ASCII, Unicode, and In Between

Douglas A. Kerr

Issue 3
June 15, 2010

ABSTRACT

The standard coded character set ASCII was formally standardized in 1963 and, in its "complete" form, in 1967. It is a 7-bit character set, including 95 graphic characters. As personal computers emerged, they typically had an 8-bit architecture. To exploit that, they typically came to use character sets that were "8-bit extensions of ASCII", providing numerous additional graphic characters. One family of these became common in computers using the MS-DOS operating system. Another similar but distinct family later became common in computers operating with the Windows operating system.

In the late 1980s, a true second-generation coded character set, called Unicode, was developed, and was standardized in 1991. Its structure provides an ultimate capacity of about a million graphic characters, catering thoroughly to the needs of hundreds of languages and many specialized and parochial uses. Various encoding techniques are used to represent characters from this set in 8- and 16-bit computer environments.

In this article, we will describe and discuss these coded character sets, their development, their status, the relationships between them, and practical implications of their use.

Information is included on the entry, in Windows computer systems, of characters not directly accessible from the keyboard.

BEFORE ASCII

As of 1960, the interchange of data within and between information communication and processing systems was made difficult by the lack of a uniform coded character set.

Information communication in the traditional "teletypewriter" sense generally used a 5-bit coded character set, best described as the "Murray code" (although as a result of historical misunderstandings it is often called the "Baudot code"). It was in international use, but there were variations in detail between different applications (even in the U.S.).

Specialized data communication systems (typically thought of as remote data entry systems for information processing) often used one of several 6-bit coded character sets.

“Mainframe” computers also typically used, internally, a 6-bit coded character set. Many IBM systems used a 6-bit code called the *BCD Interchange Code* (BCDIC), which evolved from a binary representation of the “Hollerith” punched card code.

This situation threatened the efficient development of the “new age of information” which was seen as coming over the horizon at the time. It also stood in the way of an aspiration of the United States federal government: to be able to separately procure different parts of information processing systems, such as central proceeding units, data storage devices, printers, and the like.

With different manufacturers using different “native” coded character sets, it would have been impractical, for example, for the Air Force to purchase central processing units from IBM but printers from Univac. And of course there were grave difficulties at the interfaces between information telecommunication systems and computers.

To seek relief from these limitations, the information processing and communication industry, encouraged by the federal government, undertook the development of a standard coded character set for information interchange. The result was the emergence of the coded character set we know as ASCII.

ASCII

Initial development

The work that led to ASCII was conducted by a technical committee structure organized by BEMA, the Business Equipment Manufacturers’ Association, under the auspices of the American Standards Association (ASA).

One initial debate was over the “word size” of the new character set. One camp felt that 6 bits was appropriate. Another camp, foreseeing the emergence of 8-bit native architecture for computer systems, urged adoption of an 8-bit code. The conclusion: 7 bits.

On June 17, 1963, the American Standards Association approved a standard entitled (in accordance with the norms of the organization) “American Standard Code for Information Interchange”. This was a 7-bit coded character set, whose 128 “code points” were assigned as follows:

- 64 graphic characters, comprising the digits 0-9, the letters A-Z (in only a single case, considered to be upper-case), the non-printing graphic character Space, and 27 punctuation marks.

- 33 control characters, which included characters used for graphic formatting, for the control of information transmission, and (in one case) for the practice of a traditional technique for "erasing" an erroneously-entered character in perforated paper tape.

- 63 unassigned code points (reserved for future definition; many presumed 26 of them to be destined to become homes for explicit lower-case letters).

The main title of the standards document, "American Standard Code for Information Interchange", was also taken as the name of the code itself. The acronym for that name, ASCII, soon became the "short name" for the code.

Finishing the job

In 1965, ASA approved a revised version of the standard, in which the lower-case letters and some additional punctuation marks were included in the previously reserved part of the code space. Some other technical improvements were made as well.

