Discussion of the Intended Meanings of the Nonprintable ASCII Characters

Michael P. Frank
FAMU-FSU College of Engineering
Friday, September 15, 2006

This document discusses how the invisible ASCII characters (that is, the page C0 control characters, codes 00_{16}-$1F_{16}$, along with the SP (20_{16}) and DEL ($7F_{16}$) characters) were apparently intended to be used originally, and also something about how they are used in practice presently. This discussion is based loosely on ANSI X3.4-1986, ISO 646-1983, ISO/IEC 6429, and other official standards specifications. However, some of the below material is also a matter of interpretation, or educated guesswork based on experience. The symbol glyphs that I use to graphically represent the control characters in the examples are my own notation, and are not standard. The choice of these characters is explained in a companion document, "A Proposed Set of Mnemonic Symbolic Glyphs for the Visual Representation of C0 Controls and Other Nonprintable ASCII Characters."

The section on transmission control characters needs to be updated to incorporate information from ANSI X3.28-1976, *American National Standard for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links*. Likewise, the sections covering the use of the shift in, shift out, and escape characters need to be updated to integrate information from ANSI X3.41-1974, *Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Standard Code for Information Interchange* (or equivalently, ISO 2022-1973).

1. Characters Addressed in this Document

Table 1 gives the complete list of characters whose meanings are discussed in this document.

Table 1. Table of characters discussed in this document. The first column gives the code point in ANSI X3.4's notation. The next four columns give the hexadecimal, decimal, octal, and binary equivalents. Next is the control sequence that can be used for typing the control character on some systems. Next come a couple of choices of glyphs for visual display of the control character, the first of these being the one intended for this use in the Arial Unicode MS font, the other being my suggestion for a more visually suggestive glyph that is still available in that same font. Next is the official 2-3 character abbreviation for the character name, followed by the official character type. Next is the character type, also represented in the background color of the table row, according to OC=other control (magenta), TC=transmission control (red), FE=format effector (green), CE=code extension (yellow), DC=device control (cyan), IS=information separator (blue). Finally is the section number that addresses that character.

x/y	Hex	Dec	Oct	Bin	Ctl	Syms	Abbr	Name	Type	Sec
0/0	00	0	000	000 0000	^@	␀ ∅	NUL	null	OC	2.1
0/1	01	1	001	000 0001	^A	␁ Γ	SOH	start of heading	TC	2.2

x/y	Hex	Dec	Oct	Bin	Ctl	Syms	Abbr	Name	Type	Sec
0/2	02	2	002	000 0010	^B	␂ ⊥	STX	start of text	TC	2.2
0/3	03	3	003	000 0011	^C	␃ ⌋	ETX	end of text	TC	2.2
0/4	04	4	004	000 0100	^D	␄ ⌁	EOT	end of transmission	TC	2.2
0/5	05	5	005	000 0101	^E	␅ ⍰	ENQ	enquiry	TC	2.2
0/6	06	6	006	000 0110	^F	␆ ☑	ACK	acknowledge	TC	2.2
0/7	07	7	007	000 0111	^G	␇ ☎	BEL	bell	OC	2.3
0/8	08	8	010	000 1000	^H	␈ ⌫	BS	backspace	FE	2.4
0/9	09	9	011	000 1001	^I	␉ ↦	HT	horizontal tabulation	FE	2.4
0/10	0A	10	012	000 1010	^J	␊ ↓	LF	line feed	FE	2.4
0/11	0B	11	013	000 1011	^K	␋ ↧	VT	vertical tabulation	FE	2.4
0/12	0C	12	014	000 1100	^L	␌ ⇟	FF	form feed	FE	2.4
0/13	0D	13	015	000 1101	^M	␍ ↲	CR	carriage return	FE	2.4
0/14	0E	14	016	000 1110	^N	␎ ⊙	SO	shift-out	CE	2.5
0/15	0F	15	017	000 1111	^O	␏ ⊗	SI	shift-in	CE	2.5
1/0	10	16	020	001 0000	^P	␐ ⌥	DLE	data link escape	TC	2.2
1/1	11	17	021	001 0001	^Q	␑ ◎	DC1	device control one	DC	2.6
1/2	12	18	022	001 0010	^R	␒ ⊛	DC2	device control two	DC	2.6
1/3	13	19	023	001 0011	^S	␓ ⊜	DC3	device control three	DC	2.6
1/4	14	20	024	001 0100	^T	␔ ⊖	DC4	device control four	DC	2.6
1/5	15	21	025	001 0101	^U	␕ ⍻	NAK	negative acknowledge	TC	2.2
1/6	16	22	026	001 0110	^V	␖ ⌚	SYN	synchronous idle	TC	2.2
1/7	17	23	027	001 0111	^W	␗ ⊡	ETB	end of transmission block	TC	2.2
1/8	18	24	030	001 1000	^X	␘ ✖	CAN	cancel	OC	2.7
1/9	19	25	031	001 1001	^Y	␙ ■	EM	end of medium	OC	2.7
1/10	1A	26	032	001 1010	^Z	␚ �	SUB	substitute character	OC	2.7
1/11	1B	27	033	001 1011	^[	␛ ⇑	ESC	escape	CE	2.7
1/12	1C	28	034	001 1100	^\	␜ ∰	FS	file separator	IS	2.8
1/13	1D	29	035	001 1101	^]	␝ ∯	GS	group separator	IS	2.8
1/14	1E	30	036	001 1110	^^	␞ ∮	RS	record separator	IS	2.8
1/15	1F	31	037	001 1111	^_	␟ ∫	US	unit separator	IS	2.8
2/0	20	32	040	010 0000	^`	␠ ␣	SP	space	FE	2.4
7/15	7F	127	177	111 1111	^?	␡ ⌦	DEL	delete	OC/FE	2.4

