Network Working Group
Request for Comments: 5198
Obsoletes: 698
Updates: 854
Category: Standards Track

J. Klensin
M. Padlipsky

March 2008

Unicode Format for Network Interchange

Status of This Memo

Abstract

The Internet today is in need of a standardized form for the transmission of internationalized "text" information, paralleling the specifications for the use of ASCII that date from the early days of the ARPANET. This document specifies that format, using UTF-8 with normalization and specific line-ending sequences.

Table of Contents

1. Introduction

1.1. Requirement for a Standardized Text Stream Format

Historically, Internet protocols have been largely ASCII-based and references to "text" in protocols have assumed ASCII text and specifically text in Network Virtual Terminal ("NVT") or "Network ASCII" form (see Appendix A and Appendix B). Protocols and formats that have moved beyond ASCII have included arrangements to specifically identify the character set and often the language being used.

In our more internationalized world, "text" clearly no longer equates unambiguously to "network ASCII". Fortunately, however, we are converging on Unicode [Unicode] [ISO10646] as a single international interchange character coding and no longer need to deal with per-script standards for character sets (e.g., one standard for each of Arabic, Cyrillic, Devanagari, etc., or even standards keyed to languages that are usually considered to share a script, such as French, German, or Swedish). Unfortunately, though, while it is certainly time to define a Unicode-based text type for use as a common text interchange format, "use Unicode" involves even more ambiguity than "use ASCII" did decades ago.

Unicode identifies each character by an integer, called its "code point", in the range 0-0x10ffff. These integers can be encoded into byte sequences for transmission in at least three standard and generally-recognized encoding forms, all of which are completely defined in The Unicode Standard and the documents cited below:

- UTF-8 [RFC3629] defines a variable-length encoding that may be applied uniformly to all code points.

- UTF-16 [RFC2781] encodes the range of Unicode characters whose code points are less than 65536 straightforwardly as 16-bit integers, and provides a "surrogate" mechanism for encoding larger code points in 32 bits.

- UTF-32 (also known as UCS-4) simply encodes each code point as a 32-bit integer.

Older forms and nomenclature, such as the 16-bit UCS-2, are now strongly discouraged.

As with ASCII, any of these forms may be used with different line-ending conventions. That flexibility can be an additional source of confusion with, e.g., index (offset) references into documents based on character counts.

This document proposes to establish "Net-Unicode" as a new standardized text transmission form for the Internet, to serve as an internationalized alternative for NVT ASCII when specified in new -- and, where appropriate, updated -- protocols. UTF-8 [RFC3629] is chosen for the coding because it has good compatibility properties with ASCII and for other reasons discussed in the existing IETF character set policy [RFC2277]. "Net-Unicode" is specified in Section 2; the subsequent sections of the document provide background and explanation.

Whenever there is a choice, Unicode SHOULD be used with the text encoding specified here. This combination is preferred to the double-byte encoding of "extended ASCII" [RFC0698] or the assorted per-language or per-country character coding systems.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Net-Unicode Definition

The Network Unicode format (Net-Unicode) is defined as follows. Parts of this definition are deliberately informal, providing guidance for specific profiles or rules in the protocols that reference this one rather than firm rules that apply globally.

1. Characters MUST be encoded in UTF-8 as defined in [RFC3629].

2. If the protocol has the concept of "lines", line-endings MUST be indicated by the sequence Carriage-Return (CR, U+000D) followed by Line-Feed (LF, U+000A), often known just as CRLF. CR SHOULD NOT appear except when followed by LF. The only other allowed context in which CR is permitted is in the combination CR NUL, which is not recommended (see the note at the end of this section).

3. The control characters in the ASCII range (U+0000 to U+001F and U+007F to U+009F) SHOULD generally be avoided. Space (SP, U+0020), CR, LF, and Form Feed (FF, U+000C) are exceptions to this principle, but use of all but the first requires care as discussed elsewhere in this document. The so-called "C1 Controls" (U+0080 through U+009F), which did not appear in ASCII, MUST NOT appear.

   FF should be used only with caution: it does not have a standard and universal interpretation and, in particular, if its use

assumes a page length, such assumptions may not be appropriate in international contexts (e.g., considering 8.5x11 inch paper versus A4). Other control characters are used to affect display format, control devices, or to structure files. None of those uses is appropriate for streams of plain text.

4. Before transmission, all character sequences SHOULD be normalized according to Unicode normalization form "NFC" (see Section 3).