This version, however, was never published, owing to important developments on the international front. An international equivalent to ASCII was emerging (we will read of it shortly), and the ASCII team wanted to hold up publication of the "new, enlarged, and improved" ASCII until the American and international work could be harmonized.

> In 1966, the name of the American Standards Association was changed to the USA Standards Institute (USASI). This led to a change in basic structure of the titles of the standards it subsequently published.

In 1967, harmonization having reached a satisfying state, USASI approved and published a standard for a "complete" version of the coded character set (all 128 code points assigned), under the title "USA Standard Code for Information Interchange". This new edition of the code included the 26 lower-case letters and 5 additional punctuation marks, and made some other changes, partly due to the harmonization with the international standard.

An official short name

There was immediately a trend in some quarters to refer to this new code with the acronym of the new title of its specification, "USASCII" (sometimes presented as "US-ASCII"), some workers using this to distinguish this version of the code from "ASCII", which to them would mean the original (lower-case only) version.

At a certain point in the succession of standards, the document prescribed the use of either short designation, “ASCII” or “USASCII”, for any version of the code. But the formal recognition of “USASCI” as a short name was later rescinded, leaving the official short name of the code (even as it would continue to evolve) as “ASCII”, which prevails to this day.

Federal status

In March 1968, U.S. President Lyndon B. Johnson, by executive order, proclaimed that “All computers and related equipment configurations brought into the Federal Government inventory on and after July 1, 1969, must have the capability to use the Standard Code for Information Interchange”. The ASCII standard (essentially the 1967 form) became Federal Information Processing Standard 1, the first interagency information processing standard.

Further refinement

In 1968, a new issue of the standard was promulgated making some very small changes (including an alternate interpretation of the “end of line” formatting characters).

> In 1969, the name of the U.S. standards organization was changed to the American National Standards Institute (ANSI), leading to another change in the basic structure of the titles of standards. Fortunately, by that time the short name ASCII was immortalized.

The text of the ASCII standard was updated slightly in 1977, and slightly again in 1986 (and that version reaffirmed, unchanged, in 1997 and 2002), but the coded character set itself has remained unchanged since the 1967 issue.

The current version

The “main title” of the current edition of the ASCII standard (1986) is now “7-Bit American National Standard Code for Information Interchange”, but in the document, the code itself is referred to as “American National Standard Code for Information Interchange”, prescribed to be identified as “ASCII”.

Bit numbering

Although it really has little bearing on our topic, it is interesting to look at the way the seven bits of an ASCII code value are identified in the ASCII standards.

Under today’s understanding of information science (and basic number system theory), the least-significant bit (on the right, the way the binary number is conventionally written) would be indentified as "B0", with the most-significant bit as "B6".

But many of the workers involved in ASCII development were not steeped in information science (which was itself still being codified at the time). Many (such as myself) were from such fields as teletypewriter engineering. To some of them, the notion of a bit named "bit zero" was thought to be hard to grasp (although there was in fact a bit 0 in one of the traditional teletypewriter codes[1]).

Accordingly, it was decided to number the least-significant bit B1 and the most-significant bit B7.

Manifestations

The code described by the ASCII standards is abstract: each character is associated with what we today call a *code point*, a value of a 7-bit number. The standard does not define how this number is to be stored in logical or physical form, how to handle it in 8-bit or 16-bit contexts, or how it is to be represented in an electrical bit-parallel or bit-serial interface. These are matters for other related industry standards.

IANA designation

The IANA[2] designation for this coded character set (used to identify the character set encoding in use in an Internet context) is "US-ASCII". It is a curious name, given that there is no "other than US" form of ASCII. It perhaps came from the once-legitimate status of "USASCII" as an alternate designation of the code.