2. Character Descriptions

In this section we give our detailed descriptions of the use of the above-listed characters. We break this section up into subsections corresponding to different classes of characters. In some cases (*e.g.* DEL), our classifications below may differ slightly from ANSIs.

2.1. Use of the Null character.

NUL (∅)

The null character NUL (∅; ^@; 00_{16}; $000,0000_2$) is intended to be used as filler for the empty space and time that exists in between meaningful data. For example, the characters ∅∅∅∅∅… may repeat indefinitely throughout the empty regions of a storage medium, or during the time between messages in a transmission medium. See also TC9/SYN (⏲) below, which is used to fill gaps when it is important not to lose synchronization with character boundaries, which may be difficult to discern in the ∅ symbol, due to its all-0s binary pattern. NUL rarely if ever appears as a character within ASCII text files on a storage medium; when it does, its meaning is application-dependent.

In string-manipulation libraries for some programming languages, such as the C language, the NUL character is used the mark the end of a character string in memory. However, a strict reading of the ASCII standard would suggest that this use of NUL is inappropriate for this purpose, since NULs that appear in an ASCII character sequence are supposed to be just skipped over without affecting the reading of the data. Therefore, perhaps a more appropriate way to self-terminate an ASCII string would be to use one of the transmission control characters ETX (⌋), ETB (▣), or EOT (↯), or perhaps one of the information separator characters, such as US (∫), RS (∮), GS (∯), or FS (∰).

2.2. Use of Transmission Control characters

TC1/SOH (⌈), TC2/STX (⊥), TC3/ETX (⌋), TC4/EOT (↯), TC5/ENQ (⍰), TC6/ACK (☑), TC7/DLE (⌥), TC8/NAK (☒), TC9/SYN (⏲), TC10/ETB (▣)

Suppose that a node at one end of an (initially asynchronous) data link wants to send some ASCII text messages to the remote end. The transmission control characters can be used to facilitate a simple protocol for doing so, which works as follows.

Synchronization stage. If necessary, the sender first begins establishing synchronization of character boundaries by sending repeating synchronous idle TC9/SYN (synchronous idle; ⏲; ^V; $16; %001,0110) characters, in other words ⏲⏲⏲… . The importance of this character is that no matter whether one is using a 7-bit or 8-bit encoding, the location of the character boundaries is well defined. For example, in 7-bit encoding, the string …∅⏲… renders as …00000000010110… at this point the receiver knows where the character boundary is, assuming it's expecting to see ⏲ before any other data. Even if the receiver tunes in during the middle of a string of ⏲'s, it simply need look for the repeating pattern 0010110 with the two adjacent 1's, and then it knows the next bit is the last bit of the character. In 8-bit encoding, the same is true.

Handshaking stage. Once the synchronization pattern is established, the sender attempts to initiate contact by sending a TC5/ENQ (enquire; ⍰; ^E; $05; %000,0101) character. If the receiver is ready to receive a message, it responds with a TC6/ACK (acknowledge; ☑; ^F; $06; %000,0110) character, possibly after a sequence of SYN characters if necessary. If the receiver receives the ENQ but is not ready yet, it instead responds with TC8/NAK (negative acknowledge; ⌧; ^U; $15; %001,0101). If NAK is received or no response is received, the sender waits a while and then polls with ⍰ again.

