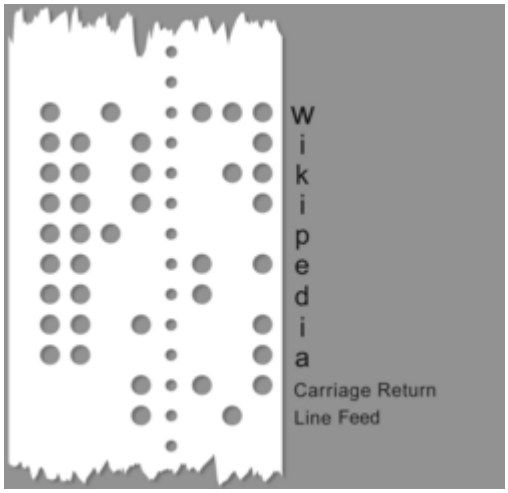
# Character encoding

By Contributors to Wikimedia projects

Published: 2001-09-20 · Archived: 2026-04-05 19:00:29 UTC




Punched tape with the word "Wikipedia" encoded in ASCII. Presence and absence of a hole represents 1 and 0, respectively; for example, W is encoded as `1010111`.

**Character encoding** is a convention of using a numeric value to represent each character of a writing script. Not only can a character set include natural language symbols, but it can also include codes that have meanings or functions outside of language, such as control characters and whitespace. Character encodings have also been defined for some constructed languages. When encoded, character data can be stored, transmitted, and transformed by a computer.[1] The numerical values that make up a character encoding are known as code points and collectively comprise a code space or a code page.

Early character encodings that originated with optical or electrical telegraphy and in early computers could only represent a subset of the characters used in languages, sometimes restricted to upper case letters, numerals and limited punctuation. Over time, encodings capable of representing more characters were created, such as ASCII, ISO/IEC 8859, and Unicode encodings such as UTF-8 and UTF-16.

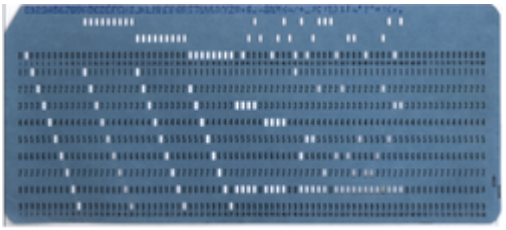
The most popular character encoding on the World Wide Web is UTF-8, which is used in 98.9% of surveyed web sites, as of January 2026.[2] In application programs and operating system tasks, both UTF-8 and UTF-16 are popular options.[3]

The history of character codes illustrates the evolving need for machine-mediated character-based symbolic information over a distance, using once-novel electrical means. The earliest codes were based upon manual and hand-written encoding and cyphering systems, such as Bacon's cipher, Braille, international maritime signal flags, and the 4-digit encoding of Chinese characters for a Chinese telegraph code (Hans Schjellerup, 1869). With the adoption of electrical and electro-mechanical techniques these earliest codes were adapted to the new capabilities and limitations of the early machines. The earliest well-known electrically transmitted character code, Morse

code, introduced in the 1840s, used a system of four "symbols" (short signal, long signal, short space, long space) to generate codes of variable length. Though some commercial use of Morse code was via machinery, it was often used as a manual code, generated by hand on a telegraph key and decipherable by ear, and persists in amateur radio and aeronautical use. Most codes are of fixed per-character length or variable-length sequences of fixed-length codes (e.g. Unicode).[4]

Common examples of character encoding systems include Morse code, the Baudot code, the American Standard Code for Information Interchange (ASCII) and Unicode. Unicode, a well-defined and extensible encoding system, has replaced most earlier character encodings, but the path of code development to the present is fairly well known.

The Baudot code, a five-bit encoding, was created by Émile Baudot in 1870, patented in 1874, modified by Donald Murray in 1901, and standardized by CCITT as International Telegraph Alphabet No. 2 (ITA2) in 1930. The name *baudot* has been erroneously applied to ITA2 and its many variants. ITA2 suffered from many shortcomings and was often improved by many equipment manufacturers, sometimes creating compatibility issues.



