Engineer

Unicode

⌨ standarden, der giver hvert tegn sit eget nummer

Hvad det er

Unicode er standarden, der giver hvert bogstav, hvert ciffer, hvert tegnsætningstegn og hvert symbol i verdens skriftsystemer sit eget nummer. Ifølge dem, der vedligeholder standarden, er Unicode den universelle tegnkodning: bygget, så tekst på den moderne verdens sprog og i dens tekniske fag kan udveksles, behandles og vises overalt, og udvidet til de klassiske og historiske tekster i mange skriftsprog. Ét og samme repertoire rummer det danske å, det ukrainske ґ, eurotegnet og bogstaverne i skrifter, som ingen har skrevet i århundreder, så et enkelt dokument eller en enkelt kolonne i en database kan indeholde dem alle.

Formelt består en version af Unicode-standarden af kernespecifikationen, kodeoversigterne, standardens bilag, Unicode Standard Annexes, og Unicode Character Database, de maskinlæsbare filer, der beskriver hvert tegns egenskaber. Unicode Technical Committee vedligeholder standarden løbende, Unicode-konsortiet udgiver den, og version 18.0 blev annonceret den 16. september 2026.

Det nummer, et tegn får, kaldes dets kodepunkt. Koderummet går fra 0 til 10FFFF hexadecimalt, i alt 1.114.112 kodepunkter, og et kodepunkt skrives som U+ efterfulgt af mindst fire hexadecimale cifre: å er U+00E5, ґ er U+0490, og eurotegnet er U+20AC. Konsortiets stabilitetspolitik lover, at et tegn aldrig flyttes eller fjernes, når det først er kodet, og at dets navn aldrig ændres, så tekst, der kodes i dag, betyder det samme i alle senere versioner.

To skel bærer det meste af standarden. Det første går mellem tegn og glyf. Unicode koder tegnet, det abstrakte bogstav, og overlader formen til fonten: kernespecifikationen slår fast, at glyffernes udseende er fontleverandørernes ansvar og ikke en del af standarden. Det samme kodepunkt kan se forskelligt ud i

to skrifttyper, og en skrifttype uden en form for det kan slet ikke tegne det. Det andet skel går mellem kodepunktet og de bytes, der gemmer det. Unicode definerer tre kodningsformer, UTF-8, UTF-16 og UTF-32, der skriver hvert kodepunkt som en eller flere enheder på 8, 16 eller 32 bit, og specifikationen kalder det en fælde at gøre det til et spørgsmål om »UTF-8 versus Unicode«: alle tre er lige gyldige måder at implementere de samme tegn på.

Brug begrebet i en beslutning

Brug »Hvad er Unicode?« 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

Før Unicode havde hver skrift sine egne tegntabeller, og en byte betød ingenting, før man vidste, hvilken tabel der var i brug. I den vesteuropæiske tegntabel windows-1252 er byten E5 et å; i den kyrilliske windows-1251 er samme byte bogstavet е. Ingen af dem kunne rumme den andens bogstaver: windows-1251 har ґ, men hverken å, æ eller ø, og windows-1252 har det omvendt. Et produkt på dansk og ukrainsk krævede to kodninger og et pålideligt overblik over, hvilken tekst der brugte hvilken, og en sætning, der citerede begge sprog, kunne ikke gemmes i nogen af de to kodninger.

Unicode gjorde det bogholderi overflødigt. Alle de sprog, et produkt udgives på, og alle de navne, en kunde skriver i en formular, passer ind i ét tegnsæt, og det samme nummer betyder det samme tegn på alle systemer, der læser det. Det er det, der lader en database, et API og en browser sende tekst imellem sig uden først at blive enige om en tegntabel.

Standarden afgør også spørgsmål, som ellers ville blive afgjort forskelligt i hvert eneste program. Den slår tegn sammen på tværs af sprog: kommaet er ét tegn, uanset om det skiller tusinder ad, som på engelsk, eller decimaler, som på dansk, så det er sproget og dets konventioner, ikke kodepunktet, der afgør betydningen. Den slår ikke tegn sammen på tværs af skrifter: det latinske o, det kyrilliske о og det græske ο ser ens ud i de fleste skrifttyper og er tre tegn, fordi de hører til tre skrifter, og ældre kodninger allerede holdt dem adskilt. Og den

fastlægger, hvilke sekvenser der er ækvivalente. Å kan gemmes som ét samlet tegn eller som A efterfulgt af en kombinerende ring, og standarden definerer de to som kanonisk ækvivalente, så de skal vises og forstås ens, med normaliseringsformer, der fører begge over i én aftalt stavemåde.

