Engineer

Unicode

⌨ standarden som gir hvert tegn sitt eget nummer

Hva det er

Unicode er standarden som gir hver bokstav, hvert siffer, hvert skilletegn og hvert symbol i verdens skriftsystemer sitt eget nummer. Ifølge dem som vedlikeholder standarden, er Unicode den universelle tegnkodingen: laget for at tekst på den moderne verdens språk og i dens tekniske fagfelt skal kunne utveksles, behandles og vises hvor som helst, og utvidet til de klassiske og historiske tekstene på mange skriftspråk. Ett og samme repertoar rommer den norske å, den ukrainske ґ, eurotegnet og bokstavene i skrifter ingen har skrevet på i århundrer, slik at ett enkelt dokument eller én enkelt kolonne i en database kan inneholde alle sammen.

Formelt består en versjon av Unicode-standarden av kjernespesifikasjonen, kodetabellene, standardens vedlegg, Unicode Standard Annexes, og Unicode Character Database, de maskinlesbare filene som beskriver egenskapene til hvert tegn. Unicode Technical Committee vedlikeholder standarden løpende, Unicode-konsortiet gir den ut, og versjon 18.0 ble kunngjort 16. september 2026.

Nummeret et tegn får, kalles kodepunktet. Koderommet går fra 0 til 10FFFF heksadesimalt, 1 114 112 kodepunkter i alt, og et kodepunkt skrives som U+ fulgt av minst fire heksadesimale sifre: å er U+00E5, ґ er U+0490, og eurotegnet er U+20AC. Konsortiets stabilitetsregler lover at et tegn aldri flyttes eller fjernes når det først er kodet, og at navnet aldri endres, så tekst som kodes i dag, betyr det samme i alle senere versjoner.

To skiller bærer det meste av standarden. Det første går mellom tegn og glyf. Unicode koder tegnet, den abstrakte bokstaven, og overlater formen til fonten: kjernespesifikasjonen slår fast at glyfenes utseende er fontleverandørenes ansvar og ikke en del av standarden. Det samme kodepunktet kan se forskjellig ut i to skrifttyper, og en skrifttype uten en form for det kan ikke tegne det i det hele

tatt. Det andre skillet går mellom kodepunktet og bytene som lagrer det. Unicode definerer tre kodingsformer, UTF-8, UTF-16 og UTF-32, som skriver hvert kodepunkt som én eller flere enheter på 8, 16 eller 32 bit, og spesifikasjonen kaller det en felle å gjøre det til et spørsmål om «UTF-8 versus Unicode»: alle tre er like gyldige måter å implementere de samme tegnene på.

Bruk begrepet i en beslutning

Bruk «Hva er Unicode?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Programvare som leses naturlig i alle markeder du selger i, også datoene, valutaene og måten navn sorteres på.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Bygde et engelsk/dansk i18n-system — 12,027 meldinger per språkversjon, på engelsk og dansk

Bygde et engelsk/dansk i18n-system Ta kontakt →

Hvorfor det er viktig

Før Unicode hadde hver skrift sine egne kodesider, og en byte betydde ingenting før man visste hvilken side som var i bruk. I den vesteuropeiske kodesiden windows-1252 er byten E5 en å; i den kyrilliske windows-1251 er den samme byten bokstaven е. Ingen av sidene kunne romme den andres bokstaver: windows-1251 har ґ, men verken å, æ eller ø, og i windows-1252 er det omvendt. Et produkt på norsk og ukrainsk trengte to kodinger og full oversikt over hvilken tekst som brukte hvilken, og en setning som siterte begge språkene, fikk ikke plass i noen av kodingene.

Unicode gjorde den bokføringen overflødig. Alle språkene et produkt publiseres på, og alle navnene en kunde skriver inn i et skjema, får plass i ett tegnsett, og det samme nummeret betyr det samme tegnet på alle systemer som leser det. Det er det som lar en database, et API og en nettleser sende tekst seg imellom uten først å bli enige om en kodeside.

