Engineer

Unicode

⌨ der Standard, der jedem Zeichen eine eigene Nummer gibt

Was es ist

Unicode ist der Standard, der jedem Buchstaben, jeder Ziffer, jedem Satzzeichen und jedem Symbol der Schriftsysteme der Welt eine eigene Nummer gibt. Seine Betreuer bezeichnen ihn als universelle Zeichenkodierung: entworfen, damit Texte in den Sprachen und technischen Fachgebieten der modernen Welt überall ausgetauscht, verarbeitet und angezeigt werden können, und zugleich fähig, ältere Schriftsysteme und historische Schriftzeugnisse darzustellen. Ein einziges Repertoire enthält das dänische å, das ukrainische ґ, das Eurozeichen und Buchstaben von Schriften, in denen seit Jahrhunderten niemand mehr schreibt. So kann ein einzelnes Dokument oder eine einzelne Datenbankspalte sie alle enthalten.

Genau genommen umfasst eine Unicode-Version mehrere aufeinander abgestimmte Teile: die Kernspezifikation, Zeichentabellen, Anhänge und eine Zeichendatenbank aus maschinenlesbaren Dateien mit den Eigenschaften jedes Zeichens. Das Unicode Technical Committee pflegt den Standard fortlaufend, das Unicode Consortium veröffentlicht ihn, und Version 18.0 wurde am 16. September 2026 angekündigt.

Die Nummer eines Zeichens ist sein Codepunkt. Der Coderaum reicht von 0 bis hexadezimal 10FFFF, insgesamt 1.114.112 Codepunkte. Geschrieben wird ein Codepunkt als U+ mit mindestens vier Hexadezimalziffern dahinter: å ist U+00E5, ґ ist U+0490 und das Eurozeichen U+20AC. Die Stabilitätsrichtlinie des Konsortiums verspricht einem kodierten Zeichen einen dauerhaften Platz: Es wird nie verschoben oder entfernt, und sein Name ändert sich nie. Heute kodierter Text bedeutet deshalb in jeder späteren Version dasselbe.

Zwei Unterscheidungen sind für das Verständnis des Standards entscheidend. Die erste trennt Zeichen und Glyphe. Unicode kodiert das Zeichen, den abstrakten Buchstaben, und überlässt seine Form der Schriftart: Laut Kernspezifikation sind Glyphenformen Sache der Schriftthersteller und kein Bestandteil des Standards. Derselbe Codepunkt kann in zwei Schriftarten unterschiedlich aussehen; eine Schriftart ohne passende Form kann ihn überhaupt nicht zeichnen. Die zweite Unterscheidung trennt den Codepunkt von den Bytes, die ihn speichern. Die drei Unicode-Kodierungsformen UTF-8, UTF-16 und UTF-32 schreiben jeden Codepunkt als eine oder mehrere Einheiten von 8, 16 oder 32 Bit. Die Spezifikation bezeichnet den Versuch, „UTF-8 gegenüber Unicode“ abzugrenzen, als Falle: Alle drei sind gleichwertige Möglichkeiten, dieselben Zeichen zu implementieren.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Unicode?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Software, die in jedem Ihrer Absatzmärkte natürlich klingt — einschließlich Datumsangaben, Währungen und der Sortierung von Namen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Englisch-dänisches i18n-System aufgebaut — 12,027 Nachrichten je Sprachversion, auf Englisch und Dänisch

Englisch-dänisches i18n-System aufgebaut Kontakt aufnehmen →

Warum es wichtig ist

Vor Unicode hatte jede Schrift eigene Codeseiten, und ein Byte bedeutete nichts, solange die verwendete Seite unbekannt war. In der westeuropäischen Codeseite windows-1252 steht das Byte E5 für å, in der kyrillischen windows-1251 steht dasselbe Byte für den Buchstaben е. Keine Seite konnte die Buchstaben der anderen aufnehmen: windows-1251 hat ґ und kein å, æ oder ø, windows-1252 umgekehrt. Ein Produkt auf Dänisch und Ukrainisch brauchte zwei Kodierungen und eine verlässliche Zuordnung des Textes zur jeweiligen Kodierung. Ein Satz mit Zitaten in beiden Sprachen passte in keine von beiden.