Hollerith 80-column punch card with EBCDIC character set

Herman Hollerith invented punch card data encoding in the late 19th century to analyze census data. Initially, each hole position represented a different data element, but later, numeric information was encoded by numbering the lower rows 0 to 9, with a punch in a column representing its row number. Later alphabetic data was encoded by allowing more than one punch per column. Electromechanical tabulating machines represented date internally by the timing of pulses relative to the motion of the cards through the machine.

When IBM went to electronic processing, starting with the IBM 603 Electronic Multiplier, it used a variety of binary encoding schemes that were tied to the punch card code. IBM used several binary-coded decimal (BCD) six-bit character encoding schemes, starting as early as 1953 in its 702[5] and 704 computers, and in its later 7000 Series and 1400 series, as well as in associated peripherals. Since the punched card code then in use was limited to digits, upper-case English letters and a few special characters, six bits were sufficient. These BCD encodings extended existing simple four-bit numeric encoding to include alphabetic and special characters, mapping them easily to punch-card encoding which was already in widespread use. IBM's codes were used primarily with IBM equipment. Other computer vendors of the era had their own character codes, often six-bit, such as the encoding used by the UNIVAC I.[6] They usually had the ability to read tapes produced on IBM equipment. IBM's BCD encodings were the precursors of their Extended Binary-Coded Decimal Interchange Code (usually abbreviated as EBCDIC), an eight-bit encoding scheme developed in 1963 for the IBM System/360 that featured a larger character set, including lower case letters.

In 1959, the U.S. military defined its Fieldata code, a six-or seven-bit code, introduced by the U.S. Army Signal Corps. While Fieldata addressed many of the then-modern issues (e.g. letter and digit codes arranged for machine collation), it fell short of its goals and was short-lived. In 1963 the first ASCII code was released (X3.4-1963) by the ASCII committee (which contained at least one member of the Fieldata committee, W. F. Leubbert), which addressed most of the shortcomings of Fieldata, using a simpler seven-bit code. Many of the changes were subtle, such as collatable character sets within certain numeric ranges. ASCII63 was a success, widely adopted by industry, and with the follow-up issue of the 1967 ASCII code (which added lower-case letters and fixed some "control code" issues) ASCII67 was adopted fairly widely. ASCII67's American-centric nature was somewhat addressed in the European ECMA-6 standard.[7] Eight-bit extended ASCII encodings, such as various vendor extensions and the ISO/IEC 8859 series, supported all ASCII characters as well as additional non-ASCII characters.

While trying to develop universally interchangeable character encodings, researchers in the 1980s faced the dilemma that, on the one hand, it seemed necessary to add more bits to accommodate additional characters, but on the other hand, for the users of the relatively small character set of the Latin alphabet (who still constituted the majority of computer users), those additional bits were a colossal waste of then-scarce and expensive computing resources (as they would always be zeroed out for such users). In 1985, the average personal computer user's hard disk drive could store only about 10 megabytes, and it cost approximately US$250 on the wholesale market (and much higher if purchased separately at retail),[8] so it was very important at the time to make every bit count.

The compromise solution that was eventually found and developed into Unicode[*vague*] was to break the assumption (dating back to telegraph codes) that each character should always directly correspond to a particular sequence of bits. Instead, characters would first be mapped to a universal intermediate representation in the form of abstract numbers called code points. Code points would then be represented in a variety of ways and with various default numbers of bits per character (code units) depending on context. To encode code points higher than the length of the code unit, such as above 256 for eight-bit units, the solution was to implement variable-length encodings where an escape sequence would signal that subsequent bits should be parsed as a higher code point.

The various terms related to character encoding are often used inconsistently or incorrectly.[9] Historically, the same standard would specify a repertoire of characters and how they were to be encoded into a stream of code units – usually with a single character per code unit. However, due to the emergence of more sophisticated character encodings, the distinction between terms has become important.