5. As suggested in Section 6 of RFC 3629, the Byte Order Mark ("BOM") signature MUST NOT appear at the beginning of these text strings.

6. Systems conforming to this specification MUST NOT transmit any string containing any code point that is unassigned in the version of Unicode on which they are dependent. The version of NFC and the version of Unicode used by that system MUST be consistent.

The use of LF without CR is questionable; see Appendix B for more discussion. The newer control characters IND (U+0084) and NEL ("Next Line", U+0085) might have been used to disambiguate the various line-ending situations, but, because their use has not been established on the Internet, because many protocols require CRLF, and because IND and NEL fall within the "C1 Controls" group (see below), they MUST NOT be used. Similar observations apply to the yet newer line and paragraph separators at U+2028 and U+2029 and any future characters that might be defined to serve these functions. For this specification and protocols that depend on it, lines end in CRLF and only in CRLF. Anything that does not end in CRLF is either not a line or is severely malformed.

The NVT specification contained a number of additional provisions, e.g., for the optional use of backspacing and "bare CR" (sent as CR NUL) to generate overstruck character sequences. The much greater number of precomposed characters in Unicode, the availability of combining characters, and the growing use of markup conventions of various types to show, e.g., emphasis (rather than attempting to do that via the use of special characters), should make such sequences largely unnecessary. These sequences SHOULD be avoided if at all possible. However, because they were optional in NVT applications and this specification is an NVT superset, they cannot be prohibited entirely. The most important of these rules is that CR MUST NOT appear unless it is immediately followed by LF (indicating end of line) or NUL. Because NUL (an octet whose value is all zeros, i.e., %x00 in the notation of [RFC5234]) is hostile to programming languages that use that character as a string delimiter, the CR NUL sequence SHOULD be avoided for that reason as well.

3. Normalization

There are cases where strings of Unicode are fundamentally equivalent, essentially representing the same text. These are called "canonical equivalents" in the Unicode Standard. For example, the following pairs of strings are canonically equivalent:

U+2126 OHM SIGN
U+03A9 GREEK CAPITAL LETTER OMEGA

U+0061 LATIN SMALL LETTER A, U+0300 COMBINING GRAVE ACCENT
U+00E0 LATIN SMALL LETTER A WITH GRAVE

Comparison of strings becomes much easier if any such cases are always represented by a single unique form. The Unicode Consortium specifies a normalization form, known as NFC [NFC], which provides the necessary mappings and mechanisms to convert all canonically equivalent sequences to a single unique form. Typically, this form produces precomposed characters for any sequences that can be represented in that fashion. It also reorders other combining marks so that they have a unique and unambiguous order.

Of the various normalization forms defined as part of Unicode, NFC is closest to actual use in practice, minimizes side-effects due to considering characters equivalent that may not be equivalent in all situations, and typically requires the least work when converting from non-Unicode encodings.

The section above requires that, except in very unusual circumstances, all Net-Unicode strings be transmitted in normalized form. Recognition of the fact that some implementations of applications may rely on operating system libraries over which they have little control and adherence to the robustness principle suggests that receivers of such strings should be prepared to receive unnormalized ones and to not react to that in excessive ways.

4. Versions of Unicode

Unicode changes and expands over time. Large blocks of space are reserved for future expansion. New versions, which appear at regular intervals, add new scripts and characters. Occasionally they also change some property definitions. In retrospect, one of the advantages of ASCII [ASCII] when it was chosen was that the code space was full when the Standard was first published. There was no practical way to add characters or change code point assignments without being obviously incompatible.

While there are some security issues if people deliberately try to trick the system (see Section 6), Unicode version changes should not have a significant impact on the text stream specification of this document for the following reasons:

- The transformation between Unicode code table positions and the corresponding UTF-8 code is algorithmic; it does not depend on whether a code point has been assigned or not.

- The normalization recommended here, NFC (see Section 3), performs a very limited set of mappings, much more limited than those of the more extensive NFKC used in, e.g., Nameprep [RFC3491].

The NFC tables may be updated over time as new characters are added, but the Unicode Consortium has guaranteed the stability of all NFC strings. That is, if a string does not contain any unassigned characters, and it is normalized according to NFC, it will always be normalized according to all future versions of the Unicode Standard. The stability of the Net-Unicode format is thus guaranteed when any implementation that converts text into Net-Unicode format does not permit unassigned characters.

Because Unicode code points that are reserved for private use do not have standard definitions or normalization interpretations, they SHOULD be avoided in strings intended for Internet interchange.