Unicode beseitigte diese Buchführung. Jede Sprache, in der ein Produkt veröffentlicht wird, und jeder Name, den ein Kunde in ein Formular eingibt, passen in einen Zeichensatz. Dieselbe Nummer bedeutet auf jedem lesenden System dasselbe Zeichen. Dadurch können Datenbank, API und Browser Text austauschen, ohne sich zuerst auf eine Codeseite zu einigen.

Der Standard klärt auch Fragen, die sonst jedes Programm auf seine Weise lösen würde. Er vereinheitlicht Zeichen über Sprachen hinweg: Das Komma ist ein Zeichen, ob es wie im Englischen Tausendergruppen trennt oder wie im Französischen als Dezimalzeichen dient. Die Bedeutung kommt also von der Locale, nicht vom Codepunkt. Über Schriftgrenzen hinweg vereinheitlicht er nicht: Das lateinische o, das kyrillische о und das griechische ο sehen in den meisten Schriftarten gleich aus und sind drei Zeichen, weil sie zu drei Schriften gehören und ältere Kodierungen sie bereits getrennt hielten. Außerdem hält er fest, welche Folgen äquivalent sind. Å kann als ein vorkomponiertes Zeichen gespeichert werden oder als A mit einem folgenden kombinierenden Ring. Der Standard definiert beide als kanonisch äquivalent, gleich anzuzeigen und zu interpretieren. Normalisierungsformen überführen beide in eine vereinbarte Schreibweise.

Diese Allgemeinheit hat ihren Preis, und ihn zahlt die programmierende Person. Ein Zeichen ist kein Byte, nicht immer eine Einheit eines Strings und nicht immer das, was Leser als einen Buchstaben zählen. Code, der diese drei Dinge gleichsetzt, besteht Tests mit englischen Daten und scheitert an Namen, Adressen und Nachrichten der Menschen, für die er geschrieben wurde.

Wie es in der Praxis aussieht

JavaScript macht die Schichten sichtbar. Ein String ist eine Folge von UTF-16-Codeeinheiten; das Aufteilen in ein Array mit dem Spread-Operator zerlegt ihn in Codepunkte:

```
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);
}
```

In Node.js 25 ausgeführt, gibt das Programm jedes Zeichen, seinen Codepunkt, seine Länge in UTF-16-Einheiten und seine Länge in Codepunkten aus:

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

Die letzte Zeile zeigt einen altitalischen Buchstaben. Sein Codepunkt liegt jenseits von U+FFFF, weshalb UTF-16 ihn als zwei Einheiten, ein Surrogatpaar, speichert. `length` meldet somit 2 für ein einziges Zeichen.

Äquivalenz wird auf dieselbe Weise sichtbar. Diese drei Strings werden alle als Å angezeigt:

```
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
```

Der erste ist der vorkomponierte Buchstabe U+00C5. Der zweite ist ein einfaches A, gefolgt von U+030A, dem kombinierenden Ring darüber. Der dritte ist U+212B, das Angströmzeichen, das die Unicode-Zeichendatenbank auf U+00C5 abbildet. Unverändert verglichen sind es drei verschiedene Strings; nach der Normalisierung nach NFC sind alle drei derselbe. Auf dem Bildschirm unterscheidet sie nichts. Eine Suche, ein eindeutiger Index oder eine Anmeldung, die rohe Strings vergleicht, sieht deshalb drei verschiedene Namen.

