Unicode

Péter Jeszenszky

September 10, 2024

What is Unicode?

- Universal character encoding standard for written characters and text.
- Covers all the characters for all the writing systems of the world, modern and ancient.
- It also includes technical symbols, punctuations, and many other characters used in writing text.
- Widely used and supported.

Coverage

Examples:

- Cherokee: https://www.unicode.org/charts/PDF/U13A0.pdf
- Imperial Aramaic: https://www.unicode.org/charts/PDF/U10840.pdf
- Old Hungarian: https://www.unicode.org/charts/PDF/U10C80.pdf
- Egyptian hieroglyphs: https://www.unicode.org/charts/PDF/U13000.pdf
- Emoticons: https://www.unicode.org/charts/PDF/U1F600.pdf
- Alchemical symbols: https://www.unicode.org/charts/PDF/U1F700.pdf

Emojis (1)

- Emoji are “picture characters” originally associated with mobile phone usage in Japan, but are now popular worldwide.
- The word emoji comes from the Japanese word 絵文字 (emoji), whereby 絵 (e) means picture and 文字 (moji) means character.
- Emoji are pictographs (pictorial symbols) that are typically presented in a colorful form and used inline in text.
- They represent things such as faces, weather, vehicles and buildings, food and drink, animals and plants, or icons that represent emotions, feelings, or activities.
- See: Unicode Frequently Asked Questions (FAQ)—Emoji and Pictographs

Emojis (2)

- Unicode contains 3,700+ emoji.
  - The total number of emoji: https://unicode.org/emoji/charts/emoji-counts.html
- Further information:
  - https://home.unicode.org/emoji/
  - https://unicode.org/emoji/techindex.html
  - https://unicode.org/emoji/charts/full-emoji-list.html
  - Emojipedia

Emojis (3)

Fun facts:

- The general recommendation is that emoji depicting people or body parts should have as neutral or generic depictions as possible with respect to physical appearance.
- For example, non-realistic skin tone colors (e.g., orange) should be used.
- However, many emoji may be followed by an emoji modifier character specifying one of five possible skin tones.

Standard

- Developed by the Unicode Consortium, a non-profit organization.
  - See: https://www.unicode.org/consortium/consort.html
- The current standard is version 15.1.0 released on September 12, 2023.
  - See: https://www.unicode.org/versions/latest/
- The next version will be 16.0.0, planned for release on September 10, 2024.
  - Introduces a total of 5185 new characters.
  - See: https://www.unicode.org/versions/beta-16.0.0.html

Universal Coded Character Set (UCS) (1)

- A standard character set defined by ISO.
- Current standard: *ISO/IEC 10646:2020 Information technology—Universal Coded Character Set* https://www.iso.org/standard/76835.html
  - Can be downloaded from here: https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

Universal Coded Character Set (UCS) (2)

- Developed in conjunction with Unicode.
  - The characters and their code points are the same in both standards.
  - The difference is that Unicode imposes additional constraints on implementations to ensure that they treat characters uniformly across platforms and applications.
- Further information:
  - *Frequently Asked Questions—Unicode and ISO 10646*
    https://www.unicode.org/faq/unicode_iso.html

Basic Concepts

- **Codespace**: the range of integers used to code the characters.
- **Code point**: an element of the codespace, i.e., an integer encoding a character.

Code Points

When referring to code points, the usual practice is to refer to them by their numeric value expressed in hexadecimal using four to six digits, with a `U+` prefix.

- Leading zeros are omitted, unless the code point would have fewer than four hexadecimal digits.
- Examples: U+0020, U+265F, U+130E0

Properties

Unicode associates a rich set of semantics with characters (code points), these semantics are defined by character properties.

- Unicode identifies more than 100 different character properties, such as:
  - Name
  - General category (letter, number, symbol, punctuation, ⋯)
  - Case (uppercase, lowercase, titlecase)
- The Unicode Character Database (UCD) contains character properties: https://unicode.org/ucd/

Character Names

Each character is identified by a unique name, such as:

- U+0041: LATIN CAPITAL LETTER A (A) https://www.fileformat.info/info/unicode/char/0041/index.htm
- U+2605: BLACK STAR (★) https://www.fileformat.info/info/unicode/char/2605/index.htm
- U+1F63A: SMILING CAT FACE WITH OPEN MOUTH (😺) https://www.fileformat.info/info/unicode/char/1f63a/index.htm

Characters and Glyphs

- The character identified by a Unicode code point is an abstract entity, such as *LATIN CAPITAL LETTER A* or *BENGALI DIGIT FIVE*.
- A visual representation of a character is called a **glyph**.
- The Unicode standard does not define glyph images.
- The visual appearance of characters on a device (e.g., screen or printer) is fully left to the software or hardware responsible for rendering characters.