ISO 646—ASCII's international cousin

ISO 646[3] (first formalized in 1967) is an internationally-standardized 7-bit coded character set almost identical to ASCII, except that 18 code points are reserved for local assignment of characters in distinct national variants of the code (formally recognized). ASCII itself thus becomes a particular national variant of ISO 646 (called ISO 646-US

[1] The Teletypesetter code, used to transmit "ready to print" text, was in effect a 6-bit superset of the Murray code. The "added" bit was designated "bit 0" not for any mathematical reason but because, for reasons of physical symmetry, it was placed before "bit 1" on the perforated paper tape used to store messages.

[2] IANA refers to the Internet Assigned Numbers Authority, operated by the Internet Corporation for Assigned Names and Numbers (ICANN).

[3] ISO refers to the International Organization for Standardization. No, ISO is not the initials of its name in any of the official languages. Rather, it is an anagram of its English initials to make a Greek pun (*iso* = same).

or ISO 646-006, depending on which of two different systems of notation is involved)

THE PERSONAL COMPUTER

As the personal computer emerged[4], the storage of data as 8-bit words[5] became normative.

The developers did not want to "waste" an 8-bit storage "cell" on a 7-bit ASCII character (with a "pad" bit), so they constructed "extended ASCII" character sets that had 256 code points, the first 128 of which were identical to the characters of ASCII. The added 128 code points (or at least some of them) were used for added graphic characters. Typically these included characters used in languages beyond English, as well as "drawing" characters that could be used, on a character-oriented printer or display, to draw various shapes for block diagrams or organization charts, make outlines for cells in tables, and the like.

Different character sets of this type were used by different manufacturers and on different computer models.

When the MS-DOS operating system came into use, in the U.S. it typically used a particular character set of this type, identified by Microsoft as Code Page 437 (CP-437). This in fact incorporated the repertoire of character glyphs embedded in the ROM of the IBM Enhanced Graphics Adapter (EGA). (Bill Gates said he took that repertoire from the character set of the Wang dedicated word processing machines, whose function he hoped would soon be taken over by software running on personal computers.)

Although the first 32 code positions of CP-437 correspond to 32 control characters of ASCII, at the display interface (where only graphic characters are meaningful), an alternate form is used in which those code points are assigned to 32 graphic characters (including a fixed-width space for use with numeric characters).

There are other code pages in this series, mainly catering to various non-English languages (CP-866, for example, carrying the Cyrillic alphabet).

[4] I use the term in its basic sense, not in the sense of "was it an (IBM) Personal Computer or a Mackintosh?"

[5] I use *word* here in its generic sense in information science, not of itself implying any bit size.

ENTER WINDOWS

When the Windows operating environment (later recognized as an actual operating system) emerged, again the storage of character data in 8-bit words was assumed, and again an “extended ASCII” character set was adopted to best exploit the available code space.

As in the MS-DOS context, Microsoft established a family of such Windows coded character sets, each intended to be optimal in different application regions (based on the language(s) to be best supported). These code tables were also called Code Pages (an extension of the series established in MS-DOS).

The one normally provided with Windows-based computers in the United States is identified as (Windows) Code Page 1252 (CP-1252). It includes all the characters of ASCII (with their normal code point values, albeit expressed in 8-bit form, with a leading zero) plus 123 additional graphic characters (five of the 128 added code point values are “blank”). These characters accommodate many non-English languages, as well as other documentation needs.

This coded character set never became the subject of any international standard. A variation of it, however, is standardized as ISO 8859-1 (see the next section).

There are other Windows Code Pages catering to the needs of non-Latin alphabet languages (CP-1251, for example, carrying the Cyrillic alphabet).

IANA designation

The IANA designation for this coded character set (used to identify the character set encoding in use in an Internet context) is “Windows-1252”.

No, it’s not ASCII

The use of CP-1252 has become so widespread in U.S. computer operation that some people think it is ASCII; speaking of data in text form, encoded in CP-1252, they may say, “this item is in ASCII”.

