Engineer

Internationalization

designing software so that it can easily be localized

What it is

Internationalization is the engineering that lets one product work in many languages and regions without its code being changed for each of them. The W3C defines it as designing and building a product, an application or a document so that localizing it later is easy, whatever the culture, region or language of the audience. The word is also spelled internationalisation, and it is shortened to i18n, because 18 letters stand between its first i and its last n. Some people call the same idea globalization.

The W3C puts the division of labour in one sentence: the people who design and build products internationalize them “so that they can be easily localized by translators and localizers.” Internationalization is done once, in the code and the templates. Localization is done again for every market, in data that the code reads. A product can be fully internationalized and still ship in a single language.

The unit both of them work with is the locale. CLDR’s specification defines a locale as an identifier for the preferences that users across large parts of the world tend to have in common: a language, often narrowed by a region, so that `en-GB` and `en-US` share a language and differ in how they write a date. The identifier selects a translation and a set of conventions at the same time.

The W3C lists four kinds of work under the heading. The first removes barriers: the product handles text as Unicode, or treats an older character encoding deliberately where it must; it does not build sentences by joining fragments; and its code never depends on the wording shown to the user. The second adds support that only a later language will use, such as markup for text that mixes right-to-left and left-to-right scripts, a way to declare which language a passage is in, and style rules for vertical text. The third lets the code use local conventions, usually from existing libraries rather than from tables the team writes itself: date

and time formats, calendars, number formats and numeral systems, the order in which lists sort, and the conventions for writing people's names and for addressing them. The fourth keeps everything translatable out of the source code, so that the right version can be loaded for each reader.

Use the concept in a decision

Put "What is internationalization?" 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 W3C's argument is about timing. Reworking a product built around one language and one culture for a global market takes far more effort and far more time than designing it for the whole world from the start, and the W3C compares the work to the Y2K effort to undo two-digit year fields that assumed every year began with 19. Done late, internationalization tends to mean awkward and expensive re-engineering. Done early, it is a handful of habits that cost little on the day they are adopted.

The cost of a missing habit grows with every language. A sentence assembled in code from three fragments is one defect in an English product and five defects in a product with five locales, and in each of them the fix is not a better translation but a change to the code, reviewed and released like any other. The same holds for a date formatted by hand, a list sorted by comparing bytes, or a label that a test finds by its English text.

The habits pay even when nothing is ever translated, a point the W3C makes explicitly. A product sold only in English still receives names written with letters English does not use, addresses laid out the way another country lays them out, and dates from systems that write the day before the month. Text that is stored as Unicode, and data that keeps a date as a date until the moment it is shown, handle all of that without a special case.

What it looks like in practice

The quickest test of whether a product is internationalized is a sentence with a number in it. The version that is not builds the sentence in code:

```
const label = count + (count === 1 ? " day" : " days");
```

That is correct in English and cannot be translated. Ukrainian needs three forms for whole numbers and a fourth for fractions: день after 1, 21 and 101, дні after 2, 3, 4 and 22, днів after 0, 5 to 20 and 25, and дня after a fraction. Whatever single word a translator puts in the second branch, it is wrong for most numbers, because 21 takes the same form as 1 and 2 takes a different form from 5.

Internationalized, the sentence becomes a key, and each language's table supplies every form the language needs. In Hugo, translation tables live in the `i18n` directory, one file per language, named by its language tag:

```
# i18n/en.toml
[days]
one = '{{ . }} day'
other = '{{ . }} days'
```

```
# i18n/uk.toml
[days]
one = '{{ . }} день'
few = '{{ . }} дні'
many = '{{ . }} днів'
other = '{{ . }} дня'
```

The template asks for the key and passes the number. It contains no word of either language and no rule about plurals:

```
{{ T "days" 21 }}
```

The English pages render `21 days` and the Ukrainian pages `21 день`; with 2 they render `2 days` and `2 дні`, and with 5, `5 days` and `5 днів`. The names `one`, `few`, `many` and `other` are not Hugo's invention. They are the plural categories defined by CLDR, the Unicode Consortium's collection of locale data, and Hugo's documentation points to CLDR's plural rules chart for the categories each language has. A new language arrives as a new file, with no change to a template.

