$ cd reference / Date and time formats

Date and time formats: ISO 8601, RFC 3339, Unix

Every example on this page describes the same instant: 7 October 2026, 09:30 in a zone two hours ahead of UTC, which is 07:30 UTC.

[01] One instant, several notations

FORMAT	EXAMPLE
ISO 8601 / RFC 3339, UTC	`2026-10-07T07:30:00Z`
ISO 8601 / RFC 3339, with offset	`2026-10-07T09:30:00+02:00`
With milliseconds	`2026-10-07T07:30:00.000Z`
ISO 8601 basic format	`20261007T073000Z`
Unix seconds	`1791358200`
Unix milliseconds	`1791358200000`
HTTP date (IMF-fixdate)	`Wed, 07 Oct 2026 07:30:00 GMT`
ISO week date	`2026-W41-3`
ISO ordinal date	`2026-280`

[02] ISO 8601 versus RFC 3339

RFC 3339 is a strict profile of ISO 8601 for timestamps on the internet. It always has a full date, a time and an offset, with `Z` or `±hh:mm`. A note in the RFC permits a space instead of `T` for readability. ISO 8601 also permits week dates, ordinal dates, reduced precision, durations such as `PT90M` and intervals, none of which RFC 3339 accepts.

If you produce timestamps, produce RFC 3339 in UTC with a `Z`. Every mainstream parser reads it, and as long as every value uses `Z` and the same number of fraction digits, it sorts correctly as plain text.

[03] Unix time

Unix time counts seconds since the Unix epoch, which is timestamp 0 at midnight UTC, and pretends leap seconds do not exist. JavaScript, Java and many JSON APIs use milliseconds; most databases, shells and C APIs use seconds. A ten-digit value is seconds and a thirteen-digit value is milliseconds, for current dates.

TASK	COMMAND
Current time in seconds	`date +%s`
Seconds to a readable UTC date (GNU)	`date -u -d @1791358200`
Same on BSD and macOS	`date -u -r 1791358200`
ISO 8601 in UTC	`date -u +%Y-%m-%dT%H:%M:%SZ`
In JavaScript, seconds	`Math.floor(Date.now() / 1000)`

A signed 32-bit counter of seconds overflows after 2147483647. Embedded systems and old file formats that store `time_t` in 32 bits still exist; use 64 bits.

[04] Offsets and time zones

An offset such as `+02:00` fixes one instant. A time zone such as `Europe/Berlin` is a named set of rules from the IANA database. The rules say which offset applies on which date and change when governments decide. Store the instant in UTC. Store the zone name separately when you need to show local time or schedule something "every day at 09:00 local".

Abbreviations like `CST` or `IST` are ambiguous: each stands for several zones on different continents. Never parse them; output them only for people to read.

[05] Parsing traps

INPUT	TRAP
`new Date("2026-10-07")`	date-only ISO is read as UTC midnight, which is the previous evening in zones west of UTC
`new Date("2026-10-07T09:30")`	date-time without offset is read as local time; same string, different instant on another machine
`new Date(2026, 9, 7)`	months are counted from 0, so 9 is October
`new Date("10/07/2026")`	non-ISO formats are implementation-defined; current engines read this as month first, 7 October, whatever the user's locale
`TIMESTAMP '2026-10-07 09:30'`	in PostgreSQL, `timestamp` has no zone; use `timestamptz`, which stores UTC
`2026-10-07T24:00:00`	24:00 as the end of a day is allowed by some editions of ISO 8601, but RFC 3339 limits hours to 00-23 and many parsers reject it; write the next day's 00:00 instead

[06] Daylight saving time

On the night clocks go forward, an hour of local times does not exist; on the night they go back, an hour happens twice. "Add one day" and "add 24 hours" give different results across those nights. Do calendar arithmetic in the local zone with a library that knows the rules, and duration arithmetic in UTC.

[07] Sources

[1] RFC 3339: date and time on the internet

[2] RFC 9110, section 5.6.7: HTTP dates

[3] ECMAScript: the date time string format

[4] PostgreSQL: date and time types

[5] IANA time zone database