Were Unicode to be changed in a way that violated these assumptions, i.e., that either invalidated the byte string order specified in RFC 3629 or that changed the stability of NFC as stated above, this specification would not apply. Put differently, this specification applies only to versions of Unicode starting with version 5.0 and extending to, but not including, any version for which changes are made in either the UTF-8 definition or to NFC stability. Such changes would violate established Unicode policies and are hence unlikely, but, should they occur, it would be necessary to evaluate them for compatibility with this specification and other Internet uses of NFC.

If the specification of a protocol references this one, strings that are received by that protocol and that appear to be UTF-8 and are not otherwise identified (e.g., by charset labeling) SHOULD be treated as using UTF-8 in conformance with this specification.

5. Applicability and Stability of this Specification

5.1. Use in IETF Applications Specifications

During the development of this specification, there was some confusion about where it would be useful given that, e.g., the individual MIME media types used in email and with HTTP have their own rules about UTF-8 character types and normalization, and the application transport protocols impose their own conventions about line endings. There are three answers. The first is that, in retrospect, it would have been better to have those protocols and content types standardized in the way specified here, even though it is certainly too late to change them at this time. The second is that we have several protocols that are dependent on either the original Telnet design or other arrangements requiring a standard, interoperable, string definition without specific content-labels of one sort or another. Whois [RFC3912] is an example member of this group. As consideration is given to upgrading them for non-ASCII use, this specification provides a normative reference that provides the same stability that NVT has provided the ASCII forms. This specification is intended for use by other specifications that have not yet defined how to use Unicode. Having a preferred standard Internet definition for Unicode text streams -- rather than just one for transmission codings -- may help improve the specification and interoperability of protocols to be developed in the future. This specification is not intended for use with specifications that already allow the use of UTF-8 and precisely define that use.

5.2. Unicode Versions and Applicability

The IETF faces a practical dilemma with regard to versions of Unicode. Each new version brings with it new characters and sometimes new combining characters. Version 5.0 introduces the new concept of sequences of characters named as if they were individual characters (see [NamedSequences]). The normalization represented by NFC is stable if all strings are transmitted and stored in normalized form if corrections are never made to character definitions or normalization tables and if unassigned code points are never used. The latter is important because an unassigned code point always normalizes to itself. However, if the same code point is assigned to a character in a future version, it may participate in some other normalization mapping (some specific difficulties in this regard are discussed in [RFC4690]). It is worth noting that transmission in normalized form is not required by either the IETF's UTF-8 Standard [RFC3629] or by standards dependent on the current version of Stringprep [RFC3454].

All would be well with this as described in Section 4 except for one problem: Applications typically do not perform their own conversions to Unicode and may not perform their own normalizations but instead rely on operating system or language library functions -- functions that may be upgraded or otherwise changed without changes to the application code itself. Consequently, there may be no plausible way for an application to know which version of Unicode, or which version of the normalization procedures, it is utilizing, nor is there any way by which it can guarantee that the two will be consistent.

Because of per-version changes in definitions and tables, Stringprep and documents depending on it are now tied to Unicode Version 3.2 [Unicode32] and full interoperability of Internet Standard UTF-8 [RFC3629], when used with normalization as specified here, is dependent on normalization definitions and the definition of UTF-8 itself not changing after Unicode Version 5.0. These assumptions seem fairly safe, but they are still assumptions. Rather than being linked to the latest available version of Unicode, version 5.0 [Unicode] or broader concepts of version independence based on specific assumptions and conditions, this specification could reasonably have been tied, like Stringprep and Nameprep to Unicode 3.2 [Unicode32] or some more recent intermediate version, but, in addition to the obvious disadvantages of having different IETF standards tied to different versions of Unicode, the library-based application implementation behavior described above makes these version linkages nearly meaningless in practice.

In theory, one can get around this problem in four ways:

1. Freeze on a particular version of Unicode and try to insist that applications enforce that version by, e.g., containing lists of unassigned characters and prohibiting their use. Of course, this would prohibit evolution to include newly-added scripts and the tables of unassigned code points would be cumbersome.

2. Require that every Unicode "text" string or file start with a version indication, somewhat akin to the "byte order mark" indicator. It is unlikely that this provision would be practical. More important, it would require that each application implementation be prepared to either support multiple normalization tables and versions or that it reject text from Unicode versions with which it was not prepared to deal.

3. Devise a different set of normalization rules that would, e.g., guarantee that no character assigned to a previously-unassigned code point in Unicode was ever normalized to anything but itself and use those rules instead of NFC. It is not clear whether or not such a set of rules is possible or whether some other

completely stable set of rules could be devised, perhaps in combination with restrictions on the ways in which characters were added in future versions of Unicode.

