Engineer

CLDR

Unicode's data on each locale's dates, numbers and sorting

What it is

CLDR is a body of locale data that works alongside Unicode. Unicode defines the characters a text is written in; CLDR records how readers of a given language in a given region expect a date, a number, a price, a unit or a list to be written, and in what order they expect names to be sorted. The Unicode Consortium maintains both. CLDR's full name is now Unicode Common Locale Data & Rules. It was formerly called the Common Locale Data Repository, and its specification was formerly called the Locale Data Markup Language, or LDML; the specification says that references under the old names remain valid.

The project's site names three parts. The first is the data itself, curated and structured for software to use. The second is the specification, Unicode Technical Standard #35, which documents how the data is structured and how algorithms use it, with requirements for conformance. The third is the code that collects the data from language specialists, checks it for validity and consistency, and converts it into the formats developers use. The specification's own summary of what the data covers makes a good map: the identifiers used for locales, for units, for numbering systems and for calendars; the formatting of dates, times, time zones, numbers, currencies and measurements; and sorting and searching. Plural rules, list patterns and the names of languages and regions sit alongside them.

The data is organised by locale: a language, which can be narrowed by a script and a region. Each locale holds only what differs from its parent. The specification's example is English: `en` holds the bulk of the data, and `en_IE`, English as used in Ireland, holds only a few items such as currency. Above every language sits a locale called root, and the locale for a language's primary region holds no data of its own, because its parent already holds everything it needs. A few facts belong to the region alone, whatever the language: the currency, the

measurement system and the conventions of the week. So if no bundle exists for French as used in the United States, a request for it gets its formats from `fr` and its amounts in US dollars.

The source data is published for each release as XML in the format the specification defines, with tools that convert it to JSON and POSIX formats. Releases follow a schedule of two cycles a year, and CLDR 49 was released on 22 October 2026. New data comes in through an online Survey Tool, which usually opens once a year, both to collect data for newly added structures and to fix data in earlier releases, and native speakers and linguistic experts vet what goes in.

Use the concept in a decision

Put “What is CLDR?” to work in a real decision. We can help relate the concept to the system you are building or buying. Software that reads naturally in every market you sell in, including the dates, the currencies and the way names sort.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Built an English/Danish i18n system — 12,027 messages a locale, in English and Danish

Built an English/Danish i18n system Get in touch →

Why it matters

The project’s site states the problem plainly: writing out a date, a number, a sum of money or a measurement correctly is “far more complicated across different languages and regions than most people recognize.” CLDR exists so that no team has to learn that complexity one language at a time. It supports over 100 languages, and its data is vetted by native speakers and linguistic experts, which is more review than any product team can give the date format of a market it has just entered.

Most teams never touch CLDR directly, and that is by design. The project counts every modern operating system and browser among those that use its data, along with languages such as Java, C#, .NET, Swift and JavaScript. The use is often indirect: an application asks an operating system service to format a date, the service calls ICU, Unicode’s production library for C, C++, Java and Rust, and ICU reads CLDR. Node.js builds its `Intl` objects on ICU and ships the full ICU data by default; Python programs reach the same data through Babel, and Ruby programs through TwitterCLDR. When a team calls a locale-aware formatter, the answer almost always comes from CLDR. What the team still decides is which locale to ask for, and which release it is running.

The alternative is a table written by hand: twelve month names per language, a decimal separator, a thousands separator, an order for the alphabet. At best such a table is right on the day it is written, and it stays frozen from then on, while CLDR's Survey Tool reopens every year to correct data already released. It is also usually written by someone who speaks only some of the languages in it. The orders in which Danish and Swedish sort the same six place names, shown below, are not something a developer outside those languages would guess.

What it looks like in practice

Ask the runtime which CLDR release it carries, then let that data sort six Danish and Swedish place names, join three Danish city names into a list and look up the week of four locales:

```
console.log("CLDR", process.versions.cldr);

const towns = ["Ørsted", "Odense", "Åsa", "Aabenraa", "Ærø", "Ängelholm"];
console.log("code point", [...towns].sort().join(", "));
for (const tag of ["en", "da", "sv"]) {
  const sorted = [...towns].sort(new Intl.Collator(tag).compare);
  console.log(tag.padEnd(10), sorted.join(", "));
}

const three = ["Aarhus", "Odense", "Aalborg"];
for (const tag of ["en-GB", "en-US", "da", "uk"]) {
  console.log(tag.padEnd(10), new Intl.ListFormat(tag).format(three));
}

for (const tag of ["en", "en-GB", "en-DK", "da"]) {
  const locale = new Intl.Locale(tag);
  const { firstDay } = locale.getWeekInfo();
  console.log(tag.padEnd(10), locale.maximize().toString(), firstDay);
}
```

Node 25.2.1 prints:

```
CLDR 47.0
code point Aabenraa, Odense, Ängelholm, Åsa, Ærø, Ørsted
en         Aabenraa, Ærø, Ängelholm, Åsa, Odense, Ørsted
da         Odense, Ängelholm, Ærø, Ørsted, Aabenraa, Åsa
sv         Aabenraa, Odense, Åsa, Ängelholm, Ærø, Ørsted
en-GB      Aarhus, Odense and Aalborg
en-US      Aarhus, Odense, and Aalborg
da         Aarhus, Odense og Aalborg
```

```
uk          Aarhus, Odense i Aalborg
en          en-Latn-US 7
en-GB       en-Latn-GB 1
en-DK       en-Latn-DK 1
da          da-Latn-DK 1
```

The first line is the release this runtime was built with. It is 47.0, although CLDR 49 is out: a runtime carries the data of its own build, not the newest data there is.

