

kaiv for TOML Users

A Guide

The kaiv Project

for kaiv 0.5.0

1 Introduction

Of all the formats kaiv converts, TOML is the closest relative — TOML was a design influence on kaiv, and for flat and tabled configuration the two correspond almost line for line. What kaiv adds is everything TOML deliberately leaves out: a schema language, explicit types on every line, tagged unions, constraints, variables, per-value provenance, units — and conversion out to seven other formats from the same canonical form.

Every example is a verified transcript of the reference kaiv tool (`cargo install kaiv-cli`), run against the fixture files shipped beside this document.

2 Hello, kaiv

A TOML file exercising the things TOML is loved for — bare keys, tables, arrays of tables, and all four datetime flavors:

```
title = "kaiv demo"
launched = 2026-07-17T09:30:00Z
local = 2026-07-17T09:30:00
day = 2026-07-17
teatime = 16:00:00

[owner]
name = "Ada"
active = true

[[mirrors]]
host = "eu.example.com"
```

```
weight = 2

[[mirrors]]
host = "us.example.com"
weight = 1
```

Build it to canonical form:

```
$ kaiv import site.toml | kaiv build
.!daiv
!str'::title=kaiv demo
!std/time/datetime'::launched=2026-07-17T09:30:00Z
!std/time/localdatetime'::local=2026-07-17T09:30:00
!std/time/date'::day=2026-07-17
!std/time/time'::teatime=16:00:00
!str'/owner::name=Ada
!bool'/owner::active=true
!str'/@mirrors/0::host=eu.example.com
!int'/@mirrors/0::weight=2
!str'/@mirrors/1::host=us.example.com
!int'/@mirrors/1::weight=1
```

One line per field: `!type ' address=value`. Note the datetimes: TOML is the only mainstream config format with first-class dates, and all four flavors survive — as named types from `kaiv's std/time` library (`datetime`, `localdatetime`, `date`, `time`), each a pattern-refined string with temporal ordering. Nothing collapses to a bare string; nothing gains a timezone it never had.

3 The Same Document, Authored in kaiv

`kaiv's` authoring syntax makes the kinship obvious. Here is that document as a hand-written `.kaiv` file — TOML's `[owner]` becomes the namespace block (`/owner`), and each `[[mirrors]]` becomes an array block `/@mirrors`:

```
!.!kaiv
!.!types std/time

title=kaiv demo
&datetime
launched=2026-07-17T09:30:00Z
```

```

(/owner)
name=Ada
!bool
active=true
()

[/@mirrors]
host=eu.example.com
!int
weight=2
[/@mirrors]
host=us.example.com
!int
weight=1
[]

```

It builds to exactly the canonical form you saw above (minus the three datetime-flavor fields this shorter version omits):

```

$ kaiv build site.kaiv
.!daiv
!str'::title=kaiv demo
!std/time/datetime'::launched=2026-07-17T09:30:00Z
!str'/owner::name=Ada
!bool'/owner::active=true
!str'/@mirrors/0::host=eu.example.com
!int'/@mirrors/0::weight=2
!str'/@mirrors/1::host=us.example.com
!int'/@mirrors/1::weight=1

```

The differences are principled: values are unquoted (no escaping anywhere in kaiv), types are annotations rather than inferred from spelling, and the reuse machinery (&datetime type references, variables) is explicit.

4 The Mapping

TOML	kaiv	
key = "value"	key=value	quoting never needed
[table]	(/table) block	or inline /table::key=
[[array]] tables	[/@name] blocks	one per element, [] closes
dotted keys	namepaths	/a/b::c=
inline table	:= map	/point:=x=1 y=2
array	:= vector	/@tags:=a;b
datetimes (4 flavors)	std/time named types	all four preserved
int / float / bool	!int / !float / !bool	explicit, not inferred
# comment	# comment	same character
multi-line string	!text	: -joined lines
(no null)	!null	kaiv has it; TOML export refuses