Standarden avgjør også spørsmål som ellers ville blitt avgjort ulikt i hvert eneste program. Den slår sammen tegn på tvers av språk: kommaet er ett tegn enten det skiller tusener, som på engelsk, eller desimaler, som på norsk, så det er språket og konvensjonene, ikke kodepunktet, som avgjør betydningen. Den slår ikke sammen tegn på tvers av skrifter: den latinske o, den kyrilliske о og den greske ο ser like ut i de fleste skrifttyper og er tre tegn, fordi de hører til tre skrifter og eldre kodinger allerede holdt dem atskilt. Og den fastsetter hvilke sekvenser som er ekvivalente. Å kan lagres som ett ferdig sammensatt tegn eller

som A fulgt av en kombinerende ring, og standarden definerer de to som kanonisk ekvivalente, slik at de skal vises og tolkes likt, med normaliseringsformer som fører begge over i én avtalt skrivemåte.

Den bredden har en pris, og det er programmereren som betaler den. Et tegn er ikke en byte, ikke alltid én enhet i en streng og ikke alltid det en leser teller som én bokstav. Kode som behandler de tre som det samme, består testene sine med engelske data og feiler på navnene, adressene og meldingene til nettopp de menneskene den ble skrevet for.

Slik ser det ut i praksis

JavaScript gjør lagene synlige. En streng er en rekke UTF-16-kodeenheter, og sprer man den ut i en array, deles den opp 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);
}
```

Kjørt i Node.js 25 skriver koden ut hvert tegn, kodepunktet, lengden i UTF-16-enheter og lengden 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 siste linjen er en gammelitalisk bokstav. Kodepunktet ligger over U+FFFF, så UTF-16 lagrer den som to enheter, et surrogatpar, og `length` melder 2 for det som er ett tegn.

Ekvivalens viser seg på samme måte. Her er tre strenger som 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 den ferdig sammensatte bokstaven, U+00C5. Den andre er en vanlig A fulgt av U+030A, den kombinerende ringen over. Den tredje er U+212B, ångstrømtegnet, som Unicode Character Database fører tilbake til U+00C5. Sammenlignet som de er, er de tre ulike strenger; etter normalisering til NFC er alle tre den samme. Ingenting på skjermen skiller dem fra hverandre, så et søk, en unik indeks eller en innlogging som sammenligner de rå strengene, ser tre forskjellige navn.

Også dette nettstedet er avhengig av kodepunkter, bare et mindre synlig sted. Notatet om understrekingen som krysset hver bokstav beskriver nettstedets latinske og kyrilliske Ubuntu-skrifter for brødtekst som seks snitt, tre vekter med latinske tegn og tre med kyrilliske, slik at en ukrainsk side settes i samme skrifttype som en engelsk. Hvert snitt oppgir hvilket tegnområde det dekker. Områdene styrer valget av skrift for tegnene på siden; en uttrykkelig forhåndsinnlasting kan også be om en fil før nettleseren ellers oppdager skriften. Den kyrilliske deklarasjonen oppgir området sitt i kodepunkter:

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

Det første området går fra È til ѵ og rommer de ukrainske Є, І og Ї sammen med resten av det grunnleggende kyrilliske alfabetet. Det andre rommer bare Ґ og ґ, som ligger lenger ut i den kyrilliske blokken. Tegnområdene oppgir hvilket skriftsnitt som kan velges for et tegn, men skriften må også inneholde tegnets glyf. De garanterer ikke at en side henter filer for bare ett skriftsystem. Engelsk og dansk tekst bruker de latinske snittene, men den ukrainske språketiketten kan kreve et kyrillisk snitt, og nettstedet forhåndslaster uttrykkelig det lette kyrilliske brødtekstsnittet på ikke-kyrilliske sider for skjermvisning.

Hvor det går galt

Lengden telles i kodeenheter. JavaScripts `length` teller UTF-16-enheter, så 𐐷 teller som to, og en A med kombinerende ring teller som to der en leser ser én bokstav. En grense som håndheves i enheter, avviser tekst som får plass, og et kutt ved en fast posisjon kan dele et tegn: `"A\u030A".slice(0, 1)` beholder A-en og mister ringen, og deler man 𐐷 på midten, står man igjen med et enslig surrogat som UTF-8-koderen erstatter med U+FFFD, erstatningstegnet. Bestem for hver grense hva den teller: byte der det er lagringen som setter grensen, og tegn slik leseren ser dem der det er et menneske som gjør det. `Intl.Segmenter` teller på den siste måten, og for Ålesund skrevet med kombinerende ring gir den 7 der `length` gir 8. Klipp tekst bare ved grensene den melder.