A plain `sort()` with no comparator puts Ä, Å, Æ and Ø after every unmarked letter, in the order of their character codes, which matches none of the three languages. English sorts each name by its base letter, so the four names that begin with an A come first and Ørsted follows Odense. Danish ends its alphabet with Æ, Ø and Å, treats Ä as a variant of Æ, and sorts Aabenraa, spelled with the older double a, among the Å names; the collation part of the specification calls this the Danish contraction “aa”. Swedish ends its alphabet with Å, Ä and Ö, treats Æ as a variant of Ä and Ø as a variant of Ö, and sorts Aabenraa as a name that starts with two a’s. Norwegian Bokmål, `nb`, returns the Danish order. None of this is in the code; each order comes from CLDR’s collation rules for that language.

The lists work the same way. Each locale has a list pattern with separate pieces for the start, the middle and the end of a list, and for a list of two. British English puts no comma before “and” and American English does, Danish joins with “og” and Ukrainian with “i”, from one call that contains no word of any of these languages.

The last four lines show data that belongs to a region rather than a language. Each gives the tag, the full form that `maximize()` fills in from CLDR’s likely subtags, and the first day of the week as an ISO 8601 number, where 1 is Monday and 7 is Sunday. The runtime reads a bare `en` as English in the Latin script as used in the United States, so its week starts on Sunday. English as used in Britain or in Denmark starts the week on Monday, as Danish does: the week comes from the region, whatever the language.

The conventions CLDR records are also the ones a translated text has to get right by hand. One of this site’s projects, a cross-language figure check, extracts every number, percentage, multiplier and unit from an English string and requires each

of them in every translation. It also checks each language against the conventions it must use and must not, including decimal separators, thousands grouping and quotation marks.

Where it goes wrong

Formatted text is compared as a fixed string. A test that expects `"1 500"` from `new Intl.NumberFormat("sv").format(1500)` fails although the output looks exactly like that, because Swedish, Norwegian and Ukrainian separate thousands with U+00A0 NO-BREAK SPACE, which an editor shows as an ordinary space. The same test can also break on an upgrade, because a new CLDR release can correct data that an older one shipped. Compare formatted output with the output of the same formatter, check its structure with `formatToParts`, which labels each piece as integer, group, decimal or fraction, and never parse formatted text back into a number. The specification's advice is to store and exchange language-neutral data and format it for the reader.

Two runtimes carry two releases. Node 25.2.1 carries CLDR 47.0 while the project has released 49, and a browser carries whatever its own build shipped. A page that formats a date on the server and again in the browser can show two strings for one value, and a snapshot recorded on one machine can fail on another. Format each value in one place, record `process.versions.cldr` beside the test snapshots, and upgrade the runtime deliberately.

A missing locale falls back without an error. `new Intl.NumberFormat("tlh")` formats as American English and throws nothing, because the runtime has no data for Klingon and falls back to its default locale. Node's documentation describes a quieter case: a binary built with the small-icu option typically carries only English data, and there a Spanish month name prints "M01" or "January" instead of "enero". Check `resolvedOptions().locale` or `supportedLocalesOf` before trusting a locale, and run builds with the full ICU data, which is Node's default and what its official binaries use.

A tag without a region gets the likeliest one. The runtime reads `en` as `en-Latn-US`, so a reader in Ireland, Britain or Denmark who is served `en` gets a week that starts on Sunday and dates written as 10/27/2026, month first. The specification describes the likely-subtags data behind this as heuristically derived and says it may change over time. Pass the reader's region along with the language whenever it is known, and where only a language is known, call `maximize()` to see which region the runtime will assume.

CLDR's identifiers are passed to a BCP 47 API. CLDR's files and examples write locales with underscores, such as `en_IE`, and the specification treats `_` and `-` as equivalent. JavaScript's `Intl` does not: `new Intl.NumberFormat("da_DK")` throws `RangeError: Invalid language tag: da_DK`. Convert at the boundary as the specification's own conversion to BCP 47 does, by replacing each `_` with `-`, and pass the result through `Intl.getCanonicalLocales`, which returns `da-DK`.

Further reading

- Unicode CLDR Project — what CLDR is, its three components, who uses it and how, and the release schedule.
- UTS #35, Part 1: Core — the specification: locale identifiers, the former names, inheritance between locales, likely subtags and the advice to keep data language-neutral.
- UTS #35, Part 2: General — display names, measurement systems and the list patterns used above.
- UTS #35, Part 5: Collation — how CLDR tailors the default sort order for each language, with Danish as an example.
- Node.js: Internationalization support — how Node builds its `Intl` objects on ICU, the build options and how to detect what a binary supports.
- ECMAScript Internationalization API Specification — the standard behind `Intl`, including the `Locale` methods used above and the CLDR week data and likely subtags they rely on.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is CLDR?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/cldr/