4. Devise a normalization process that is otherwise equivalent to NFC but that rejects code points that are unassigned in the current version of Unicode, rather than mapping those code points to themselves. This would still leave some risk of incompatible corrections in Unicode and possibly a few edge cases, but it is probably stable enough for Internet use in the overwhelming number of cases. This process has been discussed in the Unicode Consortium under the name "Stable NFC".

None of these approaches seems ideal: the ideal procedure would be as stable and predictable as ASCII has been. But that level is simply not feasible as long as Unicode continues to evolve by the addition of new code points and scripts. The fourth option listed above appears to be a reasonable compromise.

6. Security Considerations

This specification provides a standard form for the use of Unicode as "network text". Most of the same security issues that apply to UTF-8, as discussed in [RFC3629], apply to it, although it should be slightly less subject to some risks by virtue of requiring NFC normalization and generally being somewhat more restrictive. However, shifts in Unicode versions, as discussed in Section 5.2, may introduce other security issues.

Programs that receive these streams should use extreme caution about assuming that incoming data are normalized, since it might be possible to use unnormalized forms, as well as invalid UTF-8, as part of an attack. In particular, firewalls and other systems that interpret UTF-8 streams should be developed with the clear knowledge that an attacker may deliberately send unnormalized text, for instance, to avoid detection by naive text-matching systems.

NVT contains a requirement, of necessity repeated here (see Section 2), that the CR character be immediately followed by either LF or ASCII NUL (an octet with all bits zero). NUL may be problematic for some programming languages that use it as a string terminator, and hence a trap for the unwary, unless caution is used. This may be an additional reason to avoid the use of CR entirely, except in sequence with LF, as suggested above.

The discussion about Unicode versions above (see Section 4 and Section 5.2) makes several assumptions about future versions of Unicode, about NFC normalization being applied properly, and about

UTF-8 being processed and transmitted exactly as specified in RFC 3629. If any of those assumptions are not correct, then there are cases in which strings that would be considered equivalent do not compare equal. Robust code should be prepared for those possibilities.

7. Acknowledgments

Many thanks to Mark Davis, Martin Duerst, and Michel Suignard for suggestions about Unicode normalization that led to the format described here, and especially to Mark for providing the paragraphs that describe the role of NFC. Thanks also to Mark, Doug Ewell, Asmus Freytag for corrected text describing Unicode transmission forms, and to Tim Bray, Carsten Bormann, Stephane Bortzmeyer, Martin Duerst, Frank Ellermann, Clive D.W. Feather, Ted Hardie, Bjoern Hoehrmann, Alfred Hoenes, Kent Karlsson, Bill McQuillan, George Michaelson, Chris Newman, and Marcos Sanz for a number of helpful comments and clarification requests.

Appendix A. History and Context

This subsection contains a review of prior work in the ARPANET and Internet to establish a standard text type, work that establishes the context and motivation for the approach taken in this document. The text is explanatory rather than normative: nothing in this section is intended to change or update any current specification. Those who are uninterested in this review and analysis can safely skip this section.

One of the earlier application design decisions made in the development of ARPANET, a decision that was carried forward into the Internet, was the decision to standardize on a single and very specific coding for "text" to be passed across the network [RFC0020]. Hosts on the network were then responsible for translating or mapping from whatever character coding conventions were used locally to that common intermediate representation, with sending hosts mapping to it and receiving ones mapping from it to their local forms as needed. It is interesting to note that at the time the ARPANET was being developed, participating host operating systems used at least three different character coding standards: the antiquated BCD (Binary Coded Decimal), the then-dominant major manufacturer-backed EBCDIC (Extended BCD Interchange Code), and the then-still emerging ASCII (American Standard Code for Information Interchange). Since the ARPANET was an "open" project and EBCDIC was intimately linked to a particular hardware vendor, the original Network Working Group agreed that its standard should be ASCII. That ASCII form was precisely "7-bit ASCII in an 8-bit field", which was in effect a compromise between hosts that were natively 7-bit oriented (e.g., with five seven-bit characters in a 36-bit word), those that were 8-bit oriented (using eight-bit characters) and those that placed the seven-bit ASCII characters in 9-bit fields with two leading zero bits (four characters in a 36-bit word).

