izi.authentication
tags-v2	izi.tagsV2
admin-alerts	izi.adminAlerts
template-builder-config	izi.templateBuilderConfig

The mapping is mechanical, and the properties are declared on an interface the generator emits (`Resources`), merged into the class by declaration merging. That is what makes `izi.` complete correctly in an editor: the fields are assigned in a loop at runtime, but their types are stated, not inferred.

4.2 — Method names are the backend's function names

```
await izi.templates.renderTemplateById('abc123'); // def render_template_by_id(...  
await izi.tagsV2.mergeTags('a', 'b');           // def merge_tags(...)
```

Where two routes in one group share a handler name — the same function serving `GET` and `POST` — the verb is appended, and only then.

4.3 — Arguments follow the route

Path parameters first, positionally; then the body; then an optional `query` object:

```
await izi.templates.renderTemplateById('abc123', { delivery: 'split' });  
await izi.templates.updateTemplate('abc123', { content: nouveauTexte });
```

A query value left `undefined` or `null` is **not sent**. `?delivery=undefined` is not the same request as omitting the parameter, and the backend distinguishes them.

4.4 — Return types, and where they stop

```
const r = await izi.templates.renderTemplateById(uid);  
//    ^ unknown — la route ne déclare pas de modèle de réponse
```

Ninety-four interfaces are generated from the backend's Pydantic models and used wherever a route declares one. Many prompt routes return a bare dictionary, so their type is `unknown` — deliberately. Typing them `any` would let `r.rendered` compile and fail at runtime; `unknown` forces a narrowing that puts the assumption in the code where a reader can see it:

```
interface Rendu { success: boolean; rendered: string; warnings?: unknown[] }  
const r = await izi.templates.renderTemplateById(uid) as Rendu;
```

That cast is a claim about the server. Writing it down is better than having it implied.

4.5 — Enumerating at runtime

```
import { RESSOURCES_TS, CHEMINS_LIBRES } from '@izibuddy/sdk';  
  
Object.keys(RESSOURCES_TS); // les 26 noms de ressources  
CHEMINS_LIBRES;           // les 7 paramètres de chemin qui acceptent des `/'`
```

Chapter V — Errors

5.1 — One family, six names

Every refusal throws a subclass of `IziBuddyError` carrying the same four facts: `status`, `detail`, `method`, `path`.

class	status	means
<code>AuthError</code>	401, 403	no token, expired token, or not allowed
<code>NotFound</code>	404	
<code>Conflict</code>	409	the state forbids it
<code>ValidationError</code>	422	the body does not satisfy the backend's model
<code>ServerError</code>	5xx	
<code>IziBuddyError</code>	anything else, and 0 for a network failure	

```
import { IziBuddyError, NotFound, ValidationError } from '@izibuddy/sdk';

try {
  await izi.templates.getTemplateById(uid);
} catch (e) {
  if (e instanceof NotFound) { /* ... */ }
  else if (e instanceof IziBuddyError) console.error(e.status, e.path);
  else throw e;
}
```

`instanceof` works because each class sets `this.name` from `new.target`, and because the classes are exported as values, not as types.

5.2 — 422 is the useful one

FastAPI answers a malformed body with a list saying which field, why, and what was received. `detail` carries it untouched:

```
catch (e) {
  if (e instanceof ValidationError) {
    for (const f of e.detail as Array<{loc: string[]; msg: string}>) {
      console.log(f.loc.join('.'), '→', f.msg);
    }
  }
}
```

```
}  
}  
}
```

body.email → value is not a valid email address: The part after the @-sign is a special-use or reserved name that cannot be used with email.

Flattening that into a string throws away `loc` — the only part a caller can act on.

5.3 — Not every failure is an exception

This is the caveat to read twice. A number of endpoints report failure in the body, with HTTP 200:

```
await izi.templates.getTemplateById('nexistepas');  
// { success: false, error: 'Template nexistepas not found' }
```

Nothing throws, because nothing went wrong at the HTTP level: the server understood the question and answered it. Code that only wraps calls in `try/catch` will read `result.prompt`, get `undefined`, and blame the SDK.

