Unicode & UTF

by Markus Kuhn

Unicode is well on the way to replace ASCII and Latin-1 in a few years at all levels. It allows you not only to handle text in practically any script and language used on this planet, it also provides you with a comprehensive set of mathematical and technical symbols that will simplify scientific information exchange.

The UTF-8 encoding allows Unicode to be used in a convenient and backwards compatible way in environments that, like Unix, were designed entirely around ASCII. UTF-8 is the way in which Unicode is used under Unix, Linux, and similar systems. It is now time to make sure that you are well familiar with it and that your software supports UTF-8 smoothly.

Contents

What are UCS and ISO 10646?

The international standard **ISO 10646** defines the **Universal Character Set (UCS)**. UCS is a superset of all other character set standards. It guarantees round-trip compatibility to other character sets. If you convert any text string to UCS and then back to the original encoding, then no information will be lost.

UCS contains the characters required to represent practically all known languages. This includes not only the Latin, Greek, Cyrillic, Hebrew, Arabic, Armenian, and Georgian scripts, but also also Chinese, Japanese and Korean Han ideographs as well as scripts such as Hiragana, Katakana, Hangul, Devangari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Khmer, Bopomofo, Tibetian, Runic, Ethiopic, Canadian Syllabics, Cherokee, Mongolian, Ogham, Myanmar, Sinhala, Thaana, Yi, and others. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This includes not only Cuneiform, Hieroglyphs and various Indo-European languages, but even some selected artistic scripts such as Tolkien's Tengwar and Cirth. UCS also covers a large number of graphical, typographical, mathematical and scientific symbols, including those provided by TeX, Postscript, APL, MS-DOS, MS-Windows, Macintosh, OCR fonts, as well as many word processing and publishing systems, and more are being added.

ISO 10646 defines formally a 31-bit character set. However, of this huge code space, so far characters have been assigned only to the first 65534 positions (0x0000 to 0xFFFD). This 16-bit subset of UCS is called the **Basic Multilingual Plane (BMP)** or Plane 0. The characters that are expected to be encoded outside the 16-bit BMP belong all to rather exotic scripts (e.g., Hieroglyphs) that are only used by specialists for historic and scientific purposes. Current plans suggest that there will never be characters assigned outside the 21-bit code

space from 0x000000 to 0x10FFFF, which covers a bit over one million potential future characters. The ISO 10646-1 standard was first published in 1993 and defines the architecture of the character set and the content of the BMP. A second part ISO 10646-2 which defines characters encoded outside the BMP is under preparation, but it might take a few years until it is finished. New characters are still being added to the BMP on a continuous basis, but the existing characters will not be changed any more and are stable.

UCS assigns to each character not only a code number but also an official name. A hexadecimal number that represents a UCS or Unicode value is commonly preceded by "U+" as in U+0041 for the character "Latin capital letter A". The UCS characters U+0000 to U+007F are identical to those in US-ASCII (ISO 646 IRV) and the range U+0000 to U+00FF is identical to ISO 8859-1 (Latin-1). The range U+E000 to U+F8FF and also larger ranges outside the BMP are reserved for private use.

The full name of the UCS standard is

> International Standard ISO/IEC 10646-1, Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. Second edition, International Organization for Standardization, Geneva, 2000-09-15.

It can be ordered online from ISO as a set of PDF files on CD-ROM for 80 CHF (~53 EUR, ~45 USD, ~32 GBP).

What are combining characters?

Some code points in UCS have been assigned to **combining characters**. These are similar to the non-spacing accent keys on a typewriter. A combining character is not a full character by itself. It is an accent or other diacritical mark that is added to the previous character. This way, it is possible to place any accent on any character. The most important accented characters, like those used in the orthographies of common languages, have codes of their own in UCS to ensure backwards compatibility with older character sets. Accented characters that have their own code position, but could also be represented as a pair of another character followed by a combining character, are known as **precomposed characters**. Precomposed characters are available in UCS for backwards compatibility with older encodings such as ISO 8859 that had no combining characters. The combining character mechanism allows to add accents and other diacritical marks to any character, which is especially important for scientific notations such as mathematical formulae and the International Phonetic Alphabet, where any possible combination of a base character and one or several diacritical marks could be needed.

