Engineer

CLDR

Unicodes data om hvert sprogs datoer, tal og sortering

Hvad det er

CLDR er en samling data om sprog og regioner, der supplerer Unicode. Unicode definerer de tegn, en tekst skrives med; CLDR beskriver, hvordan læsere af et bestemt sprog i en bestemt region forventer at se en dato, et tal, en pris, en enhed eller en liste skrevet, og i hvilken rækkefølge de forventer, at navne sorteres. Unicode-konsortiet vedligeholder begge dele. CLDR's fulde navn er i dag Unicode Common Locale Data & Rules. Tidligere hed det Common Locale Data Repository, og specifikationen hed Locale Data Markup Language, forkortet LDML; ifølge specifikationen er henvisninger under de gamle navne stadig gyldige.

Projektets websted nævner tre dele. Den første er selve dataene, udvalgt og struktureret, så software kan bruge dem. Den anden er specifikationen, Unicode Technical Standard #35, der beskriver, hvordan dataene er opbygget, og hvordan algoritmer bruger dem, med krav til overensstemmelse. Den tredje er koden, der indsamler dataene fra sprogspecialister, kontrollerer, at de er gyldige og konsistente, og konverterer dem til de formater, udviklere bruger. Specifikationens eget resumé af, hvad dataene dækker, er et godt kort: identifikatorer for locales, måleenheder, talsystemer og kalendere; formatering af datoer, klokkeslæt, tidszoner, tal, valutaer og måleværdier; og sortering og søgning. Flertalsregler, listemønstre og navne på sprog og regioner hører med.

Dataene er ordnet efter locale: et sprog, der kan afgrænses yderligere med et skriftsystem og en region. Hver locale rummer kun det, der afviger fra dens forælder. Specifikationens eksempel er engelsk: `en` rummer hovedparten af dataene, og `en_IE`, engelsk som det bruges i Irland, rummer kun nogle få ting, for eksempel valutaen. Øverst over alle sprog ligger en locale ved navn root, og localen for et sprogs primære region har ingen egne data, fordi dens forælder allerede rummer alt, hvad den skal bruge. Nogle få oplysninger hører alene til regionen, uanset sproget: valutaen, målesystemet og konventionerne for ugen.

Har CLDR ingen særskilte data for fransk, som det bruges i USA, får man derfor formaterne fra `fr` og beløbene i amerikanske dollar, når man beder om netop den kombination.

Kildedataene udgives for hver version som XML i det format, specifikationen definerer, med værktøjer, der konverterer dem til JSON- og POSIX-formater. Udgivelserne følger en plan med to cyklusser om året, og CLDR 49 udkom den 22. oktober 2026. Nye data kommer ind gennem et onlineværktøj, Survey Tool, der som regel er åbent én gang om året for at indsamle data til nye strukturer og rette data, der allerede er udgivet, og folk med sproget som modersmål og sprogeksperter gennemgår det, der kommer ind.

Brug begrebet i en beslutning

Brug »Hvad er CLDR?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Software, der læses naturligt på hvert marked, I sælger på — også datoer, valutaer og den måde, navne sorteres på.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har bygget et engelsk/dansk i18n-system — 12,027 beskeder pr. sprog, på engelsk og dansk

Har bygget et engelsk/dansk i18n-system Kontakt os →

Hvorfor det betyder noget

Projektets websted formulerer problemet ligeud: at formatere datoer, tal, valutaer og måleenheder til forskellige sprog og regioner er langt mere kompliceret, end de fleste er klar over. CLDR findes, for at intet team skal lære den kompleksitet at kende ét sprog ad gangen. Projektet dækker over 100 sprog, og dataene gennemgås af folk med sproget som modersmål og af sprogeksperter, hvilket er mere gennemsyn, end noget produktteam kan give datoformatet på et marked, det lige er gået ind på.