There are many data format standards that prescribe text to be encoded in ASCII (and mean exactly that). Nevertheless, these specifications are often embraced on a liberal basis, with CP-1252 encoding being used in applications where an expanded character set is desired. (Yes, of course this can cause serious problems!)

ISO/IEC 8859-1

Although Code Page 1252 never became the subject of any formal international standard, a coded character set closely related to it is the subject of international standard ISO/IEC 8859-1[6]. In that standard, code points 0x0 through 0X1F and 0x7F through 0x9F are unassigned. The remainder basically follows CP-1252. Ther are other code pages in the series (with different suffix numbers). They have character sets intended for the support of languages other than those using the Latin alphabet.

IANA ISO-8859-1

IANA does not recognize the ISO/IEC 8859-1 character set for Internet use, and accordingly there is no IANA identifier for it.

But IANA did concoct an "augmented" form of the ISO/IEC 8859-1 character set for Internet use. Its IANA identifier is ISO-8859-1![7]

It differs from the ISO/IEC 8859-1 character set as follows:

- Code points 0x0 through 0X1F and 0x7F are assigned to the corresponding ASCII control characters (as in CP-1252), known in this context as the "C0" control characters.
- Code points 0x80 through 0x9F are assigned to a new set of specialized control characters, known in this context as the "C1" control characters. (Their story is too curious for even this article.)

"ASCII" VS. "ANSI" CHARACTER SETS

In connection with Windows systems we often find a distinction drawn between "ASCII" and "ANSI" character sets. What does that mean? Well, in that context, as used by Microsoft:

- "ASCII" means "ASCII"
- "ANSI" means Windows Code Page 1252 (CP-1252).

How odd. Does this mean that:

- CP-1252 is an ANSI standard.
- ASCII is not.

[6] IEC refers to the International Electrotechnical Commission.

[7] Hey, I don't (anymore) do these things—I just report 'em.

No, just the opposite:

- ASCII **is an ANSI standard** (one of several for character sets, in fact), and has been as long as there has been an ANSI (since 1969).
- CP-1252 **is not an ANSI standard**, and never has been. (But at one time Microsoft had hoped that it would become one.)

So, go figure. The Mind of Microsoft!

We will find a related curiosity of notation in the next section.

KEYBOARD ENTRY OF CHARACTERS BEYOND ASCII

In Windows systems, there are two schemes of entering characters not found on the keyboard (i.e., which are not in ASCII proper) with the use of the Alt key and the numeric cluster. One is called "entry of ASCII codes" and one is called "entry of ANSI codes". Both terms are misnomers.

In a Windows system, to enter the character whose code in CP-1252[8] is 178 (decimal), for example, one would hold the Alt key and enter 0178 on the numeric cluster (Alt+0178 for short). The leading zero is a clue to the O/S that the numeric value entered is in fact the code for the wanted character in the native character set for Windows. (Why might it not be? Read on.) This is spoken of (inexplicably) as the "ANSI code entry" scheme, and those numbers (with the leading zero) are (inexplicably) spoken of as the "ANSI codes" for the characters.

If we instead keyed Alt+178 (omitting the leading zero), we would not get the character whose code in the CP-1252 character set is 178 (decimal), but rather the character in CP-1252 whose glyph was assigned code 178 (decimal) in CP-437[9], the "extended ASCII" character set used in MS-DOS (but not in Windows).

This was done to allow people used to a certain keyboard entry process for a certain extended character in MS-DOS to use the same entry in Windows, to the extent that the characters were available in both sets.

This scheme is spoken of (inexplicably) as "ASCII code entry", and those numbers are (inexplicably) spoken of as the "ASCII codes" for the characters.

[8] Or an alternate Windows Code Page that may be in effect.

[9] Or the MS-DOS Code Page corresponding to the Windows Code Page in effect.

