Engineer

Unicode

⌨ standarden som ger varje tecken ett eget nummer

Vad det är

Unicode är standarden som ger varje bokstav, siffra, skiljetecken och symbol i världens skriftsystem ett eget nummer. Enligt dem som underhåller standarden är Unicode den universella teckenkodningen: utformad för att text på den moderna världens språk och inom dess tekniska fält ska kunna utbytas, bearbetas och visas var som helst, och utökad till klassiska och historiska texter på många skriftspråk. En och samma repertoar rymmer det svenska å, det ukrainska ґ, eurotecknet och bokstäverna i skrifter som ingen har skrivit på i århundraden, så att ett enda dokument, eller en enda kolumn i en databas, kan innehålla dem alla.

Formellt består en version av Unicode-standarden av kärnspecifikationen, kodtabellerna, standardens bilagor, Unicode Standard Annexes, och Unicode Character Database, de maskinläsbara filer som beskriver varje teckens egenskaper. Unicode Technical Committee underhåller standarden löpande, Unicode-konsortiet ger ut den, och version 18.0 tillkännagavs den 16 september 2026.

Numret som ett tecken får kallas dess kodpunkt. Kodrymden går från 0 till 10FFFF hexadecimalt, 1 114 112 kodpunkter totalt, och en kodpunkt skrivs som U+ följt av minst fyra hexadecimala siffror: å är U+00E5, ґ är U+0490 och eurotecknet är U+20AC. Konsortiets stabilitetsregler lovar att ett tecken aldrig flyttas eller tas bort när det väl har kodats, och att dess namn aldrig ändras, så text som kodas i dag betyder samma sak i alla senare versioner.

Två distinktioner bär det mesta av standarden. Den första går mellan tecken och glyf. Unicode kodar tecknet, den abstrakta bokstaven, och överlåter formen åt typsnittet: kärnspecifikationen slår fast att glyfernas utseende är typsnittsleverantörernas ansvar och inte en del av standarden. Samma kodpunkt kan se olika ut i två typsnitt, och ett typsnitt som saknar en form för den kan inte

rita den alls. Den andra går mellan kodpunkten och de byte som lagrar den. Unicode definierar tre kodningsformer, UTF-8, UTF-16 och UTF-32, som skriver varje kodpunkt som en eller flera enheter om 8, 16 eller 32 bitar, och specifikationen kallar det en fälla att göra det till en fråga om ”UTF-8 versus Unicode”: alla tre är lika giltiga sätt att implementera samma tecken.

Använd begreppet i ett beslut

Använd ”Vad är Unicode?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Mjukvara som känns naturlig på varje marknad du säljer på, inklusive datum, valutor och hur namn sorteras.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Byggde ett i18n-system för engelska och danska — 12,027 meddelanden per språkversion, på engelska och danska

Byggde ett i18n-system för engelska och danska Hör av er →

Varför det spelar roll

Före Unicode hade varje skrift sina egna kodsidor, och en byte betydde ingenting förrän man visste vilken sida som användes. I den västeuropeiska kodsidan windows-1252 är byten E5 ett å; i den kyrilliska windows-1251 är samma byte bokstaven е. Ingen av sidorna kunde rymma den andras bokstäver: windows-1251 har ґ men varken å, ä eller ö, och i windows-1252 är det tvärtom. En produkt på svenska och ukrainska behövde två kodningar och god ordning på vilken text som använde vilken, och en mening som citerade båda språken gick inte att lagra i någon av de två kodningarna.

Unicode gjorde den bokföringen överflödig. Alla språk som en produkt ges ut på, och alla namn som en kund skriver i ett formulär, ryms i en och samma teckenuppsättning, och samma nummer betyder samma tecken i alla system som läser det. Det är det som låter en databas, ett API och en webbläsare skicka text mellan sig utan att först komma överens om en kodsida.

Standarden avgör också frågor som annars skulle avgöras olika i varje program. Den slår ihop tecken över språkgränser: kommat är ett och samma tecken oavsett om det skiljer tusental åt, som på engelska, eller decimaler, som på svenska, så det är språket och dess konventioner, inte kodpunkten, som avgör betydelsen. Den slår inte ihop tecken över skriftgränser: det latinska o, det kyrilliska о och det grekiska ο ser likadana ut i de flesta typsnitt och är tre tecken, eftersom de hör till tre skrifter och äldre kodningar redan höll isär dem. Och den slår fast vilka sekvenser som är ekvivalenta. Å kan lagras som ett färdigt

sammansatt tecken eller som A följt av en kombinerande ring, och standarden definierar de två som kanoniskt ekvivalenta, så att de ska visas och tolkas likadant, med normaliseringsformer som för över båda till en överenskommen stavning.