Codespace

- The range of integers from 0_{16} to $10FFFF_{16}$.
- The total number of code points is 1,114,112 of which 149,186 are used currently.
- Character code charts: https://www.unicode.org/charts/

Planes and Blocks

- The codespace is divided into planes, each of which contains 65 536 (2^{16}) code points.
  - The last four hexadecimal digits in each code point indicate a character's position inside a plane. The remaining digits indicate the plane.
    - For example, U+130F7 is found at location $30F7_{16}$ in Plane 1.
- The total number of planes is 17 (0_{16}, $\cdots$, 10_{16}).
- Planes are divided into non-overlapping blocks.
  - Blocks are named ranges, where the number of code points in a block is always a multiply of 16.
  - Characters used in a single writing system may be found in several different blocks.

Basic Multilingual Plane (BMP)

- The plane containing the first 65,536 code points (range U+0000 - U+FFFF) (Plane 0).
- Contains the common-use characters for all the modern scripts of the world as well as many historical and rare characters.
- By far the majority of all Unicode characters for almost all textual data can be found in the BMP.

Character Encodings

- Character encodings defined by the Unicode standard:
  - UTF-8
  - UTF-16
  - UTF-32
- All three encoding forms can be used to represent the full range of Unicode characters.
- The abbreviation UTF stands for Unicode transformation format.

UTF-32

- Each code point is represented by 4 bytes (fixed-width character encoding form).
- The simplest Unicode encoding form.
- The most efficient in terms of processing.
- The least efficient encoding in terms of the number of bytes used.

UTF-16

- Each code point is represented by 2 or 4 bytes (variable-width character encoding form):
  - Code points in the BMP are represented by using 2 bytes, for all other code points 4 bytes are used.
- Optimizes the representation of characters in the BMP:
  - For code points in the BMP can effectively be treated as if it were a fixed-width encoding form.
- Maintains a balance between efficient access and economical use of storage.

UTF-8 (1)

- Each code point is represented by using from 1 to 4 bytes (variable-width character encoding form):
  - Code points in the range U+0000-U+007F are represented by a single byte (the 128 ASCII characters).
  - Code points in the range U+0080-U+07FF are represented by using 2 bytes.
  - All other code points in the BMP are represented by using 3 bytes.
  - Code points outside of the BMP are represented by using 4 bytes.
- The first byte of a byte sequence representing a code points determines the number of bytes in the byte sequence.

UTF-8 (2)

- The most compact encoding in terms of the number of bytes used.
- It is less efficient when used for East Asian writing systems, such as Chinese, Japanese, and Korean.

Byte Order (1)

- For the UTF-16 and UTF-32 encoding forms the byte order (big-endian or little-endian) must also be also specified.
- Taking account of the byte order, Unicode defines the following seven encoding schemes:
  - UTF-8
  - UTF-16, UTF-16BE, UTF-16LE
  - UTF-32, UTF-32BE, UTF-32LE
- For the UTF-16 and UTF-32 encoding schemes the byte order is determined by the BOM at the very beginning of text.

Byte Order (2)

Byte order mark (**BOM**):

- The character U+FEFF (ZERO WIDTH NO-BREAK SPACE) is used as the byte order mark to indicate the byte order at the very beginning of text.
- It is not part of the textual content and should be removed before processing.

Encoding Scheme	Byte Sequence
UTF-16 big-endian	FE FF
UTF-16 little-endian	FF FE
UTF-32 big-endian	00 00 FE FF
UTF-32 little-endian	FF FE 00 00

ISO/IEC 8859

- 8-bit character encoding standards (ISO/IEC 8859-1, ⋯, ISO/IEC 8859-16).
  - See: ISO/IEC 8859—8-bit single-byte coded graphic character sets
- Relevant encodings for us in Hungary:
  - ISO/IEC 8859-1 (Latin-1): for Western European languages
  - ISO/IEC 8859-2 (Latin-2): for Central European languages
    - Suitable to be used for the following languages: Albanian, Bosnian, Czech, Croatian, Polish, Hungarian, German, Romanian, Serbian (Latin alphabet), Slovakian, Slovenian, Sorbian.

