Character encoding

In computing **character encoding** is used to represent a repertoire of characters by some kind of encoding system.[1] Depending on the abstraction level and context, corresponding code points and the resulting code space may be regarded as bit patterns, octets, natural numbers, electrical pulses, etc. A character encoding is used in computation, data storage, and transmission of textual data. "Character set", "character map", "codeset" and "code page" are related, but not identical, terms.

Early character codes associated with the optical or electrical telegraph could only represent a subset of the characters used in written languages, sometimes restricted to upper case letters, numerals and some punctuation only. The low cost of digital representation of data in modern computer systems allows more elaborate character codes (such as Unicode) which represent most of the characters used in many written languages. Character encoding using internationally accepted standards permits worldwide interchange of text in electronic form.

Contents

History

Terminology

Unicode encoding model

Character sets, character maps and code pages

Character encoding translation

See also

Common character encodings

References

Further reading

External links

History

Early binary repertoires include Bacon's cipher, Braille, International maritime signal flags, and the 4-digit encoding of Chinese characters for a Chinese telegraph code (Hans Schjellerup, 1869). Common examples of character encoding systems include Morse code, the Baudot code, the American Standard Code for Information Interchange (ASCII) and Unicode.[2]

Morse code was introduced in the 1840s and is used to encode each letter of the Latin alphabet, each Arabic numeral, and some other characters via a series of long and short presses of a telegraph key. Representations of characters encoded using Morse code varied in length.

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.

Fieldata, a six- or seven-bit code, was introduced by the U.S. Army Signal Corps in the late 1950s.

IBM's Binary Coded Decimal (BCD) was a six-bit encoding scheme used by IBM in as early as 1959 in its 1401 and 1620 computers, and in its 7000 Series (for example, 704, 7040, 709 and 7090 computers), as well as in associated peripherals. BCD extended existing simple four-bit numeric encoding to include alphabetic and special characters, mapping it easily to punch-card encoding which was already in widespread use. It was the precursor to EBCDIC.

ASCII was introduced in 1963 and is a seven-bit encoding scheme used to encode letters, numerals, symbols, and device control codes as fixed-length codes usingintegers.

IBM's Extended Binary Coded Decimal Interchange Code (usually abbreviated as EBCDIC) is an eight-bit encoding scheme developed in 1963.

The limitations of such sets soon became apparent, and a number of *ad hoc* methods were developed to extend them. The need to support more writing systems for different languages, including theCJK family of East Asian scripts, required support for a far larger number of characters and demanded a systematic approach to character encoding rather than the previous ad hoc approaches.

In 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).

The compromise solution that was eventually found and developed into Unicode 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 8-bit units, the solution was to implement variable-width encodings where an escape sequence would signal that subsequent bits should be parsed as a higher code point.

Terminology

Terminology related to code unit:

- A *character* is a minimal unit of text that has semantic value.
- A *character set* is a collection of characters that might be used by multiple languages.

Example: The Latin character set is used by English and most European languages, though the Greek character set is used only by the Greek language.

- A *coded character set* is a character set in which each character corresponds to a unique number
- A *code point* of a coded character set is any allowed value in the character set.
- A *code unit* is a bit sequence used to encode each character of a repertoire within a given encoding form.

Character repertoire (the abstract set of characters)

The character repertoire is an abstract set of more than one million characters found in a wide variety of scripts including *Latin, Cyrillic, Chinese, Korean, Japanese, Hebrew, and Aramaic.*

Other symbols such as musical notation are also included in the character repertoire. Both the Unicode and GB18030 standards have a character repertoire. As new characters are added to one standard, the other standard also adds those characters, to maintain parity

The code unit size is equivalent to the bit measurement for the particular encoding:

- A code unit in US-ASCII consists of 7 bits;
- A code unit in UTF-8, EBCDIC and GB18030 consists of 8 bits;
- A code unit in UTF-16 consists of 16 bits;
- A code unit in UTF-32 consists of 32 bits.

Example of a code unit: Consider a *string* of the letters "abc" followed by `U+10400` 𐐀 DESERET CAPITAL LETTER LONG I (represented with 1 char32_t, 2 char16_t or 4 char8_t). That string contains:

- four characters;
- four code points
- either:

  four code units in UTF-32 (00000061, 00000062, 00000063, 00010400)
  five code units in UTF-16 (0061, 0062, 0063, d801, dc00), or
  seven code units in UTF-8 (61, 62, 63, f0, 90, 90, 80).

To express a character in Unicode, the hexadecimal value is prefixed with the string 'U+'. The range of valid code points 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 the plane 0, called the *Basic Multilingual Plane* (BMP). This plane contains most commonly-used characters. Characters in the range U+10000 to U+10FFFF in the other planes are called *supplementary characters*.