Combining characters follow the character which they modify. For example, the German umlaut character Ä ("Latin capital letter A with diaeresis") can either be represented by the precomposed UCS code U+00C4, or alternatively by the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": U+0041 U+0308. Several combining characters can be applied when it is necessary to stack multiple accents or add combining marks both above and below the base character. For example with the Thai script, up to two combining characters are needed on a single base character.

What is Unicode?

Historically, there have been two independent attempts to create a single unified character set. One was the ISO 10646 project of the International Organization for Standardization (ISO), the other was the Unicode Project organized by a consortium of (initially mostly US) manufacturers of multi-lingual software. Fortunately, the participants of both projects realized in around 1991 that two different unified character sets is not what the world needs. They joined their efforts and worked together on creating a single code table.

Both projects still exist and publish their respective standards independently, however the Unicode Consortium and ISO/IEC JTC1/SC2 have agreed to keep the code tables of the Unicode and ISO 10646 standards compatible and they closely coordinate any further extensions. Unicode 1.1 corresponded to ISO 10646-1:1993 and Unicode 3.0 corresponds to ISO 10646-1:2000. All Unicode versions since 2.0 are compatible, only new characters will be added, no existing characters will be removed or renamed in the future.

The Unicode Standard can be ordered like any normal book, for instance via amazon.com for around 50 USD:

> The Unicode Consortium: The Unicode Standard, Version 3.0, Reading, MA, Addison-Wesley Developers Press, 2000, ISBN 0-201-61633-5.

If you work frequently with text processing and character sets, you definitely should get a copy. It is also available online now.

So what is the difference between Unicode and ISO 10646?

The Unicode Standard published by the Unicode Consortium contains exactly the ISO 10646-1 Basic Multilingual Plane at implementation level 3. All characters are at the same positions and have the same names in both standards.

The Unicode Standard defines in addition much more semantics associated with some of the characters and is in general a better reference for implementors of high-quality typographic publishing systems. Unicode specifies algorithms for rendering presentation forms of some scripts (say Arabic), handling of bi-directional texts that mix for instance Latin and Hebrew, algorithms for sorting and string comparison, and much more.

The ISO 10646 standard on the other hand is not much more than a simple character set table, comparable to the well-known ISO 8859 standard. It specifies some terminology related to the standard, defines some encoding alternatives, and it contains specifications of how to use UCS in connection with other established ISO standards such as ISO 6429 and ISO 2022. There are other closely related ISO standards, for instance ISO 14651 on sorting UCS strings. A nice feature of the ISO 10646-1 standard is that it provides CJK example glyphs in five different style variants, while the Unicode standard shows the CJK ideographs only in a Chinese variant.

What is UTF-8?

UCS and Unicode are first of all just code tables that assign integer numbers to characters. There exist several alternatives for how a sequence of such characters or their respective integer values can be represented as a sequence of bytes. The two most obvious encodings store Unicode text as sequences of either 2 or 4 bytes sequences. The official terms for these encodings are UCS-2 and UCS-4 respectively. Unless otherwise specified, the most significant byte comes first in these (Bigendian convention). An ASCII or Latin-1 file can be transformed into a UCS-2 file by simply inserting a 0x00 byte in front of every ASCII byte. If we want to have a UCS-4 file, we have to insert three 0x00 bytes instead before every ASCII byte.

Using UCS-2 (or UCS-4) under Unix would lead to very severe problems. Strings with these encodings can contain as parts of many wide characters bytes like '\0' or '/' which have a special meaning in filenames and other C library function parameters. In addition, the majority of UNIX tools expects ASCII files and can't

read 16-bit words as characters without major modifications. For these reasons, **UCS-2** is not a suitable external encoding of **Unicode** in filenames, text files, environment variables, etc.