Until the backend is uniform: **catch AND check success when the payload has one.** The SDK does not turn `{success: false}` into an exception on your behalf — it would be wrong the day an endpoint returns that key with a meaning of its own.

5.4 — A network failure is not a status

Unreachable host, refused connection, abort on timeout: `IziBuddyError` with `status === 0` and the reason in `detail`. Same family, because a caller that wants to retry or give up treats them together — but the zero says plainly that the server never answered, which is a different fact from a bad answer.

5.5 — What the SDK never does

It does not retry, does not swallow, and never turns a failure into a value a caller might mistake for an empty result. Retry policy belongs to the program that knows whether an operation is idempotent — and many of the 252 are not.

Chapter VI — Working with templates

Templates carry **57 operations**, more than a fifth of the API. What follows is an actual run against a live backend.

6.1 — Create

`content` is POO²L source. The backend stores it and the engine renders it; the SDK carries the string and does not read it.

```
const source =
  '@@@template ts_note\n' +
  '@@@name Note TS\n' +
  '@@@section corps : Corps\n' +
  'Bonjour depuis TypeScript.\n';

await izi.templates.createTemplate({
  name: 'Note TS',
  template_id: 'ts_note',
  content: source,
});
// { success: true, message: 'Template created successfully', version: '0.1' }
```

The answer reports the outcome, not the object: no uid comes back.

6.2 — Two identifiers, and they are not interchangeable

```
const { templates } = await izi.templates.listAllTemplates() as { templates: any[] }
templates[0].id;           // '9e0fe140-...' - identité de STOCKAGE
templates[0].template_id; // 'TS_NOTE'     - identité POO2L
```

`id` is the storage uid; `template_id` is the name POO²L resolves — case- and diacritic-insensitive, hence the upper case. **Most routes want the uid.** Passing a `template_id` where a uid is expected produces a not-found that reads oddly, because the backend looked up exactly what it was handed.

6.3 — Render

```
const r = await izi.templates.renderTemplateById(templates[0].id);
```

```
"# Note TS (TS_NOTE)\n\n## Corps\n\nBonjour depuis TypeScript.\n"
```

The `#` and `##` are POO²L structure headings emitted by the engine: level 1 names the instantiated prompt, level 2 its section. They are part of the contract, not decoration — a client that strips them is changing what the model receives.

`warnings` carries advisory diagnostics. They never block a render, and ignoring the key silently discards them.

6.4 — Delivery: where the attachments go

Attachments live in the record, never in the prompt text.

```
await izi.templates.renderTemplateById(uid, { delivery: 'attach' }); // défaut
await izi.templates.renderTemplateById(uid, { delivery: 'inline' }); // texte + bloc
await izi.templates.renderTemplateById(uid, { delivery: 'split' }); // les deux, sépar
```

Assembly happens **server-side** in all three cases, so a browser client and a desktop one asking for the same delivery receive byte-identical text.

6.5 — Delete, and what delete means

```
await izi.templates.deleteTemplate(uid);
// { success: true, message: 'Template deleted successfully', warnings: [] }
```

That is a move to the trash. `listTrash()` still shows it and `restoreTemplate()` brings it back. The irreversible form is `purgeTemplate()`, and it is a separate method for that reason.

Chapter VII — Runtimes, bytes and slashes

7.1 — Path parameters that accept slashes

Seven routes declare a parameter with FastAPI's `:path` converter — one on collections, six on tags. Those parameters are **allowed to contain** `/` : a nested collection is `Marketing/2026` , a hierarchical tag is `client/acme/urgent` .