If we wish to enter characters with codes from 32 through 127 (characters included in ASCII proper) in this mode, we can use either form. In fact, for the character "N", whose code, in ASCII, and in CP-437 (MS-DOS), and in CP-1252 (Windows), is 78 (decimal), we can enter Alt+78, Alt+078, Alt+0078, or (if we are trying to prove a point), Alt+000078.

If we use this procedure with a numeric value less that 32 (decimal), with or without a leading zero, we will get the associated character from CP-437 (in its “display” variant, which in that region of the code space has graphic characters rather than control characters).

UNICODE

Introduction

As information technology came to deal with an increasingly-global context, it because clear that ASCII and its cousins could not truly support the gigantic range of graphic characters that were encountered. As a consequence, a new coded character set structure was devised, called Unicode. It was first formally standardized in 1991. The Unicode standard is administered by The Unicode Consortium.

It can be thought of as a superset of the U.S. form of the international 7-bit coded character set, ISO 646 (and in fact the ISO standard for the equivalent of Unicode, the Universal Character Set (UCS), is ISO 10646—isn't that cute). But that characterization as a superset of ISO 646 hardly does justice to the enormous richness of Unicode.

As with the other coded character sets we have discussed, the entries of the "Unicode code chart" are *code points*, each one having a *code point value*. These values are just numbers, and do not of themselves imply any representation in bits, words, or bytes.

Many of the code points are assigned to represent characters; for graphic characters, a *glyph* (graphic symbol) is implied. There are large areas of the code space, however, that are either reserved for future assignment, or are reserved for "private" use.

In what follows, I will use the "0x-" notation (as used in several programming languages) to present numbers in hexadecimal form (as for example, “0xFEA2”), both for abstract numbers (used to identify Unicode code points) and for the hexadecimal numerical equivalent of the bit coding of individual bytes (“0x0D”), those always expressed with two hexadecimal digits.

The range of the Unicode “code space” is from 0x0 to 0x10FFFF (in decimal, 0 to 1114111). Thus, to directly represent the entire code space, we would need to use a 21-bit number.

The code space is considered to be divided into 17 code planes (16-bit code subspaces), each potentially having 65536 code points. "Code plane 0", which embraces code point values from 0x0 through 0xFFFF, is called the "Basic Multilingual plane". It covers the needed characters to write in a large range of languages.

Unicode code points are written in this form (to give an example):

U+01F7

where the number after the "+" is understood to be in hexadecimal (no radix indicator needed). For code point values not over 0xFFFF (that is, within the Basic Multilingual Plane), we write this number built out to four hexadecimal characters, for example (for 0xD):

U+000D

The basement

The first 256 code points (U+0000 through U+00FF) correspond to ISO 8859-1 (much of which corresponds to Windows code page 1252). Of that, the first 128 code points (U+0000 through U+007F) in fact correspond to ASCII. This repertoire includes a modest repertoire of accented characters for use in various non-English "Western" (Latin-alphabet) languages.

Encoding

Code point values are just abstract numbers. When we actually want to store Unicode characters in computer memory or a data file, we need to employ some form of specific encoding suitable to the context.

If for some reason we had a computer system that used 21-bit words for character storage, then we could use the obvious direct encoding. But we rarely (if ever) encounter such a context.

Thus, we must usually use some more crafty encoding.

The two encodings of most interest to us are forms of the *Unicode Transformation Format*, called UTF-16 and UTF-8.

<u>UTF-16</u>

The UTF-16 encoding is intended for the representation of Unicode characters in an environment or context emphasizing 16-bit words.

For characters in the Basic Multilingual Plane, with code point values in the range 0x0 to 0xFFFF, the UTF-16 encoding is just the code point value itself, as a 16-bit word.

However, at the next lower, more physical, level, we come to grips with two conventions found in information systems with a 16-bit semantic architecture but an 8-bit (byte) physical structure. These are the *big-endian* and *little-endian* conventions.