At scale the tables become the largest body of text in the product, and they need the same discipline as code. One of this site's projects, an English and Danish internationalization system, carries 12,027 messages per locale in 458 namespace files. Each file is held to a budget of 400 lines, a linter compares the keys across the two locales so that no message is present in one and missing in the other, and the Danish side follows a locked vocabulary, down to the informal "du", that the tooling enforces on every commit. The page credits the budget with keeping the work maintainable: without a limit per file, it says, the 458 files would have been twelve.

Where it goes wrong

Sentences are glued together. A message built as `"Delete " + n + " files?"` reaches the translator as two fragments with no sentence around them, in an order that only English needs. Another language may want the number at the end, or a verb form that depends on the noun. The W3C names care over the concatenation of strings as part of the work. Make every sentence one message with named placeholders, which a translator can move; Hugo's documentation shows `{{ T "age" (dict "name" "Will" "count" 1) }}` filling both a name and a count inside one translated sentence.

The code reads the words on the screen. A test that finds a button by its label "Save", or a branch that compares a status with the text "Paid", works until the first translation and then fails without an error. The W3C lists avoiding any dependence in code on user-interface strings among the first things internationalization means. Compare keys, identifiers and enumerated values, and give tests a stable attribute to find an element by.

One plural rule for every language. `count === 1` is the English rule written as code, and it is wrong for most of the languages a product may add. CLDR's specification defines six plural categories, zero, one, two, few, many and other, of

which only other is mandatory, and gives each language its own rules for choosing among them: English uses two, Ukrainian four. Pass the number to the message and let the translation table carry every category the language uses.

A missing translation fails without a sound. Hugo's `T` falls back to the default language when a key is missing in the current one, and returns an empty string when the default has none either, so a page can ship with a gap or an English phrase in the middle of a Danish one. Build with `--printI18nWarnings` to list every missing and fallback translation, turn on `enableMissingTranslationPlaceholders` in a preview environment so that gaps show on the page, and compare the keys of every locale in continuous integration.

Dates and numbers are formatted by hand. A template that writes `MM/DD/YYYY`, or a number with a comma every three digits, has hard-coded one country. The same day is `10/27/26` in American English, `27/10/2026` in British English, `27.10.2026` in Danish and `2026-10-27` in Swedish; 12,027.5 is written `12.027,5` in Danish, and in Norwegian, Swedish and Ukrainian with a space as the thousands separator and a comma before the decimals. CLDR's specification recommends storing and exchanging language-neutral data and formatting it for the reader. Keep dates and amounts as plain values until the last step, and format them there with the locale-aware functions the platform provides. In Hugo, `time.Format ":date_full"` renders one date as `Wednesday, November 3, 2021` on an English page and `Mittwoch, 3. November 2021` on a German one.

Text is not stored as Unicode. A product that writes "blåbærsyltetøj" in UTF-8 and reads it back as Windows-1252 shows "blÃ¥bÃ¦rsyltetÃ¸j", and one that stores text in an older single-language encoding cannot hold a Ukrainian name at all. Enabling Unicode is the first item on the W3C's list. Use one Unicode encoding, normally UTF-8, from the database to the browser, and declare it at every boundary where bytes become text.

Further reading

- Localization vs. Internationalization — the W3C's working definitions of both terms, the four kinds of work internationalization involves, and why it belongs at the start.
- Hugo: Multilingual mode — translation tables, localized dates and currencies, and how Hugo reports a missing translation.
- Hugo: lang.Translate — the lookup order across languages, plural categories in translation tables, and messages with named placeholders.

- UTS #35, Part 1: Core — CLDR's specification: locale identifiers, inheritance between locales and the advice to keep data language-neutral until it is shown.
- UTS #35, Part 3: Numbers — number symbols and patterns per locale, and the plural categories with the rules for choosing among them.

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 internationalization?" 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/internationalization/