Action Plan Background: Plain Text

Author: Andrea Goethals
Release Date: 03/14/03

Change History:

04/08/2004 [AG] Added *Section 4.2 Character Encoding*.
08/14/2007 [CC] Added *Section 2.1 Technical Metadata.*

---

1 General Description

1.1 Format Name: Plain Text

1.2 Version: There are no 'version' numbers for plain text because it is not based on a specification or written standard.

1.3 MIME media type name: text

1.4 MIME subtype: plain

1.5 Short Description: human-readable text file

1.6 Common Extensions: .txt, no extension

1.7 Color depth: Not applicable

1.8 Color Space: Not applicable

1.9 Compression: Not applicable

1.10 Progressive Display: Not applicable

1.11 Animation: Not applicable

1.12 Magic number(s): Not applicable

2 Essential and Distinguishing Characteristics

Plain text files do not contain any formatting commands, font attribute specifications, processing instructions, interpretation directives, encryption, nor content markup. They are intended to be meaningful to a human displayed "as-is". No special software is required to get the full meaning of the text, aside from support for the character set used. Plain text files are linear sequences of characters, possibly interrupted by line breaks or page breaks. The end of each line is marked by special characters, either a CR/LF (carriage return, line feed) sequence, the CR character, or just the LF character. Plain text files do not require much storage space relative to other formats.

A plain text file can not be interpreted correctly without knowing which particular character set the file uses. Older US operating systems used US-ASCII as the default character set

for plain text files, which is why plain text is sometimes referred to as ASCII. More recent operating systems (ex: Windows XP/NT/2000, MacOS X) use Unicode as the default character set for plain text. See *Section 4.2 Character Encoding*.

2.1. Technical Metadata

Technical Metadata Element (G = general file metadata, GT = general text metadata	**Obligation (R = Required by spec., S= Defined in spec., D = Derived from spec., O = Optional)**
Character Set [GT]	D
The origin of character set [GT]	D
Line breaks used [GT]	D
Natural language used [GT]	D
Total number of lines [GT]	D

3 Usefulness

3.1 Version Duration: There is no 'version' for plain text files. What we call 'ASCII' has been around since 1967 and is more correctly called US-ASCII.

3.2 History of Prior Versions Duration: Not applicable

3.3 Expected Newer Versions: Not applicable

3.4 Existence of Publicly Available Complete Specifications: Not applicable

3.5 Specifications-controlling Body: Not applicable

3.6 Related Legal Issues: Not applicable

3.7 Application and Platform Support:
Plain text is the most widely supported file format. It is portable across all operating systems, provided the character set is known and supported.

3.8 Limitations:
Because there is no authoritative standard on plain text, there are different implementations of it, varying things like the end-of-line character and the default character set.

3.9 Perceived Popularity:
Plain text files are not used as much as they used to be, but they are still used for files that need to be small and highly portable, including configuration files, 'readme' files, and software source code.

4 Related Formats

4.1 Specification and File Variations:

4.2 Character Encoding
One of the characteristics of a text file that can be used to distinguish it from other text files is its character encoding. This section provides definitions for some of the major character encoding terms and sets.

ASCII (American Standard Code for Information Interchange)
This character encoding standard was first published in 1964. It is based on a 7-bit byte where 33 code points are control characters and 95 represent characters. There are 128 possible values ranging from 0 to 127 (0x7f). It is only useful to represent the English language. The first 128 Unicode and ISO 8859-1 characters are identical to ASCII.

EBCDIC (Extended Binary Coded Decimal Information Code)
A family of 8-bit encoded standards derived from the 6-bit BCDIC. EBCDIC was used on some IBM computers.

Extended ASCII
Any 8-bit encoding standards that are consistent with ASCII encoding in the first 128 code points.

ISO 8859-1 (Latin 1)
An 8-bit encoding standard used to represent Western languages using the Latin alphabet. The first 256 characters of Unicode are identical to ISO 8859-1. This encoding is being superceded by ISO 8859-15 which deleted some rarely used characters and added some characters like the euro symbol.

Unicode
Uses 16-bit words allowing up to 65,536 characters to be encoded. Unicode's purpose is to support all existing languages as well as classical and historical languages. Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. The Unicode Standard is maintained by the Unicode Consortium (www.unicode.org).