Lik tekst sammenlignes enhet for enhet. De tre formene av Å ovenfor er tre ulike strenger for `===`, for en hashfunksjon og for en unik indeks, så en kunde kan registrere det samme navnet to ganger eller ikke finne et dokument ut fra tittelen vedkommende skrev. Normaliser tekst til NFC der den kommer inn i systemet, lagre den normalisert, og sammenlign normaliserte former. Stabilitetsreglene garanterer at en streng som er normalisert under én versjon, forblir normalisert under alle senere, forutsatt at strengen bare inneholder tegn som den versjonen kjenner, så koden som normaliserer, må være like ny som teksten.

Tegn som ligner hverandre, går for å være like. Den latinske o, den kyrilliske о og den greske ο er ulike kodepunkter som de fleste skrifttyper tegner likt, så to kontonavn eller to varenumre kan se identiske ut og skille seg i ett eneste tegn. Unicode holder dem bevisst atskilt, og konsortiets standard for sikkerhetsmekanismer, UTS #39, beskriver hvordan slik forfalskning kan begrenses. Der et navn identifiserer en person eller en betalingsmottaker, bør du tillate bare én skrift per navn eller sjekke navnet mot standardens sikkerhetsdata før det godtas.

Byte leses med feil koding. Tekst som er lagret som UTF-8 og leses som windows-1252, gjør blåbær til blÃ¥bÃ¦r, fordi hver bokstav på to byte leses som to bokstaver på én byte hver. Ordet Unicode sier ikke hvordan bytene ble skrevet; det gjør bare kodingsformen. Oppgi den overalt der tekst krysser en grense, i HTTP-headeren, på siden, på databaseforbindelsen og i filformatet, og behandle en fil som kommer uten, som et spørsmål til et menneske og ikke som en gjetning for koden.

Teksten får skylden for en manglende glyf. Når en bokstav vises som en tom firkant, er årsaken som regel en skrifttype uten en form for det kodepunktet, og selve teksten er intakt. Fordi Unicode overlater formene til skrifttypene, må hver skrifttype et produkt bruker, dekke alle tegnene språkene trenger, de ukrainske Є, І, Ї og Ґ like så vel som A til я, ellers tegner nettleseren dem med den reserveskriften den finner. Notatet ovenfor oppgir nettopp dette som grunnen til at nettstedet i det hele tatt leverer kyrilliske deler: uten dem ville en ukrainsk side falle tilbake på det systemet tilbyr. Test hvert språk med hele alfabetet i stedet for én eksempelsetning.

Navn sjekkes mot A til Z. Et mønster som `^[A-Za-z' -]+$` godtar O'Neill og avviser Bjørn Ødegård, Zoë og Ґалаґан, altså nettopp de navnene et norsk eller ukrainsk produkt er til for. Avgjør heller om et tegn er en bokstav ut fra Unicode-egenskapene: i JavaScript godtar `/^[\p{L}\p{M}' -]+$/u` alle fire. Egenskapene kommer fra den kopien av Unicode Character Database som ligger i kjøremiljøet. For eksempel rapporterer Node.js 25.2.1, som brukes til eksemplene her, Unicode 16.0; det er én bestemt byggversjon, ikke en garanti for hver utgave i 25.x-serien. Bokstaver som er kommet til i senere Unicode-versjoner, avvises inntil kjøremiljøets tegndata oppgraderes. Valider navn løst, og la et menneske, ikke et mønster, avgjøre de vanskelige tilfellene.

Les mer

- About the Unicode Standard — konsortiets egen beskrivelse av standarden, hva en versjon består av, og hvor den nyeste blir publisert.
- The Unicode Standard, Chapter 2: General Structure — de ti designprinsippene, koderommet, de tre kodingsformene og reglene for ekvivalente sekvenser.
- Unicode Character Encoding Stability Policies — det som aldri kan endres når et tegn først er kodet, fra kodepunktet og navnet til normaliseringen.
- Unicode 18.0.0 — hva den gjeldende versjonen legger til, og de tekniske standardene som versjoneres sammen med den, blant dem sortering og sikkerhet.
- Node.js: process.versions — kjøremiljøets oversikt over komponentversjoner, brukt her til å knytte Unicode-dataeksempelet til én bestemt Node.js-byggversjon i stedet for en hel utgivelsesserie.
- CSS Fonts: character ranges — hvordan deklarerte tegnområder og skriftens faktiske glyfdekning sammen avgjør hvilke tegn et skriftsnitt kan gjengi.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er Unicode?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/unicode/