Den bredden har ett pris, och det är programmeraren som betalar det. Ett tecken är inte en byte, inte alltid en enhet i en sträng och inte alltid det som en läsare räknar som en bokstav. Kod som behandlar de tre som samma sak klarar sina tester med engelska data och fallerar på namnen, adresserna och meddelandena från just de människor den skrevs för.

Så ser det ut i praktiken

JavaScript gör lagren synliga. En sträng är en följd av UTF-16-kodenheter, och sprider man ut den i en array delas den upp i kodpunkter:

```
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);
}
```

Kört i Node.js 25 skriver koden ut varje tecken, dess kodpunkt, dess längd i UTF-16-enheter och dess längd i kodpunkter:

```
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 sista raden är en fornitalisk bokstav. Dess kodpunkt ligger över U+FFFF, så UTF-16 lagrar den som två enheter, ett surrogatpar, och `length` rapporterar 2 för det som är ett tecken.

Ekvivalens syns på samma sätt. Här är tre strängar som alla visas 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 första är den färdigt sammansatta bokstaven, U+00C5. Den andra är ett vanligt A följt av U+030A, den kombinerande ringen ovanför. Den tredje är U+212B, ångströmtecknet, som Unicode Character Database för tillbaka till U+00C5. Jämförda som de är, är de tre olika strängar; efter normalisering till NFC är alla tre samma sträng. Ingenting på skärmen skiljer dem åt, så en sökning, ett unikt index eller en inloggning som jämför de råa strängarna ser tre olika namn.

Även den här webbplatsen är beroende av kodpunkter, bara på ett mindre synligt ställe. Anteckningen om understrykningen som korsade varje bokstav beskriver webbplatsens latinska och kyrilliska Ubuntu-typsnitt för brödtext som sex varianter, tre vikter med latinska tecken och tre med kyrilliska, så att en ukrainsk sida sätts i samma typsnitt som en engelsk. Varje variant deklarerar det teckenintervall den täcker. Intervallen styr valet av typsnitt för tecknen på sidan; en uttrycklig förladdning kan också begära en fil innan webbläsaren normalt hittar den. Den kyrilliska deklarationen anger sitt intervall i kodpunkter:

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

Det första intervallet går från È till џ och rymmer de ukrainska Є, І och Ї tillsammans med resten av det grundläggande kyrilliska alfabetet. Det andra rymmer bara Ґ och ґ, som ligger längre fram i det kyrilliska blocket. Intervallen anger för vilka tecken ett typsnitt får väljas, men typsnittet måste också innehålla tecknens glyfer. De garanterar inte att en sida hämtar filer för bara ett skriftsystem. Engelsk och dansk text använder de latinska varianterna, men den ukrainska språketiketten kan behöva en kyrillisk variant, och webbplatsen förladdar uttryckligen den lätta kyrilliska varianten för brödtext på icke-kyrilliska sidor för skärmvisning.

Var det går fel

Längden räknas i kodenheter. JavaScripts `length` räknar UTF-16-enheter, så 𐐷 räknas som två, och ett A med kombinerande ring räknas som två där en läsare ser en bokstav. En gräns som upprätthålls i enheter avvisar text som får plats, och ett snitt vid en fast position kan dela ett tecken: `"A\u030A".slice(0, 1)` behåller A:et och tappar ringen, och delar man 𐐷 på mitten blir ett ensamt surrogat kvar, som UTF-8-kodaren ersätter med U+FFFD, ersättningstecknet. Bestäm för varje gräns vad den räknar: byte där det är lagringen som sätter gränsen, och tecken så som läsaren ser dem där det är en människa som gör det. `Intl.Segmenter` räknar på det senare sättet, och för Umeå skrivet med kombinerande ring ger den 4 där `length` ger 5. Klipp bara text vid de gränser den rapporterar.