More standardization was suggested in the first preliminary description of the Telnet protocol [RFC0097]. With the iterations of that protocol [RFC0137] [RFC0139] and the drawing together of an essentially formal definition somewhat later [RFC0318], a standard abstraction, the Network Virtual Terminal (NVT) was established. NVT character-coding conventions (initially called "Telnet ASCII" and later called "NVT ASCII", or, more casually, "network ASCII") included the requirement that Carriage Return followed by Line Feed (CRLF) be the common representation for ending lines of text (given that some participating "Host" operating systems used the one natively, some the other, at least one used both, and a few used neither (preferring variable-length lines with counts or special delimiters or markers instead) and specified conventions for some other characters. Also, since NVT ASCII was restricted to seven-bit

characters, use of the high-order bit in octets was reserved for the transmission of control signaling information.

At a very high level, the concept was that a system could use whatever character coding and line representations were appropriate locally, but text transmitted over the network as text must conform to the single "network virtual terminal" convention. Virtually all early Internet protocols that presume transfer of "text" assume this virtual terminal model, although different ones assume or limit it in different ways. Telnet, the command stream and ASCII Type in FTP [RFC0542], the message stream in SMTP transfer [RFC2821], and the strings passed to finger [RFC0742] and whois [RFC0954] are the classic examples. More recently, HTTP [RFC1945] [RFC2616] follows the same general model but permits 8-bit data and leaves the line end sequence unspecified (the latter has been the source of a significant number of problems).

Appendix B. The ASCII NVT Definition

The main body of this specification is intended as an update to, and internationalized version of, the Net-ASCII definition. The specification is self-contained in that parts of the Net-ASCII definition that are no longer recommended are not included above. Because Net-ASCII evolved somewhat over time and there has been debate about which specification is the "official" Net-ASCII, it is appropriate to review the key elements of that definition here. This review is informal with regard to the contents of Net-ASCII and should not be considered as a normative update or summary of the earlier specifications (Section 2 does specify some normative updates to those specifications and some comments below are consistent with it).

The first part of the section titled "THE NVT PRINTER AND KEYBOARD" in RFC 854 [RFC0854] is generally, although not universally, considered to be the normative definition of the (ASCII) Network Virtual Terminal and hence of Net-ASCII. It includes not only the graphic ASCII characters but a number of control characters. The latter are given Internet-specific meanings that are often more specific than the definitions in the ASCII specification. In today's usage, and for the present specification, the following clarifications and updates to that list should be noted. Each one is accompanied by a brief explanation of the reason why the original specification is no longer appropriate.

1. The "defined but not required" codes -- BEL (U+0007), BS (U+0008), HT (U+0009), VT (U+000B), and FF (U+000C) -- and the undefined control codes ("C0") SHOULD NOT be used unless required by exceptional circumstances. Either their original "network

printer" definitions are no longer in general use, common practice has evolved away from the formats specified there, or their use to simulate characters that are better handled by Unicode is no longer appropriate. While the appearance of some of these characters on the list may seem surprising, BS now has an ambiguous interpretation in practice (erasing in some systems but not in others), the width associated with HT varies with the environment, and VT and FF do not have a uniform effect with regard to either vertical positioning or the associated horizontal position result. Of course, telnet escapes are not considered part of the data stream and hence are unaffected by this provision.

2. In Net-ASCII, CR MUST NOT appear except when immediately followed by either NUL or LF, with the latter (CR LF) designating the "new line" function. Today and as specified above, CR should generally appear only when followed by LF. Because page layout is better done in other ways, because NUL has a special interpretation in some programming languages, and to avoid other types of confusion, CR NUL should preferably be avoided as specified above.

3. LF CR SHOULD NOT appear except as a side-effect of multiple CR LF sequences (e.g., CR LF CR LF).

4. The historical NVT documents do not call out either "bare LF" (LF without CR) or HT for special treatment. Both have generally been understood to be problematic. In the case of LF, there is a difference in interpretation as to whether its semantics imply "go to same position on the next line" or "go to the first position on the next line" and interoperability considerations suggest not depending on which interpretation the receiver applies. At the same time, misinterpretation of LF is less harmful than misinterpretation of "bare" CR: in the CR case, text may be erased or made completely unreadable; in the LF one, the worst consequence is a very funny-looking display. Obviously, HT is problematic because there is no standard way to transmit intended tab position or width information in running text. Again, the harm is unlikely to be great if HT is simply interpreted as one or more spaces, but, in general, it cannot be relied upon to format information.