- In the big-endian convention, the highest-order 8 bits of a 16-bit word are carried in one byte, and the lowest-order 8 bits of the word in the following byte.
- In the little-endian convention, the lowest-order 8 bits of a 16-bit word are carried in one byte, and the highest-order 8 bits of the word in the following byte.

Thus, at the byte level, the byte sequence for the UTF-16 encoding of a particular Unicode character will vary.

For characters beyond those in the Basic Multilingual Plane, the UTF-16 encoding involves two 16-bit words (32 bits). The arrangement is made possible by the fact that certain code point values in the range of the Basic Multilingual Plane are not assigned to characters, but rather to a special “surrogate” role in connection with this "32-bit" encoding. Again, here, the *big-endian/little-endian* matter pertains.

The details of this “32-bit” encoding are covered in Appendix A.

UTF-8

The UTF-8 encoding is intended for the representation of Unicode characters in an environment or context emphasizing 8-bit words (octets or bytes). It uses a clever variable-length scheme.

Code points with values through 0x007F (the ASCII character set) are coded in single-byte form; their code point value is represented in 8 bits (that is, with a leading zero). The leading zero, in fact, is a cue that this byte by itself represents a character.

Code points with values through 0x0080 through 0x07FF are coded in two-byte form. This includes all the characters needed to write the scripts of most of the non-Asian languages.

Code points in the remainder of the Basic Multilingual Plane (through 0xFFFF) require three bytes.

Code points in other planes require four bytes.

The details of this scheme are covered in Appendix B.

For ASCII characters, the UTF-8 encoding is identical to a straight ASCII representation in a byte. That is not true, however, for characters in the "upper half" of Windows Code Pages, which natively are in one byte (the "extended ASCII" concept). In UTF-8, they require two or three bytes.

There is no issue comparable to big-endian vs. little-endian in connection with the use of UTF-8 encoding.

Keyboard entry of "beyond-ASCII" Unicode characters

In a Windows system supporting a Unicode character set, we can enter "beyond ASCII" Unicode characters based on their Unicode code point values in way similar to that used to enter beyond-ASCII character from CP-437 or CP-1252.

We just hold Alt and (on the numeric cluster) enter the decimal equivalent of the Unicode code point value (with a leading zero if the value is not over 255). Thus, for the Cyrillic character upper case *Ya* (Я), whose Unicode code point is U+042F, we hold Alt and, on the numeric cluster key 1071 (or 01071, if for some reason we want to).

If we key a value such as Alt+0195, how does the system know whether we mean the character whose CP-1252 code is 195 (decimal) or the Unicode character whose code point value is (decimal) 195? We note the following:

- For code values in the range 32-127 decimal, the CP-1252 and Unicode characters are the same (the ASCII character, in fact).
- For code values (decimal) in the range 128-191, there is no graphic character in Unicode.
- For code values in the range 192-255, the CP-1252 and Unicode characters are the same.

Thus:

- For entries in the range Alt+0032 through Alt+0255, the entry is interpreted as meaning the CP-1252 character.
- For entries above Alt-0255 (or Alt+255), the entry is interpreted as meaning the Unicode character.

A RECURRENT PROBLEM

Perhaps the problem today most frequently-encountered in connection with Unicode representation relates to characters in the "upper half" of Windows Code Page 1252 (non-ASCII characters).

For example, the character © (the copyright symbol), in CP-1252, has the code point value 0xA9, and is represented as a single byte with value 0xA9.

In Unicode, this character also conveniently has the code point value 0xA9 (which we might write 0x00A9, or U+00A9). But in an 8-bit context, using Unicode UTF-8 encoding, it is represented by the two byte sequence 0xC2 0xA9.

A receiving application that expects the text to be encoded in CP-1252 form will render this sequence as Â© rather than ©.

For the characters in the "third quarter" of CP-1252, it always works out this way. The two byte encoding is always the byte C2h followed by the CP-1252 byte coding for the character.

