Engineer

Localization

adapting software to one market's language and conventions

What it is

Localization is the work of adapting a product to one particular market: its language, and with it the conventions its readers expect for dates, numbers, prices, names, addresses and the order of a list. The W3C defines it as adapting a product, an application or the content of a document to one target market, so that it suits that market's language and culture and meets whatever else the market needs, and it calls that market a locale. The word is also spelled localisation, and it is shortened to l10n, for the 10 letters between its first l and its last n.

The W3C adds that localization is often thought of as a synonym for translating the interface and the documentation, and that it is often a substantially more complex job. Its list of what may need adapting covers numeric, date and time formats; currency; keyboard layout; collation and sorting; symbols, icons and colours; text and graphics that a culture may misread or find insensitive; legal requirements; the formats of addresses, postal codes, phone numbers and the order of names; and the readability needs of each script. Where the way of doing business differs, with accounting as the W3C's example, the adaptation may mean rethinking the logic, the visual design or the presentation of the product.

Localization depends on internationalization and comes after it. Internationalization is done once, in the code, so that nothing about a single market is built into it. Localization is done for each locale, in data the code reads: the translated messages, and the conventions that turn a stored date or amount into the text a reader expects. A product that has been internationalized well can gain a locale without a change to its code; one that has not must be changed again for every market it enters.

Most of the convention data does not have to be written by the team. The Unicode Consortium's CLDR project collects it for over 100 languages, checked by specialists and people who speak each language natively, and its own example is the list of files on a phone: the format of the dates, the numbers, the units of the file sizes and the alphabetical order all vary with the language and are usually supplied by CLDR. The project states that its data is built into all modern operating systems and browsers and into languages such as Java, C#, Swift and JavaScript. A product gets most of its formats from the platform, provided the code asks the platform for them instead of writing them itself.

Use the concept in a decision

Put "What is localization?" 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

A product that is translated but not localized reads as foreign. Every label is in Danish, and the date still puts the month first, the price still carries a dollar sign and the list of towns still files Aarhus under A. Readers notice each of these at once, and each one tells them the product was made for somebody else.

Some of the mismatches change what the reader understands. The string 10/11/2026 is 11 October to an American reader and 10 November to a British one; both are valid dates, so nothing fails, and one of the two readers acts on the wrong day. The string 1,234 is one thousand two hundred and thirty-four in English and one point two three four in Danish, a factor of a thousand in a quantity or a price. A wrong translation looks wrong. A wrong convention looks right and is read wrongly.

The W3C's list also reaches past the screen. Legal requirements differ between markets, and so does the way a business is run, so a locale can need different terms, different invoice details or a different flow through a purchase. These are decisions for the business, and they belong in the plan for a locale before the first message is translated.

The cost is per locale and it does not end. Every new message has to be translated into every language, every release has to be read in every locale before it ships, and every market brings its own conventions to check. A team that adds a language adds that work to each later release, which is why the decision to enter a market is also a decision about the release process.

What it looks like in practice

The platform does most of the work when the code passes it a value and a locale. In JavaScript, the `Intl` formatters take both and return text:

```
const when = new Date("2026-10-27T12:00:00Z");
const money = { style: "currency", currency: "DKK" };
for (const locale of ["en-US", "en-GB", "da", "nb", "sv", "uk"]) {
  const date = new Intl.DateTimeFormat(locale, { dateStyle: "short" });
  const price = new Intl.NumberFormat(locale, money);
  console.log(locale, date.format(when), price.format(800));
}
```

Run in Node.js 25, which carries data from CLDR 47, it prints one date and one price of 800 Danish kroner six ways:

```
en-US 10/27/26 DKK 800.00
en-GB 27/10/2026 DKK 800.00
da 27.10.2026 800,00 kr.
nb 27.10.2026 800,00 DKK
sv 2026-10-27 800,00 Dkr
uk 27.10.26 800,00 DKK
```

The code chose none of it. The Danish reader sees kr., the Swedish reader Dkr, because kr in Sweden means the Swedish krona, and the Norwegian and Ukrainian readers see the currency code. The order of day and month, the separators, the decimal comma and the place of the currency all come from the locale data. In the real output the space between the amount and its currency is a no-break space, U+00A0, which keeps the two on one line.