The trap is that the OpenAPI schema **erases the converter**, so `/api/tags/{tag_name:path}` appears there as `/api/tags/{tag_name}` , indistinguishable from an ordinary segment. A generator reading the schema alone would emit `encodeURIComponent` and break every nested path. This one reads the converters out of `src/api/` and emits two different helpers :

```
enc(v)      // encodeURIComponent – un segment ordinaire
encPath(v)  // encode chaque segment, GARDE les `/'
```

```
await izi.templates.updateCollection('Marketing/2026', { ... }); // les `/' survivent
```

7.2 — Responses that are not JSON

Several endpoints return markdown, CSV or raw bytes. The client inspects `Content-Type` and returns an `ArrayBuffer` rather than failing to parse something that was never JSON :

```
const guide = await izi.documentation.getGuidePage('syntax') as ArrayBuffer;
const texte = new TextDecoder().decode(guide);
```

Attachment bytes are the same path — and that endpoint is the only place in the system that decodes them.

A `204 No Content` returns `undefined` , not an empty-string parse failure.

7.3 — Caching reads

```
const izi = new IziBuddy(url, { token, cacheTtlMs: 60_000 });
await izi.templates.listAllTemplates(); // 5.6 ms
await izi.templates.listAllTemplates(); // 0.019 ms
izi.cacheStats(); // { entries: 1, hits: 1, misses: 1, ttlMs: 60000 }
```

Same three rules as the Python twin, and they are not options: **GET only** — caching a write would answer without writing; **the key carries the token** — two users do not see the same templates; **errors are never kept** — a one-second outage must not last as long as the TTL.

Off by default: a cache nobody asked for serves yesterday's answer to someone who believes they are reading today.

```
await izi.templates.updateTemplate(uid, { ... });
izi.cacheClear();    // le client ne devine pas ce qu'une écriture invalide
```

`staleOnError: true` serves an expired value when the backend stops answering, rather than throwing. Off by default, because serving stale silently makes an outage look like normal operation.

In a browser this cache is per-page and per-client instance. It is not `localStorage`, and nothing is written to disk.

7.4 — Long lists

```
for await (const p of izi.pages('GET', '/api/templates/list', 'templates', 200)) {
  console.log((p as any).template_id);
}
```

The third argument is the key holding the list, and it is required because it differs between endpoints. Guessing it would be the kind of convenience that works until it silently yields nothing.

7.5 — Browsers, workers and Node

The client uses only `fetch`, `AbortController`, `URLSearchParams` and `TextDecoder` — available in Node 18+, in browsers, in service workers and in edge runtimes. Nothing is imported from `node: .`

In a browser, the backend must allow the origin: CORS is configured server-side, and a request refused there fails before the SDK sees a status, surfacing as `IziBuddyError` with `status === 0`. That zero means *no answer*, which is exactly what a CORS refusal is from the caller's side.

Partie 3

Keeping it true

Chapter VIII — Regenerating after an API change

8.1 — One command, two SDKs

```
python3 src/sdk/generate.py
```

It boots the backend in-process, reads its OpenAPI schema, and rewrites :

```
src/sdk/typescript/src/generated.ts  
src/sdk/python/izibuddy/_generated.py
```

Both, from one reading — which is why the two SDKs never disagree about what the API offers. `src/client.ts` is hand-written and never touched.

Yes, the TypeScript SDK is generated by a Python script. The schema lives in a Python application, and booting it is the only way to obtain the schema the server will actually serve — including the parts a static reading of the source would miss.

8.2 — The two guards

```
node tests/test_sdk_drift.js      # le généré correspond-il encore au backend ?  
npm run typecheck                 # le tout compile-t-il ?
```

The first regenerates into a throwaway directory and compares byte for byte. Red means the API moved and the SDK did not; the failure names the first differing line. It never rewrites anything.

The second is not a formality. It caught a real collision : the backend has a model named `ValidationError` and the client has an error class of the same name, and `export *` from both made the public surface ambiguous. The generator now renames a colliding model to `ApiValidationError`.

8.3 — Two traps worth knowing

A duplicate route makes the generator lie. Two handlers registered on the same method and path are legal Python : FastAPI serves the first, and the schema keeps one entry — possibly named after the handler that never runs. That case exists today on `GET /api/templates/sharing/{template_id}`, declared twice in `src/api/templates.py`. The SDK exposes `getTemplateSharingInfo` ; the backend answers with `get_template_sharing`. The shapes are compatible, so nothing breaks — but the name is wrong, and regenerating will not fix it. The fix is in `src/api/`.