A character is the smallest unit of text that has semantic value.[9][10] In linguistics, this is called a grapheme and each of the various ways it may be written are called glyphs. (For example, the serif form g and the sans-serif form g are each a glyph of the grapheme ⟨g⟩, U+0067 g LATIN SMALL LETTER G.)

What constitutes a character varies between character encodings. For example, for letters with diacritics, there are two distinct approaches that can be taken to encode them. They can be encoded either as a single unified character (known as a precomposed character), or as separate characters that combine into a single glyph. The former simplifies the text handling system, but the latter allows any letter/diacritic combination to be used in text. Ligatures pose similar problems. Some writing systems, such as Arabic and Hebrew, have graphemes whose shape and joining depend on context.

A character set is a collection of characters used to represent text.[9][10] For example, the Latin alphabet and Greek alphabet are character sets.

### Coded character set

[edit]

A coded character set is a character set with each item uniquely mapped to a numberic value.[10]

This is also known as a code page,[9] although that term is generally antiquated. Originally, *code page* referred to a page number in an IBM manual that defined a particular character encoding.[11] Other vendors, including Microsoft, SAP, and Oracle Corporation, also published their own code pages, including notable Windows code page and code page 437. Despite no longer referring to specific pages in a manual, many character encodings are still identified to by the same number. Likewise, the term *code page* is still used to refer to character encoding.

In Unix and Unix-like systems, the term *charmap* is commonly used; usually in the larger context of locales.

IBM's Character Data Representation Architecture (CDRA) designates each entity with a coded character set identifier (CCSID), which is variously called a *charset*, *character set*, *code page*, or *CHARMAP*.[12]

### Character repertoire

[edit]

A character repertoire is a set of characters that can be represented by a particular coded character set.[10][13] The repertoire may be closed, meaning that no additions are allowed without creating a new standard (as is the case with ASCII and most of the ISO-8859 series); or it may be open, allowing additions (as is the case with Unicode and to a limited extent Windows code pages).[13]

A code point is the value or position of a character in a coded character set.[10] A code point is represented by a sequence of code units. The mapping is defined by the encoding. Thus, the number of code units required to represent a code point depends on the encoding:

- UTF-8: code points map to a sequence of one, two, three or four code units.
- UTF-16: code units are twice as long as 8-bit code units. Therefore, any code point with a scalar value less than U+10000 is encoded with a single code unit. Code points with a value U+10000 or higher require two code units each. These pairs of code units have a unique term in UTF-16: "Unicode surrogate pairs".
- UTF-32: the 32-bit code unit is large enough that every code point is represented as a single code unit.
- GB 18030: multiple code units per code point are common, because of the small code units. Code points are mapped to one, two, or four code units.[14]

Code space is the range of numerical values spanned by a coded character set.[10][12]

A code unit is the minimum bit combination that can represent a character in a character encoding (in computer science terms, it is the word size of the character encoding).[10][12] Common code units include 7-bit, 8-bit, 16-bit,

and 32-bit. In some encodings, some characters are encoded as multiple code units.

For example:

- ASCII: 7 bits
- UTF-8, EBCDIC and GB 18030: 8 bits
- UTF-16: 16 bits
- UTF-32: 32 bits

Unicode and its parallel standard, the ISO/IEC 10646 Universal Character Set, together constitute a unified standard for character encoding. Rather than mapping characters directly to bytes, Unicode separately defines a coded character set that maps characters to unique natural numbers (code points), how those code points are mapped to a series of fixed-size natural numbers (code units), and finally how those units are encoded as a stream of octets (bytes). The purpose of this decomposition is to establish a universal set of characters that can be encoded in a variety of ways. To describe the model precisely, Unicode uses existing terms and defines new terms. [12]

### Abstract character repertoire

[edit]

An abstract character repertoire (ACR) is the full set of abstract characters that a system supports. Unicode has an open repertoire, meaning that new characters will be added to the repertoire over time.

### Coded character set

[edit]

A coded character set (CCS) is a function that maps characters to *code points* (each code point represents one character). For example, in a given repertoire, the capital letter "A" in the Latin alphabet might be represented by the code point 65, the character "B" by 66, and so on. Multiple coded character sets may share the same character repertoire; for example ISO/IEC 8859-1 and IBM code pages 037 and 500 all cover the same repertoire but map them to different code points.