Den rækkevidde har en pris, og det er programmøren, der betaler den. Et tegn er ikke en byte, ikke altid én enhed i en streng og ikke altid det, en læser tæller som ét bogstav. Kode, der behandler de tre som det samme, klarer sine test med engelske data og fejler på navne, adresser og beskeder fra netop de mennesker, den blev skrevet til.

Sådan ser det ud i praksis

JavaScript gør lagene synlige. En streng er en række UTF-16-kodeenheder, og spreder man den ud i et array, deles den op i kodepunkter:

```
const hex = (n) => n.toString(16).toUpperCase().padStart(4, "0");
for (const text of ["a", "å", "Ґ", "€", "𐌂"]) {
  const points = [...text].map((c) => "U+" + hex(c.codePointAt(0)));
  console.log(text, points.join(" "), text.length, [...text].length);
}
```

Kørt i Node.js 25 skriver koden hvert tegn, dets kodepunkt, dets længde i UTF-16-enheder og dets længde i kodepunkter:

```
a U+0061 1 1
å U+00E5 1 1
Ґ U+0490 1 1
€ U+20AC 1 1
𐌂 U+10302 2 1
```

Den sidste linje er et gammelitalisk bogstav. Dets kodepunkt ligger over U+FFFF, så UTF-16 gemmer det som to enheder, et surrogatpar, og `length` melder 2 for det, der er ét tegn.

Ækvivalens viser sig på samme måde. Her er tre strenge, der alle vises som Å:

```
const forms = ["\u00C5", "A\u030A", "\u212B"];
console.log("form length equal NFC");
for (const [i, s] of forms.entries()) {
  console.log(i + 1, s.length, s === forms[0], s.normalize("NFC") === forms[0]);
}
```

```
form length equal NFC
1 1 true true
2 2 false true
3 1 false true
```

Den første er det samlede bogstav, U+00C5. Den anden er et almindeligt A efterfulgt af U+030A, den kombinerende ring over. Den tredje er U+212B, ångstrømtegnet, som Unicode Character Database fører tilbage til U+00C5. Sammenlignet, som de er, er de tre forskellige strenge; efter normalisering til NFC er alle tre den samme. Intet på skærmen skiller dem ad, så en søgning, et unikt indeks eller et login, der sammenligner de rå strenge, ser tre forskellige navne.

Også dette site er afhængigt af kodepunkter, bare et mindre synligt sted. Noten om understregningen, der gik gennem hvert bogstav beskriver sitets Ubuntu-skrifter til brødtekst med latinske og kyrilliske tegn som seks snit, tre vægte med latinske tegn og tre med kyrilliske, så en ukrainsk side sættes i samme skrifttype som en engelsk. Hvert snit angiver det tegnområde, det dækker. Områderne styrer valget af skrifttype til sidens tegn; en eksplicit forudindlæsning kan også anmode om en fil, før browseren finder den på normal vis. Den kyrilliske erklæring angiver sit område i kodepunkter:

```
unicode-range: U+0400-045F, U+0490-0491;
```

Det første område går fra È til џ og rummer de ukrainske Є, І og Ї sammen med resten af det grundlæggende kyrilliske alfabet. Det andet rummer kun Ґ og ґ, der ligger længere henne i den kyrilliske blok. Områderne angiver, hvilket skrifttypesnit der kan vælges til et tegn, men skriften skal også rumme tegnets glyf. De er ingen garanti for, at en side kun henter filer til ét skriftsystem. Engelsk og dansk tekst bruger de latinske snit, men den ukrainske sprogbetegnelse kan kræve et kyrillisk snit, og sitet forudindlæser eksplicit det lette kyrilliske brødtekstsnit på ikke-kyrilliske sider til skærmvisning.

Hvor det går galt

Længden tælles i kodeenheder. JavaScripts `length` tæller UTF-16-enheder, så ᴋ tæller som to, og et A med kombinerende ring tæller som to, hvor en læser ser ét bogstav. En grænse, der håndhæves i enheder, afviser tekst, der sagtens kan være der, og et snit ved en fast position kan dele et tegn: `"A\u030A".slice(0, 1)` beholder A'et og taber ringen, og skærer man ᴋ over på midten, står der et enligt surrogat tilbage, som UTF-8-koderen erstatter med U+FFFD, erstatningstegnet. Beslut for hver grænse, hvad den tæller: bytes, hvor det er lagringen, der sætter grænsen, og tegn, som læseren ser dem, hvor det er et menneske. `Intl.Segmenter` tæller på den sidste måde, og for Århus skrevet med kombinerende ring giver den 5, hvor `length` giver 6. Klip kun tekst ved de grænser, den melder.