Unicode and Programming Languages

- Modern programming languages are generally based on Unicode, i.e., source programs are sequences of Unicode characters.
- Programming languages that are based on Unicode: C#, ECMAScript, Java, Kotlin, Python, Swift, ⋯

CSS

- Unicode characters can be specified with escape sequences of the form *hhhhhh*, where *hhhhhh* is a sequence of one to six hexadecimal digits representing the code point of the Unicode character.
  - If the number of hex digits is less than six and a character in the range `[0-9a-fA-F]` follows the hexadecimal number, then a whitespace character must end the escape sequence.
    - A whitespace character that immediately follows an escape sequence will be ignored.
  - Examples:
    - `\A9`, `\0A9`, ⋯, `\0000A9`
    - `\262F`, `\0262F`, `\00262F`
- See: https://www.w3.org/TR/css-syntax-3/#escaping

ECMAScript

- In string literals, regular expression literals, template literals and identifiers, any Unicode character may also be expressed using Unicode escape sequences of the form:
  - `\u`*hhhh* where *hhhh* is a sequence of four hexadecimal digits representing the code point.
    - Examples: `\u00A9`, `\u262F`
  - `\u{`*hhhhhh*`}` where *hhhhhh* is a sequence of one to six hexadecimal digits representing the code point.
    - Examples: `\u{A9}`, `\u{1F63A}`
- See: https://262.ecma-international.org/14.0/#sec-source-text

JSON

- In strings, Unicode characters in the BMP may also be expressed using escape sequences of the form `\u`*hhhh*, where *hhhh* is a sequence of four hexadecimal digits representing the code point.
  - Examples: `\u00A9`, `\u262F`
- See: https://www.rfc-editor.org/rfc/rfc8259#section-7

XML, XHTML

- In text, attribute values, and literal entity values Unicode characters may also be expressed using character references of the form:
  - `&#`*nnnn*`;` where *nnnn* is a sequence of decimal digits representing the code point.
    - Examples: `&#169;`, `&#9775;`, `&#128570;`
  - `&#x`*hhhh*`;` where *hhhh* is a sequence of hexadecimal digits representing the code point.
    - Examples: `&#xA9;`, `&#x262F;`, `&#x1F63A;`
- See: https://www.w3.org/TR/xml/#dt-charref

HMTL

- A number of Unicode characters may be expressed using named character references of the form `&`*name*`;`.
  - Examples:
    - `&Eacute;` (U+00C9 = É)
    - `&eacute;` (U+00E9 = é)
    - `&star;` (U+2606 = ☆)
- The list of supported character reference names: https://html.spec.whatwg.org/#named-character-references

Unicode Input

- Linux: In GTK+ applications Unicode characters can be entered by typing ctrl + ⇧ + U, followed by a hexadecimal Unicode code point.
- See: *Unicode input* https://en.wikipedia.org/wiki/Unicode_input

Character Encoding Detection

- On Unix-like systems, the `file` command can be used to determine the character encoding of text files.
- Example of use:

```
file --mime-encoding file.txt
file --mime-encoding *.txt
```

- See: https://man7.org/linux/man-pages/man1/file.1.html

Conversion Tools (1)

iconv:

- Website: https://www.gnu.org/software/libiconv/
- Repository: https://savannah.gnu.org/projects/libiconv/
- Written in: C
- License: LGPLv2.1
- Example of use:

```
iconv --list
iconv -f UTF-8 -t LATIN2 input.txt -o output.txt
```

Conversion Tools (2)

Recode:

- Repository: https://github.com/rrthomas/recode/
- Written in: C
- License: GPLv3
- Example of use:

```
recode --list
recode UTF-8..ISO-8859-2 file.txt
recode UTF-8..UTF-16 *.txt
```

Online Tools

- *Shapecatcher: Draw the Unicode character you want!* https://shapecatcher.com/
- *Unicode Character Search* https://www.fileformat.info/info/unicode/char/search.htm
- *Unicode Party: The Unicode Emoji Search Engine* https://unicode.party/
- &what; https://www.amp-what.com/

Recommended Reading

- Victor Stinner. *Programming with Unicode*.
  https://unicodebook.readthedocs.io/
  https://github.com/vstinner/unicode_book/