Sorting works the same way. A list of six Danish place names, sorted for three locales:

```
const towns = ["Aabenraa", "Ærø", "Åkirkeby", "Odense", "Østerbro", "Vejle"];
for (const locale of ["en", "da", "sv"]) {
  const order = new Intl.Collator(locale).compare;
  console.log(locale, towns.toSorted(order).join(", "));
}
```

```
en Aabenraa, Ærø, Åkirkeby, Odense, Østerbro, Vejle
da Odense, Vejle, Ærø, Østerbro, Aabenraa, Åkirkeby
sv Aabenraa, Odense, Vejle, Åkirkeby, Ærø, Østerbro
```

English treats æ, ø and å as forms of a and o. Danish places Æ, Ø and Å after Z and files the old spelling Aa as Å, so Aabenraa moves from first to next to last. Swedish puts its own Å, Ä and Ö after Z and files Æ with Ä and Ø with Ö. Each order is correct for its readers, and a Danish reader who looks for Aabenraa near the end of a list sorted for English has reason to think it is missing.

A Hugo site has the same tools in its templates. Hugo’s documentation passes one amount through `lang.FormatCurrency` :

```
{{ 512.5032 | lang.FormatCurrency 2 "USD" }}
```

An English page shows it as $512.50, a French one as 512,50 $US and a German one as 512,50 $, and `lang.FormatNumber` and `lang.FormatPercent` do the same for plain numbers and percentages.

Localization is part of one of this site’s services. The internationalization and localization service describes the aim as software that reads naturally in every market a client sells in, including the dates, the currencies and the way names sort. Among its results is a cross-language content check that fails when a translation drops a figure the English states, or when a language uses notation it does not use, and that found two Danish descriptions missing a metric and sixteen Ukrainian spans quoting in the English style.

Where it goes wrong

Localization is budgeted as translation. The plan for a new market covers the message files and nothing else, so the product launches in the new language with the old market's dates, prices, addresses and sort order, and the gaps are found by customers. Plan each locale against the W3C's whole list, decide which items apply to that market, and have a native reader review the built product rather than the message files.

A language is taken for a locale. `en` does not say whether 27 October is `10/27/26` or `27/10/2026`, and the output above shows that the two English locales disagree on both the order and the year. A product that stores only the language must guess the rest, and it guesses one market's conventions for every other. Keep a full tag such as `en-GB` or `da-DK` for each user, take it from an explicit setting where there is one, and pass it to every formatter.

The currency is guessed from the locale. CLDR's specification is blunt about it: an amount should never be passed between systems without a known currency code, or the same number 3.5 may end up "interpreted as $3.50 by one user and €3.50 by another." It adds that the currency of an amount is independent of the reader's language and that only badly designed software leaves the code out and infers it from the locale. In Scandinavia the guess goes wrong quietly, because Danish, Norwegian and Swedish each abbreviate their own currency kr, so a price stored as 800 reads as 800 Danish kroner to one reader and 800 Norwegian kroner to the next. Store every amount with its ISO 4217 code and format it for the reader; a Danish reader then sees a Norwegian price as 800,00 NOK.

Lists are sorted by code point. JavaScript's default `sort` compares UTF-16 code units, which puts Є and І before the rest of the Ukrainian alphabet and Ґ after Я, because their code points lie outside the range that runs from А to я. Sorted that way, `["Ялта", "Ґанок", "Гора", "Єнот", "Іван", "Дуб"]` comes out as Єнот, Іван, Гора, Дуб, Ялта, Ґанок; with `new Intl.Collator("uk").compare` it is Гора, Ґанок, Дуб, Єнот, Іван, Ялта, the order of the Ukrainian alphabet. Sort text for display with a collator for the reader's locale, in the code or in the database's collation, and keep code-point order for keys that no reader sees.

Tests compare formatted text typed by hand. A test that expects `800,00 kr.` with an ordinary space fails against the formatter, which puts U+00A0 there, and the same no-break space separates the thousands in Norwegian, Swedish and Ukrainian. Locale data is also revised from one CLDR release to the next, so a

string that matched before a platform upgrade may not match after it. Build the expected text with the same formatter and locale the code uses, or assert on the stored value and the locale passed in rather than on the rendered string.

The layout is sized for English. A button that fits Save has to fit Зберегти, twice as many letters, and a menu that fits Settings has to fit the Danish Indstillinger. Fixed widths cut the longer word off or push it onto a second line, and often the first to see it is a reader in that market. Let containers grow with their text, avoid fixed widths around translated words, and review every locale at the narrowest screen the product supports.

Further reading

- Localization vs. Internationalization — the W3C's working definition of localization, its list of what a locale may need adapted, and how localization relates to internationalization.
- Hugo: Multilingual mode — localized dates, currencies, numbers and percentages in Hugo templates, alongside translation tables and per-language URLs.
- UTS #35, Part 3: Numbers — CLDR's number and currency formats, including the rule that an amount always travels with its currency code.
- Unicode CLDR Project — the project that collects locale data for over 100 languages and supplies it to operating systems, browsers and programming languages.

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 localization?” 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/localization/