Ens tekst sammenlignes enhed for enhed. De tre former af Å ovenfor er tre forskellige strenge for `===`, for en hashfunktion og for et unikt indeks, så en kunde kan oprette det samme navn to gange eller ikke finde et dokument ud fra den titel, vedkommende skrev. Normalisér tekst til NFC, hvor den kommer ind i systemet, gem den normaliseret, og sammenlign normaliserede former. Stabilitetspolitikken garanterer, at en streng, der er normaliseret under én version, forbliver normaliseret under alle senere, forudsat at strengen kun indeholder tegn, som den version kender, så den kode, der normaliserer, skal være lige så ny som teksten.

Tegn, der ligner hinanden, går for at være ens. Det latinske o, det kyrilliske о og det græske ο er forskellige kodepunkter, som de fleste skrifttyper tegner ens, så to kontonavne eller to varenumre kan se identiske ud og afvige i et enkelt tegn. Unicode holder dem bevidst adskilt, og konsortiets standard for sikkerhedsmekanismer, UTS #39, beskriver, hvordan den slags forfalskning kan begrænses. Hvor et navn identificerer en person eller en betalingsmodtager, så tillad kun én skrift pr. navn, eller tjek navnet mod standardens sikkerhedsdata, før det godkendes.

Bytes læses i den forkerte kodning. Tekst, der er gemt som UTF-8 og læses som windows-1252, forvandler blåbær til blÃ¥bÃ¦r, fordi hvert bogstav på to bytes læses som to bogstaver på én byte hver. Ordet Unicode siger ikke, hvordan de bytes blev skrevet; det gør kun kodningsformen. Angiv den overalt, hvor tekst krydser en grænse, i HTTP-headeren, på siden, på databaseforbindelsen og i filformatet, og behandl en fil, der kommer uden, som et spørgsmål til et menneske og ikke som et gæt til koden.

Teksten får skylden for en manglende glyf. Når et bogstav vises som en tom firkant, er årsagen som regel en skrifttype uden en form for det kodepunkt, og selve teksten er intakt. Fordi Unicode overlader formerne til skrifttyperne, skal hver skrifttype, et produkt bruger, dække alle de tegn, dets sprog har brug for, de ukrainske Є, І, Ї og Ґ lige så vel som A til Я, ellers tegner browseren dem med den reserveskrift, den nu finder. Noten ovenfor giver netop det som grunden til, at sitet overhovedet serverer kyrilliske dele: uden dem ville en ukrainsk side falde tilbage på det, systemet nu tilbyder. Test hvert sprog med hele dets alfabet frem for en enkelt eksempelsætning.

Navne tjekkes mod A til Z. Et mønster som `^[A-Za-z' -]+$` godkender O'Neill og afviser Søren Ågård, Zoë og Ґалаґан, altså netop de navne, et dansk eller ukrainsk produkt er til for. Afgør i stedet, om et tegn er et bogstav, ud fra dets Unicode-egenskaber: i JavaScript godkender `/^[\p{L}\p{M}' -]+$/u` alle fire. Egenskaberne kommer fra den kopi af Unicode Character Database, der ligger i kørselsmiljøet. For eksempel rapporterer Node.js 25.2.1, som bruges til eksemplerne her, Unicode 16.0; det gælder det konkrete build og er ikke en garanti for hver udgave i 25.x-serien. Bogstaver, der er kommet til i senere Unicode-versioner, afvises, indtil kørselsmiljøets tegndata opgraderes. Validér navne løst, og lad et menneske frem for et mønster afgøre de svære tilfælde.

Læs mere

- About the Unicode Standard — konsortiets egen beskrivelse af standarden, hvad en version består af, og hvor den nyeste udgives.
- The Unicode Standard, Chapter 2: General Structure — de ti designprincipper, koderummet, de tre kodningsformer og reglerne for ækvivalente sekvenser.
- Unicode Character Encoding Stability Policies — det, der aldrig kan ændres, når et tegn først er kodet, fra kodepunktet og navnet til normaliseringen.
- Unicode 18.0.0 — det, den aktuelle version tilføjer, og de tekniske standarder, der udgives i takt med den, blandt dem sortering og sikkerhed.
- Node.js: process.versions — kørselsmiljøets oversigt over komponentversioner, som her knytter Unicode-dataeksemplet til ét bestemt Node.js-build frem for en hel udgivelsesserie.
- CSS Fonts: character ranges — hvordan erklærede tegnområder og skriftens faktiske glyfdækning tilsammen bestemmer, hvilke tegn et skrifttypesnit kan vise.

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 Unicode?« 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/unicode/