The following table shows examples of code point values:

Character	Unicode code point	Glyph
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	§

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 are 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.
- *GB18030:* 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.[3]

Unicode encoding model

Unicode and its parallel standard, the ISO/IEC 10646 Universal Character Set, together constitute a modern, unified character encoding. Rather than mapping characters directly to octets (bytes), they separately define what characters are available, corresponding natural numbers (code points), how those numbers are encoded as a series of fixed-size natural numbers (code units), and finally how those units are encoded as a stream of octets. The purpose of this decomposition is to establish a universal set of characters that can be encoded in a variety of ways.[4] To describe this model correctly requires more precise terms than "character set" and "character encoding." The terms used in the modern model follow:[4]

A **character repertoire** is the full set of abstract characters that a system supports. The repertoire may be closed, i.e. 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 the Windows code pages). The characters in a given repertoire reflect decisions that have been made about how to divide writing systems into basic information units. The basic variants of the Latin, Greek and Cyrillic alphabets can be broken down into letters, digits, punctuation, and a few *special characters* such as the space, which can all be arranged in simple linear sequences that are displayed in the same order they are read. But even with these alphabets, diacritics pose a complication: they can be regarded either as part of a single character containing a letter and diacritic (known as a precomposed character), or as separate characters. The former allows a far simpler text handling system but the latter allows any letter/diacritic combination to be used in text. Ligatures pose similar problems. Other writing systems, such as Arabic and Hebrew, are represented with more complex character repertoires due to the need to accommodate things like bidirectional text and glyphs that are joined together in different ways for different situations.

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" to 66, and so on. Multiple coded character sets may share the same 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.

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.

Next, 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 or UTF-32LE; compound character encoding schemes, such as UTF-16, UTF-32 and ISO/IEC 2022, switch between several simple schemes by using byte order marks or escape sequences; compressing schemes try to minimise the number of bytes used per code unit (such as SCSU, BOCU, and Punycode).

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

Finally, 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 historical systems which directly assign a sequence of characters to a sequence of bytes, covering all of CCS, CEF and CES layers.[4]

Character sets, character maps and code pages

Historically, the terms "character encoding", "character map", "character set" and "code page" were synonymous in computer science, as 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. But now the terms have related but distinct meanings,[5] due to efforts by standards bodies to use precise terminology when writing about and unifying many different encoding systems.[4] Regardless, the terms are still used interchangeably, with *character set* being nearly ubiquitous.

A "**code page**" usually means a byte-oriented encoding, but with regard to some suite of encodings (covering different scripts), where many characters share the same codes in most or all those code pages. Well-known code page suites are "Windows" (based on Windows-1252) and "IBM"/"DOS" (based on code page 437), see Windows code page for details. Most, but not all, encodings referred to as code pages are single-byte encodings (but see octet on byte size.)

IBM's Character Data Representation Architecture (CDRA) designates with coded character set identifiers (CCSIDs) and each of which is variously called a "charset", "character set", "code page", or "CHARMAP".[4]

The term "code page" does not occur in Unix or Linux where "charmap" is preferred, usually in the larger context of locales.

Contrasted to CCS above, a "character encoding" is a map from abstract characters to code words. A "character set" in HTTP (and MIME) parlance is the same as a character encoding (but not the same as CCS).

"Legacy encoding" is a term sometimes used to characterize old character encodings, but with an ambiguity of sense. Most of its use is in the context of Unicodification, where it refers to encodings that fail to cover all Unicode code points, or, more generally, using a somewhat different character repertoire: several code points representing one Unicode character,[6] or versa (see e.g. code page 437).

Some sources refer to an encoding as *legacy* only because it preceded Unicode.[7] All Windows code pages are usually referred to as legacy, both because they antedate Unicode and because they are unable to represent all 2^{21} possible Unicode code points.

Character encoding translation

As a result of having many character encoding methods in use (and the need for backward compatibility with archived data), many computer programs have been developed to translate data between encoding schemes as a form of data transcoding. Some of these are cited below.

Cross-platform:

- Web browsers – most modern web browsers feature automatic character encoding detection. On Firefox 3, for example, see the View/Character Encoding submenu.
- iconv – program and standardized API to convert encodings
- luit – program that converts encoding of input and output to programs running interactively
- convert_encoding.py – Python based utility to convert text files between arbitrary encodings and line endings.[8]
- decodeh.py – algorithm and module to heuristically guess the encoding of a string.[9]
- International Components for Unicode – A set of C and Java libraries to perform charset conversion. uconv can be used from ICU4C.
- chardet – This is a translation of the Mozilla automatic-encoding-detection code into the Python computer language.
- The newer versions of the Unix file command attempt to do a basic detection of character encoding (also available on Cygwin).
- charset – C++ template library with simple interface to convert between C++/user-defined streams. charset defined many character-sets and allows you to use Unicode formats with support of endianness.