It is worth noting that the telnet IAC character (an octet consisting of all ones, i.e., %xFF) itself is not a problem for UTF-8 since that particular octet cannot appear in a valid UTF-8 string. However, while few of them have been used, telnet permits other command-introducer characters whose bit sequences in an octet may be part of valid UTF-8 characters. While it causes no ambiguity in UTF-8,

Unicode assigns a graphic character ("Latin Small Letter Y with Diaeresis") to U+00FF (octets C3 B0 in UTF-8). Some caution is clearly in order in this area.

Appendix C. The Line-Ending Problem

The definition of how a line ending should be denoted in plain text strings on the wire for the Internet has been controversial from even before the introduction of NVT. Some have argued that recipients should be required to interpret almost anything that a sender might intend as a line ending as actually a line ending. Others have pointed out that this would lead to some ambiguities of interpretation and presentation and would violate the principle that we should minimize the number of forms that are permitted on the wire in order to promote interoperability and eliminate the "every recipient needs to understand every sender format" problem. The design of this specification, like that of NVT, takes the latter approach. Its designers believe that there is little point in a standard if it is to specify "anyone can do whatever they like and the receiver just needs to cope".

A further discussion of the nature and evolution of the line-ending problem appears in Section 5.8 of the Unicode Standard [Unicode] and is suggested for additional reading. If we were starting with the Internet today, it would probably be sensible to follow the recommendation there and use LS (U+2028) exclusively, in preference to CRLF. However, the installed base of use of CRLF and the importance of forward compatibility with NVT and protocols that assume it makes that impossible, so it is necessary to continue using CRLF as the "New Line Function" ("NLF", see the terminology section in that reference).

Appendix D. A Note about Related Future Work

Consideration should be given to a Telnet (or SSH [RFC4251]) option to specify this type of stream and an FTP extension [RFC0959] to permit a new "Unicode text" data TYPE.

References

Normative References

[ISO10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO/ IEC 10646-1:2000, October 2000.

[NFC] Davis, M. and M. Duerst, "Unicode Standard Annex #15: Unicode Normalization Forms", October 2006, <http://www.unicode.org/reports/tr15/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 5.0", 2007.

Boston, MA, USA: Addison-Wesley. ISBN 0-321-48091-0

[Unicode32] The Unicode Consortium, "The Unicode Standard, Version 3.0", 2000.

(Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5). Version 3.2 consists of the definition in that book as amended by the Unicode Standard Annex #27: Unicode 3.1 (http://www.unicode.org/reports/tr27/) and by the Unicode Standard Annex #28: Unicode 3.2 (http://www.unicode.org/reports/tr28/).

Informative References

[ASCII] American National Standards Institute (formerly United States of America Standards Institute), "USA Code for Information Interchange", ANSI X3.4-1968, 1968.

ANSI X3.4-1968 has been replaced by newer versions with slight modifications, but the 1968 version remains definitive for the Internet. ISO 646 International Reverence Version (IRV) [ISO.646.1991] is usually considered equivalent to ASCII.

[ISO.646.1991] International Organization for Standardization, "Information technology - ISO 7-bit coded character set for information interchange", ISO Standard 646, 1991.

[NamedSequences] The Unicode Consortium, "NamedSequences-4.1.0.txt", 2005, <http://www.unicode.org/Public/UNIDATA/NamedSequences.txt>.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0097] Melvin, J. and R. Watson, "First Cut at a Proposed Telnet Protocol", RFC 97, February 1971.

[RFC0137] O'Sullivan, T., "Telnet Protocol - a proposed document", RFC 137, April 1971.

[RFC0139] O'Sullivan, T., "Discussion of Telnet Protocol", RFC 139, May 1971.

[RFC0318] Postel, J., "Telnet Protocols", RFC 318, April 1972.

[RFC0542] Neigus, N., "File Transfer Protocol", RFC 542, August 1973.

[RFC0698] Mock, T., "Telnet extended ASCII option", RFC 698, July 1975.

[RFC0742] Harrenstien, K., "NAME/FINGER Protocol", RFC 742, December 1977.

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[RFC0954] Harrenstien, K., Stahl, M., and E. Feinler, "NICNAME/WHOIS", RFC 954, October 1985.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, September 2004.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.

Authors' Addresses

John C Klensin
1770 Massachusetts Ave, #322
Cambridge, MA 02140
USA

Phone: +1 617 491 5735
EMail: john-ietf@jck.com

Michael A. Padlipsky
8011 Stewart Ave.
Los Angeles, CA 90045
USA

Phone: +1 310-670-4288
EMail: the.map@alum.mit.edu

Full Copyright Statement

Intellectual Property