Diese Website hängt an einer weniger sichtbaren Stelle von Codepunkten ab. Die Notiz über die Unterstreichung, die jeden Buchstaben durchzog beschreibt die lateinischen und kyrillischen Ubuntu-Schriften für den Fließtext als sechs Schnitte, drei Strichstärken für Lateinisch und drei für Kyrillisch. So wird eine ukrainische Seite in derselben Schriftart gesetzt wie eine englische. Jeder Schnitt deklariert seinen Zeichenbereich. Diese Bereiche steuern die Schriftauswahl für die Zeichen auf der Seite; ein ausdrücklicher Preload kann außerdem eine Datei anfordern, bevor die normale Schriftermittlung beginnt. Die kyrillische Deklaration gibt ihren Bereich in Codepunkten an:

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

Der erste Bereich reicht von È bis ҵ und enthält neben dem übrigen kyrillischen Grundalphabet das ukrainische Є, I und Ї. Der zweite enthält nur Ґ ґ und ґ, die weiter hinten im kyrillischen Block stehen. Die Bereiche deklarieren, welcher Schriftschnitt für ein Zeichen infrage kommt; die Schriftdatei muss aber auch dessen Glyphe enthalten. Sie garantieren nicht, dass eine Seite nur Dateien eines einzigen Schriftsystems lädt. Englische und dänische Texte verwenden die lateinischen Schnitte, doch die ukrainische Sprachbezeichnung kann einen kyrillischen Schnitt benötigen. Die Website lädt den leichten kyrillischen Fließtextschnitt auf nichtkyrillischen Seiten für die Bildschirmausgabe ausdrücklich vor.

Wo es schiefgeht

Die Länge wird in Codeeinheiten gezählt. JavaScripts `length` zählt UTF-16-Einheiten. Deshalb zählt 𐐷 als zwei und ein A mit kombinierendem Ring ebenfalls als zwei, obwohl ein Leser einen Buchstaben sieht. Eine Grenze in Einheiten weist Text zurück, der eigentlich hineinpasst. Ein Schnitt an einer festen Position kann ein Zeichen zerlegen: `"A\u030A".slice(0, 1)` behält das A und entfernt den Ring. Wird 𐐷 halbiert, bleibt ein einzelnes Surrogat zurück, das der UTF-8-Kodierer durch U+FFFD, das Ersatzzeichen, ersetzt. Für jede Grenze muss feststehen, was sie zählt: Bytes, wenn der Speicher die Grenze setzt, und Zeichen aus Lesersicht, wenn ein Mensch sie setzt. `Intl.Segmenter` zählt Letztere und liefert für Århus mit kombinierendem Ring 5, wo `length` 6 liefert. Text sollte nur an den gemeldeten Grenzen abgeschnitten werden.

Gleicher Text wird Einheit für Einheit verglichen. Die drei Formen von Å oben sind für `===`, einen Hash und einen eindeutigen Index drei verschiedene Strings. Ein Kunde kann deshalb denselben Namen zweimal registrieren oder ein Dokument nicht über den eingegebenen Titel finden. Text sollte beim Eintritt ins System nach NFC normalisiert, normalisiert gespeichert und in normalisierter Form verglichen werden. Die Stabilitätsrichtlinie garantiert, dass ein unter einer Version normalisierter String auch unter jeder späteren normalisiert bleibt, sofern er nur Zeichen enthält, die die ursprüngliche Version kennt. Der normalisierende Code muss also so aktuell sein wie der Text.

Ähnlich aussehende Zeichen gelten als gleich. Das lateinische o, das kyrillische о und das griechische ο sind verschiedene Codepunkte, die die meisten Schriftarten gleich zeichnen. Zwei Kontonamen oder Produktcodes können daher identisch aussehen und sich in einem Zeichen unterscheiden. Unicode hält sie absichtlich getrennt; sein Standard für Sicherheitsmechanismen UTS #39 beschreibt, wie sich solche Täuschungen verringern lassen. Wenn ein Name eine

Person oder einen Zahlungsempfänger identifiziert, sollte pro Name nur ein Schriftsystem zugelassen oder der Name vor der Annahme gegen die Sicherheitsdaten des Standards geprüft werden.