Unix-like:

- cmv – simple tool for transcoding filenames.[10]
- convmv – convert a filename from one encoding to another.[11]
- cstocs – convert file contents from one encoding to another for the Czech and Slovak languages.
- enca – analyzes encodings for given text files.[12]
- recode – convert file contents from one encoding to another.[13]
- utrac – convert file contents from one encoding to another.[14]

Windows:

- Encoding.Convert – .NET API[15]
- MultiByteToWideChar/WideCharToMultiByte – Convert from ANSI to Unicode & Unicode to ANSI[16]
- cscvt – character set conversion tool[17]
- enca – analyzes encodings for given text files.[18]

See also

- Alt code
- Character encodings in HTML
- Character encoding – articles related to character encoding in general
- Character sets – articles detailing specific character encodings
- Hexadecimal representations
- *Mojibake* – character set mismap.
- Mojikyo – a system ("glyph set") that includes over 100,000 Chinese character drawings, modern and ancient, popular and obscure.
- TRON, part of the TRON project, is an encoding system that does not use Han Unification; instead, it uses "control codes" to switch between 16-bit "planes" of characters.
- Universal Character Set characters
- Charset sniffing – used in some applications when character encoding metadata is not available

Common character encodings

- ISO 646
  - ASCII
- EBCDIC
  - CP37
  - CP930
  - CP1047
- 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, KOI7
- 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

References

1. Definition from The Tech Terms Dictionary (http://techterms.com/definition/characterencoding)
2. Tom Henderson (17 April 2014). "Ancient Computer Character Code Tables – and Why They're Still Relevant" (http://blog.smartbear.com/development/ancient-computer-character-code-tables-and-why-theyre-still-relevant/) Smartbear. Retrieved 29 April 2014.
3. http://docs.oracle.com/javase/tutorial/i18n/text/terminology.html
4. "Unicode Technical Report #17: Unicode Character Encoding Model" (http://www.unicode.org/reports/tr17/) 11 November 2008. Retrieved 8 August 2009.
5. Shawn Steele (15 March 2005). "What's the difference between an Encoding, Code Page, Character Set and Unicode?" (https://blogs.msdn.microsoft.com/shawnste/2005/03/15/whats-the-difference-between-an-encoding-code-page-character-set-and-unicode/) *MSDN*.
6. "Processing database information using Unicode, a case study" (http://www.basistech.co.jp/knowledge-center/database/uc-trillium-2.ppt) Archived (https://web.archive.org/web/20060617193918/http://www.basistech.co.jp/knowledge-center/database/uc-trillium-2.ppt) 17 June 2006 at the Wayback Machine.
7. Constable, Peter (13 June 2001). "Character set encoding basics" (http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=IWS-Chapter03#79e846db) *Implementing Writing Systems: An introduction*. SIL International. Retrieved 19 March 2010.
8. convert_encoding.py (https://github.com/goerz/convert_encoding.py)
9. Decodeh – heuristically decode a string or text file (http://gizmojo.org/code/decodeh/) Archived (https://web.archive.org/web/20080108123255/http://gizmojo.org/code/decodeh/) 8 January 2008 at the Wayback Machine.
10. CharsetMove – Simple Tool for Transcoding Filenames (http://cmv.fossrec.com)
11. Convmv – converts filenames from one encoding to another (http://www.j3e.de/linux/convmv/man/)
12. Extremely Naive Charset Analyser (http://directory.fsf.org/project/enca/)
13. Recode – GNU project – Free Software Foundation (FSF) (https://www.gnu.org/software/recode/recode.html)
14. Utrac Homepage (http://utrac.sourceforge.net/)
15. Microsoft .NET Framework Class Library – Encoding.Convert Method (http://msdn.microsoft.com/en-us/library/system.text.encoding.convert(VS.71).aspx)
16. MultiByteToWideChar/WideCharToMultiByte – Convert from ANSI to Unicode & Unicode to ANSI (http://support.microsoft.com/kb/138813)
17. Kalytta's Character Set Converter (http://www.cscvt.de/)
18. Extremely Naive Charset Analyser (http://www.john.geek.nz/2010/02/enca-binary-compiled-for-32-bit-windows/)

Further reading

- * Mackenzie, Charles E. (1980). *Coded Character Sets, History and Development*. *The Systems Programming Series* (1 ed.). Addison-Wesley Publishing Company Inc. ISBN 0-201-14460-3. LCCN 77-90165.

External links

- Character Encoding ASCII, ANSI and Unicode
- 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