De fleste teams rører aldrig CLDR direkte, og det er med vilje. Projektet regner alle moderne operativsystemer og browsere blandt dem, der bruger dataene, sammen med sprog som Java, C#, .NET, Swift og JavaScript. Brugen er ofte indirekte: en applikation beder en tjeneste i operativsystemet om at formatere en dato, tjenesten kalder ICU, Unicodes produktionsbibliotek til C, C++, Java og Rust, og ICU læser CLDR. Node.js bygger sine `Intl`-objekter på ICU og leverer som standard de fulde ICU-data; Python-programmer når de samme data gennem Babel, og Ruby-programmer gennem TwitterCLDR. Når et team kalder

en formateringsfunktion, der tager hensyn til locale, kommer svaret næsten altid fra CLDR. Det, teamet stadig selv bestemmer, er, hvilken locale det beder om, og hvilken version det kører.

Alternativet er en tabel skrevet i hånden: tolv månedsnavne pr. sprog, et decimaltegn, et tusindtalsskilletegn, en rækkefølge for alfabetet. I bedste fald er sådan en tabel rigtig den dag, den bliver skrevet, og derefter står den stille, mens CLDR's Survey Tool åbner hvert år for at rette data, der allerede er udgivet. Den er som regel også skrevet af en, der kun taler nogle af sprogene i den. Hvordan svensk sorterer de seks stednavne nedenfor, ville en dansk udvikler ikke gætte, og omvendt.

Sådan ser det ud i praksis

Spørg runtimen, hvilken CLDR-version den har med, og lad så dataene sortere seks danske og svenske stednavne, sætte tre danske bynavne sammen til en liste og slå ugen op for fire 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 udskriver:

```
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
```

Første linje er den version, runtimen er bygget med. Den er 47.0, selv om CLDR 49 er udkommet: en runtime har dataene fra sit eget build med, ikke de nyeste data, der findes.

En almindelig `sort()` uden sammenligningsfunktion lægger Ä, Å, Æ og Ø efter alle bogstaverne fra A til Z, i rækkefølge efter deres tegnkoder, og det passer ikke til nogen af de tre sprog. Engelsk sorterer hvert navn efter dets grundbogstav, så de fire navne, der begynder med et A, kommer først, og Ørsted kommer efter Odense. Dansk slutter alfabetet med Æ, Ø og Å, behandler Ä som en variant af Æ og sorterer Aabenraa, stavet med det gamle dobbelte a, blandt Å-navnene; specifikationens del om sortering kalder det den danske kontraktion »aa«. Svensk slutter alfabetet med Å, Ä og Ö, behandler Æ som en variant af Ä og Ø som en variant af Ö og sorterer Aabenraa som et navn, der begynder med to a'er. Norsk bokmål, `nb`, giver den danske rækkefølge. Intet af det står i koden; hver rækkefølge kommer fra CLDR's sorteringsregler for det pågældende sprog.

Listerne fungerer på samme måde. Hver locale har et listemønster med særskilte dele til begyndelsen, midten og slutningen af en liste og til en liste på to. Britisk engelsk sætter intet komma før »and«, og amerikansk engelsk gør, dansk binder sammen med »og« og ukrainsk med »i«, alt sammen fra ét kald, der ikke indeholder et eneste ord på nogen af sprogene.

De sidste fire linjer viser data, der hører til en region og ikke til et sprog. Hver linje giver sprogkoden, den fulde form, som `maximize()` udfylder med CLDR's mest sandsynlige skriftsystem og region, og ugens første dag som et tal efter ISO 8601, hvor 1 er mandag og 7 er søndag. Runtimen læser et `en` uden region som engelsk med latinske bogstaver, som det bruges i USA, så ugen starter om søndagen. Engelsk, som det bruges i Storbritannien eller i Danmark, starter ugen om mandagen ligesom dansk: ugen kommer fra regionen, uanset sproget.

De konventioner, CLDR registrerer, er også dem, en oversat tekst skal ramme rigtigt i hånden. Et af projekterne på dette site, et tværsprogligt tjek af tal, udtrækker hvert tal, hver procentdel, hver multiplikator og hver enhed fra en engelsk streng og kræver hver af dem i hver oversættelse. Det kontrollerer også hvert sprog mod de konventioner, det skal bruge, og dem, det ikke må bruge, herunder decimalskilletegn, tusindgruppering og anførselstegn.