Bytes werden mit der falschen Kodierung gelesen. Als UTF-8 gespeicherter und als windows-1252 gelesener Text macht aus blåbær blÃ¥bÃ¦r, weil jeder Zwei-Byte-Buchstabe als zwei Ein-Byte-Zeichen gelesen wird. Das Wort Unicode sagt nicht, wie die Bytes geschrieben wurden; das sagt erst die Kodierungsform. Sie sollte überall deklariert werden, wo Text eine Grenze überschreitet: im HTTP-Header, in der Seite, auf der Datenbankverbindung und im Dateiformat. Eine Datei ohne diese Angabe ist eine Frage an einen Menschen statt eine Gelegenheit für den Code zu raten.

Eine fehlende Glyphe wird dem Text angelastet. Erscheint ein Buchstabe als leeres Kästchen, fehlt der Schrift meist die Form für diesen Codepunkt; der Text selbst ist intakt. Weil Unicode die Formen den Schriftarten überlässt, muss jede Schriftart eines Produkts alle Zeichen abdecken, die dessen Sprachen brauchen: Є, І, Ї und Ґ ebenso wie A bis я. Andernfalls zeichnet der Browser sie mit irgendeiner verfügbaren Ersatzschrift. Die obige Notiz nennt das als Grund, überhaupt Kyrillisch auszuliefern: Ohne diese Schnitte fiele eine ukrainische Seite auf das zurück, was das System bietet. Jede Sprache sollte mit ihrem vollständigen Alphabet getestet werden statt mit einem Beispielsatz.

Namen werden gegen A bis Z geprüft. Ein Muster wie `^[A-Za-z' -]+$` akzeptiert O’Neill und weist Søren Ågård, Zoë und Ґалаґан zurück, genau die Namen, für die ein dänisches oder ukrainisches Produkt da ist. Stattdessen sollte anhand der Unicode-Eigenschaften auf Buchstaben geprüft werden: In JavaScript akzeptiert `/^[\p{L}\p{M}' -]+$/u` alle vier. Diese Eigenschaften stammen aus der Kopie der Unicode-Zeichendatenbank in der Laufzeitumgebung. Die hier für die Beispiele verwendete Node.js-25.2.1-Laufzeit meldet zum Beispiel Unicode 16.0. Das gilt für diesen konkreten Build, nicht garantiert für jede Version von 25.x. In späteren Unicode-Versionen hinzugefügte Buchstaben bestehen die Prüfung erst, wenn die Zeichendaten der Laufzeitumgebung aktualisiert wurden. Namen sollten großzügig validiert und schwierige Fälle von einem Menschen statt von einem Muster entschieden werden.

Weiterführende Lektüre

- About the Unicode Standard — die Beschreibung des Konsortiums selbst: der Standard, die Bestandteile einer Version und der Veröffentlichungsort der neuesten Version.
- The Unicode Standard, Chapter 2: General Structure — die zehn Entwurfsprinzipien, der Coderaum, die drei Kodierungsformen und die Regeln für äquivalente Folgen.
- Unicode Character Encoding Stability Policies — was sich nach der Kodierung eines Zeichens nie mehr ändern darf, von Codepunkt und Namen bis zur Normalisierung.
- Unicode 18.0.0 — die Ergänzungen der aktuellen Version und die gemeinsam versionierten technischen Standards, darunter Sortierung und Sicherheit.
- Node.js: process.versions — der Bericht über Komponentenversionen der Laufzeitumgebung, mit dem das Beispiel der Unicode-Daten hier an einen Node.js-Build statt an eine ganze Versionsreihe gebunden wird.
- CSS Fonts: character ranges — wie deklarierte Zeichenbereiche zusammen mit der tatsächlichen Glyphenabdeckung einer Schriftdatei bestimmen, welche Zeichen ein Schnitt darstellen kann.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Unicode?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/unicode/