Likadan text jämförs enhet för enhet. De tre formerna av Å ovan är tre olika strängar för `===`, för en hashfunktion och för ett unikt index, så en kund kan registrera samma namn två gånger eller inte hitta ett dokument utifrån den titel kunden skrev. Normalisera text till NFC där den kommer in i systemet, lagra den normaliserad och jämför normaliserade former. Stabilitetsreglerna garanterar att en sträng som är normaliserad under en version förblir normaliserad under alla senare, förutsatt att strängen bara innehåller tecken som den versionen känner till, så koden som normaliserar måste vara lika ny som texten.

Tecken som ser likadana ut tas för samma tecken. Det latinska o, det kyrilliska о och det grekiska ο är olika kodpunkter som de flesta typsnitt ritar likadant, så två kontonamn eller två artikelnummer kan se identiska ut och skilja sig i ett enda tecken. Unicode håller medvetet isär dem, och konsortiets standard för säkerhetsmekanismer, UTS #39, beskriver hur den sortens förfalskning kan begränsas. Där ett namn identifierar en person eller en betalningsmottagare bör du bara tillåta en skrift per namn eller kontrollera namnet mot standardens säkerhetsdata innan det godkänns.

Byte läses med fel kodning. Text som har lagrats som UTF-8 och läses som windows-1252 förvandlar blåbär till blÃ¥bÃ¤r, eftersom varje bokstav på två byte läses som två bokstäver på en byte var. Ordet Unicode säger inte hur texten skrevs som byte; det gör bara kodningsformen. Ange den överallt där text korsar en gräns, i HTTP-headern, på sidan, på databasanslutningen och i filformatet, och behandla en fil som kommer utan som en fråga till en människa och inte som en gissning åt koden.

Texten får skulden för en saknad glyf. När en bokstav visas som en tom ruta är orsaken oftast ett typsnitt som saknar en form för den kodpunkten, och själva texten är intakt. Eftersom Unicode överlåter formerna åt typsnitten måste varje typsnitt som en produkt använder täcka alla tecken som dess språk behöver, de ukrainska Є, І, Ї och Ґ lika väl som A till я, annars ritar webbläsaren dem med det reservtypsnitt den hittar. Anteckningen ovan anger just det som skälet till att webbplatsen alls serverar kyrilliska delar: utan dem skulle en ukrainsk sida falla tillbaka på vad systemet nu erbjuder. Testa varje språk med hela dess alfabet i stället för en enstaka exempelmening.

Namn kontrolleras mot A till Z. Ett mönster som `^[A-Za-z' -]+$` godkänner O'Neill och avvisar Åsa Lindström, Zoë och Ґалаґан, alltså just de namn som en svensk eller ukrainsk produkt finns till för. Avgör i stället om ett tecken är en bokstav utifrån dess Unicode-egenskaper: i JavaScript godkänner `/^[\p{L}\p{M}' -]+$/u` alla fyra. Egenskaperna kommer från den kopia av Unicode Character Database som finns i körmiljön. Exempelvis rapporterar Node.js 25.2.1, som används för exemplen här, Unicode 16.0; det gäller en viss byggversion och är inte en garanti för varje utgåva i 25.x-serien. Bokstäver som har tillkommit i senare Unicode-versioner avvisas tills körmiljöns teckendata uppgraderas. Validera namn generöst och låt en människa, inte ett mönster, avgöra de svåra fallen.

Läs mer

- About the Unicode Standard — konsortiets egen beskrivning av standarden, vad en version består av och var den senaste publiceras.
- The Unicode Standard, Chapter 2: General Structure — de tio designprinciperna, kodrymden, de tre kodningsformerna och reglerna för ekvivalenta sekvenser.
- Unicode Character Encoding Stability Policies — det som aldrig kan ändras när ett tecken väl har kodats, från kodpunkten och namnet till normaliseringen.
- Unicode 18.0.0 — vad den aktuella versionen tillför och de tekniska standarder som versioneras tillsammans med den, bland dem sortering och säkerhet.
- Node.js: process.versions — körmiljöns redovisning av komponentversioner, som här knyter exemplet med Unicode-data till en viss byggd version av Node.js i stället för en hel utgivningsserie.
- CSS Fonts: character ranges — hur deklarerade teckenintervall och typsnittets faktiska glyftäckning tillsammans avgör vilka tecken ett typsnitt kan återge.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Unicode?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/unicode/