Unicode encoding formats (also called Unicode, or UCS, transformation formats (UTF))
The Unicode Standard's supported encoding formats are UTF-8, UTF-16 and UTF-32. There is a one-to-one correspondence between a Unicode scalar value to the set of code unit sequences for a Unicode encoding form. This ensures that a reverse mapping can always be derived unambiguously. If a Unicode code sequence purporting to be in a Unicode encoding form does not follow the specification of that encoding form it is said to be ill-formed. If it does follow the specification it is said to be well-formed or to be made of valid strings. These encoding formats can be encoded in any of the seven encoding schemes supported by the Unicode Standard.

Unicode encoding schemes
Specified byte serialization for a Unicode encoding format, including the specification of the handling of a byte order mark (BOM).

Unicode string
A code unit sequence containing code units of a particular Unicode encoding form.

Unicode code unit
The minimal bit combination that can represent a unit of encoded text.

Unicode code unit sequence
An ordered sequence of one or more code units that together represent a scalar value.

Unicode scalar value
Any Unicode code point except the high and low surrogate code points. This is the range 0x0000..0xd7ff and 0xe000..0x10ffff, inclusive. (The high surrogate range is 0xd800..0xdbff. The low surrogate range is 0xdc00..0xdfff. These are only used in UTF-16 encoding.)

UTF-8 encoding form
A Unicode encoding that uses 8-bit units ("octets"). Each scalar value is mapped to from one to four bytes in UTF-8 (See Table 1). Valid UTF-8 byte sequences are shown in Table 2. The bytes 0xfe and 0xff are never used in UTF-8. The first byte of a multi-byte sequence which represents a single non-ASCII UCS character is always in the range 0xc0 to 0xfd and indicates how long this multi-byte sequence is. All further bytes in a multi-byte sequence are in the range 0x80 to 0xbf. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.

Scalar value in binary representation	*UTF-8 in binary representation*
000000000xxxxxxx (0x00..0x7f) (0..127)	0xxxxxxx (00..7f)
00000yyyyyxxxxxx (0x80..0x7ff) (128..2,047)	110yyyyy 10xxxxxx (c0..df) (80..bf)
zzzzyyyyyyxxxxxx (0x800..0xd7ff) (2048..55,295) (0xe000..0xffff) (57,344..65,535)	1110zzzz 10yyyyyy 10xxxxxx (e0..ef) (80..bf) (80..bf)
(0x00010000..0x001FFFF) (65,536..2,097,151)	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx (f0..f7) (80..bf) (80..bf) (80..bf)

Table 1: The mapping of Unicode scalar values to UTF-8. Notice that the scalar values in the range (0xd800..0xdfff) are reserved for surrogate pairs, so any UTF-8 byte sequence that maps to that range is ill-formed.

Unicode Code Points	*UTF-8 in hexadecimal representation*			
	1st Byte	*2nd Byte*	*3rd Byte*	*4th Byte*
U+0000..U+007f	0x00..0x7f	-	-	-
U+0080..U+07ff	0xc2..0xdf	0x80..0xbf	-	-
U+0800..U+0fff	0xe0	0xa0..0xbf	0x80..0xbf	-
U+1000..U+cfff	0xe1..0xec	0x80..0xbf	0x80..0xbf	-
U+d000..U+d7ff	0xed	0x80..0x9f	0x80..0xbf	-
U+e000..U+ffff	0xee..0xef	0x80..0xbf	0x80..0xbf	-
U+10000..U+3ffff	0xf0	0x90..0xbf	0x80..0xbf	0x80..0xbf
U+40000..U+fffff	0xf1..0xf3	0x80..0xbf	0x80..0xbf	0x80..0xbf
U+100000..U+10ffff	0xf4	0x80..0x8f	0x80..0xbf	0x80..0xbf

Table 2: Well-formed UTF-8 Byte Sequences. Note that the following byte values are disallowed in UTF-8: c0..c1, and f5..ff.

UTF-8 encoding scheme

The serialized form of UTF-8 is exactly the same as the UTF-8 format encoding. UTF-8 does not need the byte order mark at the beginning of a file but it may be there (in UTF-8 format) if the data was converted from UTF-16 or UTF-32. In these cases, the byte order mark will look like <0xef 0xbb 0xbf> in UTF-8. Having this byte order mark at the beginning of UTF-8 data is not required nor recommended but when encountered at the beginning of the file is extra assurance that it really is UTF-8.