### Character encoding form

[edit]

A character encoding form (CEF) is the mapping of code points to *code units* to facilitate storage in a system that represents numbers as bit sequences of fixed length (i.e. practically any computer system). For example, a system that stores numeric information in 16-bit units can only directly represent code points 0 to 65,535 in each unit, but larger code points (say, 65,536 to 1.4 million) could be represented by using multiple 16-bit units. This correspondence is defined by a CEF.

### Character encoding scheme

[edit]

A character encoding scheme (CES) is the mapping of code units to a sequence of octets to facilitate storage on an octet-based file system or transmission over an octet-based network. Simple character encoding schemes include UTF-8, UTF-16BE, UTF-32BE, UTF-16LE, and UTF-32LE; compound character encoding schemes, such as UTF-16, UTF-32 and ISO/IEC 2022, switch between several simple schemes by using a byte order mark or escape sequences; compressing schemes try to minimize the number of bytes used per code unit (such as SCSU and BOCU).

Although UTF-32BE and UTF-32LE are simpler CESes, most systems working with Unicode use either UTF-8, which is backward compatible with fixed-length ASCII and maps Unicode code points to variable-length sequences of octets, or UTF-16BE,[*citation needed*] which is backward compatible with fixed-length UCS-2BE and maps Unicode code points to variable-length sequences of 16-bit words. See comparison of Unicode encodings for a detailed discussion.

### Higher-level protocol

[edit]

There may be a higher-level protocol which supplies additional information to select the particular variant of a Unicode character, particularly where there are regional variants that have been 'unified' in Unicode as the same character. An example is the XML attribute xml:lang.

The Unicode model uses the term "character map" for other systems which directly assign a sequence of characters to a sequence of bytes, covering all of the CCS, CEF and CES layers.[12]

### Code point documentation

[edit]

A character is commonly documented as 'U+' followed by its code point value in hexadecimal. The range of valid code points (the code space) for the Unicode standard is U+0000 to U+10FFFF, inclusive, divided in 17 planes, identified by the numbers 0 to 16. Characters in the range U+0000 to U+FFFF are in plane 0, called the Basic Multilingual Plane (BMP). This plane contains the most commonly used characters. Characters in the range U+10000 to U+10FFFF in the other planes are called supplementary characters.

The following table includes examples of code points:

| Character | Code point | Grapheme |
|---|---|---|
| Latin A | U+0041 | A |
| Latin sharp S | U+00DF | ß |
| Han for East | U+6771 | 東 |
| Ampersand | U+0026 | & |

| | | |
|---|---|---|
| Inverted exclamation mark | U+00A1 | ¡ |
| Section sign | U+00A7 | § |

Consider, "a͟bcə" – a string containing a Unicode combining character (U+0332 ◌̲ COMBINING LOW LINE to underline the ⟨b⟩) as well as a supplementary character (U+10400 𐐀 DESERET CAPITAL LETTER LONG I). This string has several Unicode representations which are logically equivalent, yet while each is suited to a diverse set of circumstances or range of requirements:

- Four composed characters:
  `a` , `b̲` , `c` , `ə`
- Five graphemes:
  `a` , `b` , `_` , `c` , `ə`
- Five Unicode code points:
  `U+0061` , `U+0062` , `U+0332` , `U+0063` , `U+10400`
- Five UTF-32 code units (32-bit integer values):
  `0x00000061` , `0x00000062` , `0x00000332` , `0x00000063` , `0x00010400`
- Six UTF-16 code units (16-bit integers)
  `0x0061` , `0x0062` , `0x0332` , `0x0063` , `0xD801` , `0xDC00`
- Nine UTF-8 code units (8-bit values, or bytes)
  `0x61` , `0x62` , `0xCC` , `0xB2` , `0x63` , `0xF0` , `0x90` , `0x90` , `0x80`

Note in particular that ə is represented with either one 32-bit value (UTF-32), two 16-bit values (UTF-16), or four 8-bit values (UTF-8). Although each of those forms uses the same total number of bits (32) to represent the grapheme, it is not obvious how the actual numeric byte values are related.