But now consider the character € (the Euro currency symbol). It is in the "fourth quarter" of CP-1252 (a "newcomer" there). Its CP-1252 code point value is 0x80, and, in CP-1252 encoding, it is represented by a single byte, 0x80.

But this character in Unicode has code point value 0x20AC. In UTF-8 encoding, this is represented by the byte sequence 0xE2 0x82 0xAC.

A receiving application that expects the text to be encoded in CP-1252 form will render this sequence as Â ‘ ¢ rather than €.

The problem is exacerbated by the fact that, in some cases where a data item is specified to be in ASCII (meaning just that), some practitioners have stretched this to mean "in ASCII or CP-1252". More recently, others have stretched it to mean "in ASCII or UTF-8". [10]

[10] One example of resulting problem is that the image editing program, Photoshop, when placing a user-composed "copyright notice" that includes the beyond-ASCII character '©' into the "Exif metadata" portion of an image file (where the file format specification prescribes ASCII) encodes the character in Unicode UTF-8. But many programs that "read" the Exif metadata assume that "beyond-ASCII characters" will be encoded in CP-1252, and accordingly deliver 'Â©' to the viewer.

A MELLOW MOMENT

I was not involved in the wonderful initial body of work that lead to ASCII. I joined the party just after the issuance of the first version in 1963. But I was heavily involved in its completion, refinement, documentation, and implementation.

At the time, there was some concern as to whether or not ASCII would “catch on”. (And not everyone felt its “advantages” were beneficial to their own commercial agendas). Now, 47 years after its first formal emergence, we see it as one of the key tools that has made possible the gigantic expansion in the accessibility to information we enjoy daily. And I am frankly proud of my small role in its life.

More recently, following that same spirit, but with greatly enlarged needs and aspirations, vastly greater insight, and an enormous arsenal of development and implementation tools, another generation has brought us Unicode (without any help, or interference, from me). I have the same vicarious pride in this that a grandfather has in the accomplishments of his grandchildren.

Carla and I view all human enterprise as semantic in nature, with communication as one of the most fundamental ingredients. We thus see work on tools that enable, and enhance, and facilitate, and encourage, and improve, communication to be among the most potent agents of human accomplishment.

This article is dedicated to all those who helped bring these wonderful tools—ASCII, Unicode, and their relatives and adjuncts—to fruition.

#

APPENDIX A

Details of UTF-16 encoding

Introduction

UTF-16 encoding is intended for the representation of Unicode characters in an environment or context emphasizing 16-bit words.

For characters in the Basic Multilingual Plane, with code point values in the range 0x0 to 0xFFFF, the UTF-16 encoding is just the code point value itself, as a 16-bit word.

For characters beyond those in the Basic Multilingual Plane, the UTF-16 encoding involves two consecutive 16-bit words (32 bits). (We will rarely encounter this.)

In either case, in a context where data is logically or physically stored or transmitted in byte form, there is, at the next lower physical level, the issue of how the 16-bit words are to be represented as two bytes each (the *big-endian* vs. *little-endian* matter). This is discussed in the body of the article.

The use of double 16-bit word encoding for code points with values beyond 0xFFFF is made possible by a system of *surrogate code points*. Of the 2^{21} possible Unicode code point values, two blocks of 1024 contiguous values each are reserved as surrogates. They do not have characters assigned, and as unpaired 16-bit words are not valid.

These two reserved blocks have code point values in these ranges:

0xD800–0xDBFF and 0xDC00-0xDFFF

Each code point with a value greater than 0xFFFF is represented in UTF-16 encoding by two consecutive 16-bit words whose values are drawn from the first and second surrogate blocks, respectively, a sequence called a *surrogate pair*.

Since those 16-bit words are never used alone to directly represent a code point, the words of a surrogate pair can be unambiguously recognized.

Principle

The Unicode code points that are to be encoded as a surrogate pair lie in the range 0x010000 through 0x10FFFF (all code points below 0x10000 are directly encoded as a single 16-bit word). That is a range of 0x100000 (2^{20}). Thus, all values in the range could be represented by a 20-bit number (by offsetting the origin).