The **UTF-8** encoding defined in ISO 10646-1:2000 Annex D and also described in RFC 2279 as well as section 3.8 of the Unicode 3.0 standard does not have these problems. It is clearly the way to go for using **Unicode** under Unix-style operating systems.

UTF-8 has the following properties:

- UCS characters U+0000 to U+007F (ASCII) are encoded simply as bytes 0x00 to 0x7F (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF-8.
- All UCS characters >U+007F are encoded as a sequence of several bytes, each of which has the most significant bit set. Therefore, no ASCII byte (0x00-0x7F) can appear as part of any other character.
- The first byte of a multibyte sequence that represents a non-ASCII character is always in the range 0xC0 to 0xFD and it indicates how many bytes follow for this character. All further bytes in a multibyte sequence are in the range 0x80 to 0xBF. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.
- All possible 2^{31} UCS codes can be encoded.
- UTF-8 encoded characters may theoretically be up to six bytes long, however 16-bit BMP characters are only up to three bytes long.
- The sorting order of Bigendian UCS-4 byte strings is preserved.
- The bytes 0xFE and 0xFF are never used in the UTF-8 encoding.

The following byte sequences are used to represent a character. The sequence to be used depends on the Unicode number of the character:

U-00000000 - U-0000007F:	0*xxxxxxx*
U-00000080 - U-000007FF:	110*xxxxx* 10*xxxxxx*
U-00000800 - U-0000FFFF:	1110*xxxx* 10*xxxxxx* 10*xxxxxx*
U-00010000 - U-001FFFFF:	11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-00200000 - U-03FFFFFF:	111110*xx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-04000000 - U-7FFFFFFF:	1111110*x* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

The *xxx* bit positions are filled with the bits of the character code number in binary representation. The rightmost *x* bit is the least-significant bit. Only the shortest possible multibyte sequence which can represent the code number of the character can be used. Note that in multibyte sequences, the number of leading 1 bits in the first byte is identical to the number of bytes in the entire sequence.

Examples: The Unicode character U+00A9 = 1010 1001 (copyright sign) is encoded in UTF-8 as

11000010 10101001 = 0xC2 0xA9

and character U+2260 = 0010 0010 0110 0000 (not equal to) is encoded as:

11100010 10001001 10100000 = 0xE2 0x89 0xA0

The official name and spelling of this encoding is UTF-8, where UTF stands for **U**CS **T**ransformation **F**ormat. Please do not write UTF-8 in any documentation text in other ways (such as utf8 or UTF_8), unless of course you refer to a variable name and not the encoding itself.

An important note for developers of UTF-8 decoding routines: For security reasons, a UTF-8 decoder must not accept UTF-8 sequences that are longer than necessary to encode a character. For example, the character U+000A (line feed) must be accepted from a UTF-8 stream **only** in the form 0x0A, but not in any of the following five possible overlong forms:

0xC0 0x8A
0xE0 0x80 0x8A
0xF0 0x80 0x80 0x8A
0xF8 0x80 0x80 0x80 0x8A
0xFC 0x80 0x80 0x80 0x80 0x8A

Any overlong UTF-8 sequence could be abused to bypass UTF-8 substring tests that look only for the shortest possible encoding. All overlong UTF-8 sequences start with one of the following byte patterns:

1100000*x* (10*xxxxxx*)
11100000 100*xxxxx* (10*xxxxxx*)
11110000 1000*xxxx* (10*xxxxxx* 10*xxxxxx*)
11111000 10000*xxx* (10*xxxxxx* 10*xxxxxx* 10*xxxxxx*)
11111100 100000*xx* (10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*)

Also note that the code positions U+D800 to U+DFFF (UTF-16 surrogates) as well as U+FFFE and U+FFFF must not occur in normal UTF-8 or UCS-4 data. UTF-8 decoders should treat them like malformed or overlong sequences for safety reasons.