Hvor det går galt

Formateret tekst sammenlignes som en fast streng. En test, der forventer `"1 500"` fra `new Intl.NumberFormat("sv").format(1500)`, fejler, selv om resultatet ser præcis sådan ud, fordi svensk, norsk og ukrainsk adskiller tusinder med U+00A0 NO-BREAK SPACE, som en editor viser som et almindeligt mellemrum. Den samme test kan også gå i stykker ved en opgradering, fordi en ny CLDR-version kan rette data, som en ældre leverede. Sammenlign formateret output med output fra den samme formateringsfunktion, kontrollér strukturen med `formatToParts`, der mærker hver del som integer, group, decimal eller fraction, og pars aldrig formateret tekst tilbage til et tal. Specifikationens råd er at gemme og udveksle sprogneutrale data og først formatere dem til læseren.

Server og browser har hver sin version. Node 25.2.1 leverer CLDR 47.0, mens projektet er nået til 49, og en browser har den version med, som dens eget build fik. En side, der formaterer en dato på serveren og igen i browseren, kan vise to forskellige strenge for én værdi, og et snapshot, der er optaget på én maskine, kan fejle på en anden. Formatér hver værdi ét sted, gem `process.versions.cldr` sammen med testenes snapshots, og opgradér runtimen bevidst.

En manglende locale falder tilbage uden en fejl. `new Intl.NumberFormat("tlh")` formaterer som amerikansk engelsk uden at kaste en fejl, fordi runtimen ikke har data for klingon og falder tilbage til sin standardlocale. Dokumentationen til Node.js beskriver et mere lumsk tilfælde: et binært build med small-icu-indstillingen har typisk kun engelske data, og dér udskrives et spansk månedsnavn som »M01« eller »January« i stedet for »enero«. Kontrollér `resolvedOptions().locale` eller `supportedLocalesOf`, før du stoler på en locale, og byg med de fulde ICU-data, som er standard i Node og det, de officielle binære udgaver bruger.

En sprogkode uden region får den mest sandsynlige. Runtimen læser `en` som `en-Latn-US`, så en læser i Irland, Storbritannien eller Danmark, der får serveret `en`, får en uge, der starter om søndagen, og datoer skrevet som 10/27/2026

med måneden først. Dataene bag, som specifikationen kalder likely subtags, er efter dens egen beskrivelse heuristisk udledte og kan ændre sig over tid. Send læserens region med sammen med sproget, når den er kendt, og kald `maximize()`, når kun sproget er kendt, så du kan se, hvilken region runtimen vil gå ud fra.

CLDR's identifikatorer sendes til et BCP 47-API. CLDR's filer og eksempler skriver locales med understregning, for eksempel `en_IE`, og specifikationen behandler `_` og `-` som ligeværdige. Det gør JavaScripts `Intl` ikke: `new Intl.NumberFormat("da_DK")` kaster `RangeError: Invalid language tag: da_DK`. Konvertér ved grænsen, sådan som specifikationens egen konvertering til BCP 47 gør, ved at erstatte hver `_` med `-`, og send resultatet gennem `Intl.getCanonicalLocales`, der returnerer `da-DK`.

Læs mere

- Unicode CLDR Project — hvad CLDR er, de tre dele, hvem der bruger det og hvordan, og udgivelsesplanen.
- UTS #35, Part 1: Core — specifikationen: locale-identifikatorer, de tidligere navne, nedarvning mellem locales, likely subtags og rådet om at holde data sprogneutrale.
- UTS #35, Part 2: General — visningsnavne, målesystemer og de listemønstre, der bruges ovenfor.
- UTS #35, Part 5: Collation — hvordan CLDR tilpasser standardsorteringen til hvert sprog, med dansk som eksempel.
- Node.js: Internationalization support — hvordan Node bygger sine `Intl`-objekter på ICU, byggeindstillingerne, og hvordan man finder ud af, hvad et binært build understøtter.
- ECMAScript Internationalization API Specification — standarden bag `Intl`, herunder de `Locale`-metoder, der bruges ovenfor, og CLDR's ugedata og likely subtags, som de bygger på.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er CLDR?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/cldr/