To apply the offset, we subtract 0x10000 from the code point value, giving a value with a range from 0x0 through 0x100000 (which can indeed be expressed as a 20-number). This 20-bit value is divided into two 10-bit parts. Each part is carried by the rightmost 10 bits of one of the two surrogate words. The first six bits of each of the two surrogate words is fixed (different for the two), allowing them to be unambiguously recognized as surrogate words, not words directly representing a code point. (No code point assigned to a character has either of those six initial bit patterns—those ranges of code point values are "reserved" for use as surrogates.)

Encoding

We will use S1 and S2 represent the first and second 16-bit words in the surrogate pair. Let C be the Unicode code point value to be encoded:

- Let C′=C - 0x10000
- Consider C′ as a 20-bit number
- Form S1 in binary by prepending 1101 10 to the most-significant 10 bits of C′.
- Form S1 in binary by prepending 1101 11 to the least-significant 10 bits of C′.

Parsing

If the data arrives in byte form, form 16-bit words in accordance with the endianness in effect. Then:

- If a 16-bit word has its first five bits not equal to 11011, it is a single-word character, and its code point value is just the word value.
- If the first six bits equal 111010, it is the first word (S1) of a two-word surrogate pair. The following word must be an S2 word.
- If the first six bits equal 111011, it is the second word (S21) of a two-word surrogate pair. The preceding word must be an S1 word.

Decoding surrogate pairs

The code point value, C, represented by a surrogate pair can be decoded with the following equation (in hexadecimal arithmetic), where S1 and S2 represent values of the first and second 16-bit words:

C = + 0x0400*(S1-0xD800) + (S2-0xDCOO) + 0x1000

APPENDIX B

Details of UTF-8 encoding

Introduction

UTF-8 encoding is intended for the representation of Unicode characters in an environment or context emphasizing 8-bit words (bytes or octets). A Unicode character is represented by a sequence of from one to four bytes, using a clever variable-length encoding syntax. The syntax can be unambiguously parsed without requiring the maintenance of synchronism from the beginning of the file.

- Code points with values through 0x7F (the ASCII character set) are coded in single-byte form; their code point value is represented in 8 bits (that is, with a leading zero). The representation is identically that used for ASCII encoding proper.

- Code points with values through 0x80 through 0x7FF are coded in two-byte form. This includes all the characters needed to write the scripts of most of the non-Asian languages.

- Code points in the remainder of the Basic Multilingual Plane require three or four bytes.

Encoding

In the table that follows, the 21 bits of the Unicode code point value are represented this way (the bits are arbitrarily separated into groups of 4 for clarity):

a bcde fghi jklm nopq rstu

The encoding of the characters in the different code point value ranges is as follows:

Range of code point values (hexadecimal)	1st byte (binary)	2nd byte (binary)	3rd byte (binary)	4th byte (binary)
000000-00007F	0*opq rstu*	Bits not included in the encoding are always 0 in the pertinent range		
000080-0007FF	110*k lmno*	10*pq rstu*		
000800-00FFFF	1110 *fghi*	10*jk lmno*	10*pq rstu*	
010000-10FFFF	1111 0*abc*	10*de fghi*	10*jk lmno*	10*pq rstu*

Parsing

In parsing a byte stream:

- Any byte whose leftmost bit is 0 is a singleton, and is interpreted as if in ASCII.
- Any byte whose two leftmost bits are 11 is the first byte of a sequence; the total length of the sequence (in bytes) is equal to the number of contiguous bits (from the left) that are 1.
- Any byte whose leftmost two bits are 10 is a later byte of a sequence.

Decoding multi-byte sequences

The decoding proceeds as intimated by the encoding table, above. The resulting word, representing the code point value, may be thought of as extended to 21 bits with leading zeros where required.