UTF-16 encoding form

A Unicode encoding that uses 16-bit units. The code units have the same value as the scalar values for the ranges 0x0000..0xd7ff and 0xe000..0xffff. Surrogate pairs are used to represent the scalar values in the range 0x10000..0x10ffff. A surrogate pair consists of two bytes - the leading (high surrogate code unit) and trailing (low surrogate code unit) bytes. The leading byte must be in the range 0xd800..0xdbff. The trailing byte must be in the range 0xdc00..0xdfff. Table 2 shows how a UTF-16 encoding form is determined for a scalar value.

Scalar value in binary	***UTF-16***
xxxxxxxxxxxxxxxx (0x00..0x10000) (0..65536)	xxxxxxxx xxxxxxxx (the same number)
000uuuuuvvvvvvxxxxxxxxxx	110110ww wwvvvvvv 110111xx xxxxxxxx (where wwww = uuuuu-1)

Table 3: The mapping of Unicode scalar values to UTF-16. The second row demonstrates the use of surrogate pairs for scalar values greater than or equal to 0x10000.

Byte order mark (BOM)

The Unicode character is U+feff (zero width no-break space). Used to distinguish between UTF-16 (or UTF-32) BE and LE encoding schemes. In the absence of the BOM in UTF-16 encoding it is assumed to be UTF-16 BE. In the absence of the BOM in UTF-32 encoding it is assumed to be UTF-32 BE. *Table 4* shows how the BOM is encoded in each of these encoding schemes.

Encoding scheme	***BOM***
UTF-16 BE	0xfeff
UTF-16 LE	0xfffe
UTF-32 BE	0x0000feff
UTF-32 LE	0xfffe0000

Table 4: The BOM encoding in Unicode encoding schemes.

UTF-16 BE encoding scheme

The big-endian serialization of a UTF-16 code unit sequence. May or may not start with the BOM. The UTF-16 code sequence <0x004d 0x0430 0x4e8c 0xd800 0xdf02> would be serialized as <0xfeff 0x004d 0x0430 0x4e8c 0xd800 0xdf02>, or without the BOM as <0x004d 0x0430 0x4e8c 0xd800 0xdf02>.

UTF-16 LE encoding scheme

The little-endian serialization of a UTF-16 code unit sequence. Must start with the BOM. The UTF-16 code sequence <0x004d 0x0430 0x4e8c 0xd800 0xdf02> would be serialized as <0xfffe 0x4d00 0x3004 0x8c4e 0x00d8 0x02df>.

UTF-32 encoding form
A Unicode encoding that uses 32-bit units. In this scheme, the encoded code unit has the same value as the scalar value. UTF-32 code units in the range 0xd800..0xdfff, as well as any code unit greater than 0x10ffff is ill-formed.

UTF-32 BE encoding scheme
The big-endian serialization of a UTF-32 code unit sequence. May or may not start with the BOM. The UTF-32 code sequence <0x0000004d 0x00000430 0x00004e8c 0x00010302> would be serialized as <0x0000feff 0x0000004d 0x00000430 0x00004e8c 0x00010302>, or without the BOM as <0x0000004d 0x00000430 0x00004e8c 0x00010302>.

UTF-32 LE encoding scheme
The little-endian serialization of a UTF-32 code unit sequence. Must start with the BOM. The UTF-32 code sequence <0x0000004d 0x00000430 0x00004e8c 0x00010302> would be serialized as <0xfffe0000 0x4d000000 0x30040000 0x8c4e0000 0x02030100>.

5 Summary and Conclusions

Plain text files are the most widely supported file format. All that is needed to view one is a very basic text editor, viewer or shell terminal. They have existed for decades.

The only problems encountered with correctly interpreting plain text files tend to be related to the different implementations of it by different operating systems (end-of-line treatment) or by applications intended for different geographic populations (character set issues). The end-of-line treatment is a minimal problem because there are few known variations: CR, LF, or CR/LF. The character set encoding issue is becoming more of an issue as the Internet allows files to be easily shared among populations that use different character sets. There is the beginning of a trend towards operating systems, and to a lesser extent, applications, supporting the Unicode character set. Once Unicode use is more common the character set problem should be reduced, because Unicode is a superset of all other character sets.

6 References

Freed, N.; and N. Borenstein. “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”, Request for Comments: 2046, Network Working Group. November 1996.

Gellons, R; editor. “The Text/Plain Format Parameter”, Request for Comments: 2646, Network Working Group, August 1999.

Gillam, Richard. “Unicode Demystified: A Practical Guide to the Encoding Standard”, Addison-Wesley: Boston, 2003.

[UC 2003] The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1).