A renamed handler is a breaking change. Renaming a Python function renames a TypeScript method. That is the price of naming methods after their handlers, and it is worth paying: the alternative is a stable name that no longer tells you where the code lives. A diff in `generated.ts` makes the break visible before a consumer discovers it.

Chapter IX — Publishing and consuming

9.1 — Where it stands today

npm does not carry `@izibuddy/sdk` yet, and this book will say so until it does. The claim is a ten-second test; admitting the gap costs less than being disproved.

Everything up to the act of publishing exists. Every release run builds the tarball, **installs it into an empty consumer project** and imports it, before anything reaches a registry — a check worth more than the build, because a package can build and still ship without its `dist/`.

9.2 — Installing today

From a checkout:

```
npm install ./src/sdk/typescript      # après `npm run build`
```

Or against the packed tarball, which is what a consumer will actually receive:

```
npm --prefix src/sdk/typescript pack --pack-destination /tmp  
npm install /tmp/izibuddy-sdk-0.1.0.tgz
```

9.3 — Installing once published

```
npm install @izibuddy/sdk
```

ESM only — `package.json` declares `"type": "module"` and the build emits ES2022 with explicit `.js` specifiers. In a CommonJS project, reach it with a dynamic `import()`.

Nothing else is pulled in: no runtime dependencies, and `typescript` is a `devDependency` of this package only.

9.4 — Public or private — a decision, not a setting

`package.json` currently sets `publishConfig.access` to `restricted`, so the package will be **private**: visible to the `izibuddy` organisation and to nobody else. A public package needs that value and the `--access` flag in the release workflow to both say `public`.

It is a distribution decision. Flipping it because a build asked would be the wrong reason, and it cannot be undone quietly — a package that has been public has been fetched.

9.5 — Publishing, for a maintainer

One tag releases **both** SDKs, since one workflow builds them from one reading of the schema :

```
git tag sdk-v0.1.0 && git push origin sdk-v0.1.0
```

npm is published with `--provenance` , which attaches to the package the verifiable record of the commit and the workflow that produced it — a consumer can check where their `dist/` came from without trusting anyone's word.

The token lives in a `release` environment rather than in the repository's secrets at large, so a required reviewer can stand in front of it. Publishing is irreversible ; a version number is never reusable.

Versions live in two files — `src/sdk/typescript/package.json` and `src/sdk/python/pyproject.toml` — and they must agree before the tag. The tag triggers the release ; it does not set the version.

The full checklist is `src/sdk/RELEASE.md` .

Appendices

Appendix A — The twenty-six resources

Every property of the client, with the backend tag it comes from and the number of operations it carries.

property	backend tag	operations
izi.admin	admin	8
izi.adminAlerts	admin-alerts	7
izi.analytics	analytics	12
izi.api	(untagged — root routes)	5
izi.authentication	authentication	17
izi.config	config	5
izi.diagnostic	diagnostic	3
izi.diff	diff	1
izi.documentation	documentation	6
izi.events	events	4
izi.git	git	12
izi.groups	groups	19
izi.invitations	invitations	9
izi.license	license	5
izi.migration	migration	7
izi.presets	presets	4
izi.templateBuilderConfig	template-builder-config	14
izi.templateStructure	template-structure	4
izi.templates	templates	57
izi.scoring	scoring	5
izi.status	status	4
izi.tags	tags	11

property	backend tag	operations
izi.tagsV2	tags-v2	10
izi.users	users	16
izi.versioning	versioning	5
izi.webhooks	webhooks	2

252 operations. The listing comes from the same schema as the code, and `tests/test_sdk_drift.js` fails if the two disagree.

The `Resources` interface in `generated.ts` states the same list to the compiler, which is what makes editor completion on `izi.` complete.