To support environments using multiple character encodings, software has been developed to translate text between character encoding schemes, a process known as transcoding. Notable software includes:

- Web browser – Modern browsers feature automatic character encoding detection
- iconv – Program and standardized API to convert encodings
- luit – Program that converts encoding of input and output to programs running interactively
- International Components for Unicode – A set of C and Java libraries for charset conversion
- Encoding.Convert – .NET API[15]
- MultiByteToWideChar/WideCharToMultiByte – Windows API functions for converting between ANSI and Unicode[16][17]

## Common character encodings

[edit]

This section **needs expansion** with: Popularity and comparison:

- Statistics on popularity

- Especially, a comparison of the advantages and disadvantages of the few 3–5 most common character encodings (e.g. UTF-8, UTF-16 and UTF-32). You can help by adding missing information. *(June 2024)*

The most used character encoding on the web is UTF-8, used in 98.9% of surveyed web sites, as of January 2026.[2] In application programs and operating system tasks, both UTF-8 and UTF-16 are popular options.[3][18]

- ISO 646
  - ASCII
- EBCDIC
- ISO 8859:
  - ISO 8859-1 Western Europe
  - ISO 8859-2 Western and Central Europe
  - ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
  - ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
  - ISO 8859-5 Cyrillic alphabet
  - ISO 8859-6 Arabic
  - ISO 8859-7 Greek
  - ISO 8859-8 Hebrew
  - ISO 8859-9 Western Europe with amended Turkish character set
  - ISO 8859-10 Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
  - ISO 8859-11 Thai
  - ISO 8859-13 Baltic languages plus Polish
  - ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
  - ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
  - ISO 8859-16 Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872
- MS-Windows character sets:
  - Windows-1250 for Central European languages that use Latin script, (Polish, Czech, Slovak, Hungarian, Slovene, Serbian, Croatian, Bosnian, Romanian and Albanian)
  - Windows-1251 for Cyrillic alphabets
  - Windows-1252 for Western languages
  - Windows-1253 for Greek
  - Windows-1254 for Turkish
  - Windows-1255 for Hebrew
  - Windows-1256 for Arabic
  - Windows-1257 for Baltic languages
  - Windows-1258 for Vietnamese

- Mac OS Roman
- KOI8-R, KOI8-U, KOI-7
- MIK
- ISCII
- TSCII
- VISCII
- JIS X 0208 is a widely deployed standard for Japanese character encoding that has several encoding forms.
  - Shift JIS (Microsoft Code page 932 is a dialect of Shift_JIS)
  - EUC-JP
  - ISO-2022-JP
- JIS X 0213 is an extended version of JIS X 0208.
  - Shift_JIS-2004
  - EUC-JIS-2004
  - ISO-2022-JP-2004
- Chinese Guobiao
  - GB 2312
  - GBK (Microsoft Code page 936)
  - GB 18030
- Taiwan Big5 (a more famous variant is Microsoft Code page 950)
  - Hong Kong HKSCS
- Korean
  - KS X 1001 is a Korean double-byte character encoding standard
  - EUC-KR
  - ISO-2022-KR
- Unicode (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane')
  - UTF-8
  - UTF-16
  - UTF-32
- ANSEL or ISO/IEC 6937

- Percent-encoding – Method of encoding characters in a URI
- Alt code – Input method
- Character encodings in HTML – Use of encoding systems for international characters in HTML
- Charset sniffing – Practice of deducing the file type of a bitstream
- Category:Character encoding – articles related to character encoding in general
- Category:Character sets – articles detailing specific character encodings
- Hexadecimal – Base-16 numeric representation
- Mojibake – Garbled text as a result of incorrect character encodings
- Mojikyō – Character encoding scheme
- Presentation layer – Sixth layer of the OSI model of telecommunications
- Tofu (symbol) – Mark shown when a codepoint cannot be resolved
  - .notdef, a character within a font to be used for this purpose
- TRON (encoding) – Multi-byte character encoding