5 The Round Trip

Back out to TOML, tables and datetime flavors intact:

```
$ kaiv import site.toml | kaiv build > site.daiv
$ kaiv export --toml site.daiv
title = "kaiv demo"
launched = 2026-07-17T09:30:00Z
local = 2026-07-17T09:30:00
day = 2026-07-17
teatime = 16:00:00

[owner]
name = "Ada"
active = true

[[mirrors]]
host = "eu.example.com"
weight = 2

[[mirrors]]
host = "us.example.com"
weight = 1
```

And the same `site.daiv` exports to JSON, YAML, XML, or CBOR — the canonical form is the hub, TOML one of its spokes.

6 Inline Tables and Multiline Strings

Inline tables become `:=` maps; multiline strings — `kaiv` values are single-line by design — become `!text` lines, the text’s lines joined by the fixed `| : |` separator:

```
$ printf 'point = { x = 1, y = 2 }\ndesc =  
  ↪  ""\ntwo\nlines""\n' | kaiv import --toml  
.!kaiv  
  
!int  
/point:=x=1|y=2  
!text  
desc=two|:|lines
```

One separator, one newline — the value stays readable in place, and exporters resolve `| : |` back to real line breaks. Content the separator cannot carry (a CR, a literal `| : |`) still embeds via the `std/enc` channel, `base64url`-armored. Where multiline prose dominates as laid-out paragraphs, TOML remains the better authoring surface.

7 Wider Integers, Honest Exits

TOML integers are 64-bit. `kaiv` integers are unbounded decimal strings — 2^{64} is just another value:

```
$ printf '!.kaiv\n!int\nbig=18446744073709551616\n' | kaiv  
  ↪  build > big.daiv  
$ cat big.daiv  
!.daiv  
!int'::big=18446744073709551616  
$ kaiv export --toml big.daiv  
kaiv: TOML cannot represent the integer  
  ↪  18446744073709551616 (exceeds i64)
```

The export refuses rather than rounding through a float — fidelity is never silent. The same policy covers `null`: `kaiv` has `!null`, TOML does not, and a document containing one exports to TOML with an error, not a guess.

8 The Schema TOML Never Had

TOML has no schema language at all. `kaiv` infers one from your existing file — datetime flavors included:

```
$ kaiv infer --name acme/site site.toml > site.saiv
$ head -8 site.saiv
.!saiv 1 acme/site
.!types std/time

title=
&datetime
launched=
&localdatetime
local=
```

Now the contract holds. Write the date the European way and the validator names the field, the value, the violated pattern, and the line:

```
$ sed 's/day = 2026-07-17/day = "17.07.2026"/' site.toml >
↪ eu.toml
$ kaiv import eu.toml | kaiv build > eu.daiv
$ kaiv validate eu.daiv site.saiv
kaiv: ConstraintViolationError: ::day=17.07.2026 (type
↪ !str) violates /^d{4}-d{2}-d{2}$/ ..time (line
↪ 5)
```

The constraint shown is `std/time/date`'s compiled form: the ISO pattern plus temporal ordering, exactly as checked. Exit code 1; pass and exit 0 otherwise.

9 When to Stay with TOML

An honest map marks the roads not taken. TOML remains the right choice when:

- your ecosystem reads it natively — the `Cargo.toml` and `pyproject.toml` toolchains are TOML by specification;
- multiline strings must stay laid out as paragraphs (`kaiv`'s `!text` keeps them readable, but on one line);

- the file is small, hand-edited, schema-free configuration and will never need validation, conversion, or provenance.

kaiv’s pull grows with what surrounds the data: a schema inferred from the file you already have, validation errors that name the field, types that do not depend on how a value is spelled, unbounded integers, null — and seven more formats reachable from the same canonical form.

10 Where Next

From here:

- the [kaiv specification](#) — the formal grammar and semantics;
- `kaiv help` — the full toolchain surface;
- the sibling guides — kaiv for JSON, YAML, XML, Protocol Buffers, Avro, CBOR, ASN.1, and GraphQL users.