- Universal Character Set characters – Complete list of the characters available on most computers


1. ^ *"Character Encoding Definition". The Tech Terms Dictionary. 24 September 2010.*
2. ^ Jump up to: **a** **b** *"Usage Survey of Character Encodings broken down by Ranking". W3Techs. Retrieved 1 January 2026.*
3. ^ Jump up to: **a** **b** *"Charset". Android Developers. Retrieved 2 January 2021. "Android note: The Android platform default is always UTF-8."*
4. ^ *Tom Henderson (17 April 2014). "Ancient Computer Character Code Tables – and Why They're Still Relevant". Smartbear. Archived from the original on 30 April 2014. Retrieved 29 April 2014.*
5. ^ *"IBM Electronic Data-Processing Machines Type 702 Preliminary Manual of Information" (PDF). 1954. p. 80. 22-6173-1. Archived (PDF) from the original on 9 October 2022 – via bitsavers.org.*
6. ^ *"UNIVAC System" (PDF) (reference card).*
7. ^ *Tom Jennings (20 April 2016). "An annotated history of some character codes". Sensitive Research. Retrieved 1 November 2018.*
8. ^ *Strelho, Kevin (15 April 1985). "IBM Drives Hard Disks to New Standards". InfoWorld. Popular Computing Inc. pp. 29–33. Retrieved 10 November 2020.*
9. ^ Jump up to: **a** **b** **c** **d** *Shawn Steele (15 March 2005). "What's the difference between an Encoding, Code Page, Character Set and Unicode?". Microsoft Docs.*
10. ^ Jump up to: **a** **b** **c** **d** **e** **f** **g** *"Glossary of Unicode Terms". Unicode Consortium.*
11. ^ *"VT510 Video Terminal Programmer Information". Digital Equipment Corporation (DEC). 7.1. Character Sets - Overview. Archived from the original on 26 January 2016. Retrieved 15 February 2017. "In addition to traditional DEC and ISO character sets, which conform to the structure and rules of ISO 2022, the VT510 supports a number of IBM PC code pages (page numbers in IBM's standard character set manual) in PCTerm mode to emulate the console terminal of industry-standard PCs."*
12. ^ Jump up to: **a** **b** **c** **d** **e** *Whistler, Ken; Freytag, Asmus (11 November 2022). "UTR#17: Unicode Character Encoding Model". Unicode Consortium. Retrieved 12 August 2023.*
13. ^ Jump up to: **a** **b** *"Chapter 3: Conformance". The Unicode Standard Version 15.0 – Core Specification (PDF). Unicode Consortium. September 2022. ISBN 978-1-936213-32-0.*
14. ^ *"Terminology (The Java Tutorials)". Oracle. Retrieved 25 March 2018.*
15. ^ *"Encoding.Convert Method". Microsoft .NET Framework Class Library.*
16. ^ *"MultiByteToWideChar function (stringapiset.h)". Microsoft Docs. 13 October 2021.*
17. ^ *"WideCharToMultiByte function (stringapiset.h)". Microsoft Docs. 9 August 2022.*
18. ^ *Galloway, Matt (9 October 2012). "Character encoding for iOS developers. Or UTF-8 what now?". Matt Galloway. Retrieved 2 January 2021. "in reality, you usually just assume UTF-8 since that is by far the most common encoding."*

- *Mackenzie, Charles E. (1980). Coded Character Sets, History and Development (PDF). The Systems Programming Series (1 ed.). Addison-Wesley Publishing Company, Inc. ISBN 978-0-201-14460-4. LCCN 77-90165. Archived (PDF) from the original on May 26, 2016. Retrieved August 25, 2019.*


Wikimedia Commons has media related to Encodings.

- Character sets registered by Internet Assigned Numbers Authority (IANA)
- Characters and encodings, by Jukka Korpela
- Unicode Technical Report #17: Character Encoding Model
- Decimal, Hexadecimal Character Codes in HTML Unicode – Encoding converter
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) by Joel Spolsky (Oct 10, 2003)

---