Transmission of messages. Once the receiver is ready, the sender begins sending a sequence of messages. These are of the general format ⌈*message-headers*⊥*message-body-text*⌋ where we're using the character TC1/SOH (start of header; ⌈; ^A; $01; %000,0001) to mark the start of the message header, TC2/STX (start of text; ⊥; ^B; $02; %000,0010) to mark the boundary between the end of the header and the start of the message body text, and TC3/ETX (end of text; ⌋; ^C; $03; %000,0011) to mark the end of the message text. Finally, after all messages in the current transmission have been completed, a TC4/EOT (end of transmission; ↯; ^D; $04; %000,0100) is sent to mark the end of the transmission. After this, sender and receiver may either continue or drop their synchronization signals ⌚, depending on how badly they wish to make sure that the receiver does not lose synchronization before the start of the next message. (If messages need to be received with low latency, it may be useful to keep synchronization.) Also, whether the ENQ/ACK sequence will need to be repeated before the next transmission depends on the details of the protocol.

Data blocking. Finally, some media may require that transmissions be broken up into blocks of limited size for purposes of coding, medium access control, flow control, or other purposes. For example, Ethernet and IP protocols specify a maximum size for their data packets. If the transmission needs to be broken up into blocks like this, each block should be terminated with a ETB (end transmission block; ▣; ^W; $17; %001,0111) character. This tells the receiver that the transmission is being stopped intentionally, rather than accidentally, and that it will resume shortly. Filling any empty space between blocks may be ∅ or ⌚ characters, depending on the stringency of the requirements for maintaining character-boundary synchronization in the meantime. ETBs may come between messages or in the middle of messages, depending on the protocol.

Protocol extensions. In more complex transmission-control protocols, the limited set of characters provided above may not be sufficient. Therefore the DLE (data link escape; ⌥; ^P; 10_{16}; $001,0000_2$) character is provided for extending the protocol. Following this character comes a limited sequence of characters whose interpretation is determined by the specific protocol extension. ASCII itself does not define it.

As an example of the simple transmission control scheme outlined above, here is a complete self-synchronizing transmission with handshaking of four text messages with headers, broken into three blocks:

```
∅∅∅⌚⌚⌚⌚⍰⌚⌚⌚⌚⌚⌈header1⊥text1⌋⌈header2⊥text2⌋▣⌚⌚⌚⌚⌚⌚⌚⌚⌚⌚⌚⌈h
eader3⊥text3⌋▣⌚⌚⌚⌚⌚⌚⌚⌚⌈header4⊥text4⌋↯∅∅∅∅∅∅
```

The presumption here is that the receiver returned an ACK (☑) sometime between when we sent the ENQ (⍰) and the SOH (⌈).

2.3. Use of the Bell character.

BEL (☎)

When sending data to a remote terminal manned by human operators, there may be a need to alert the humans that a particularly important or urgent message is arriving. This may be accomplished by inserting one or more BEL (bell; ☎; ^G; 07_{16}; $000{,}0111_2$) characters into the data stream. These can be arranged to cause a bell to ring or a light to flash at the receiving end, thereby calling attention to the message. (In a more modern environment, we can imagine that the BEL character might be followed by a pager number or instant-messaging identifier which would cause the designated human operator to be alerted regardless of his physical location.)

2.4. Use of the Format Effector characters

FE0/BS (⌫), FE1/HT (↦), FE2/LF (↓), FE3/VT (↧), FE4/FF (⇟), FE5/CR (↵), SP (␣), DEL (⌦)

These characters are intended for specifying the control of text-formatting characters for use with a teletype printer, text CRT terminal, or text editor software. Many modern command-line console applications and terminal emulators (e.g., `cmd` in Windows, `xterm` in Unix) as well as modern word processors such as MS Word handle many of these characters.

FE0/BS (backspace; ⌫; ^H; 08_{16}; $000{,}1000_2$) causes the cursor to move horizontally backwards one character position, optionally erasing or deleting the character it passes over in the process; precise behavior depends on the capabilities and mode settings of the display system. It is usually generated by the Backspace or left-arrow (←) key on a computer keyboard. ⌫ characters are intended primarily for text entry, and are not normally stored in text files. However, sometimes they do appear in files that are generated directly from keyboard transcripts.

FE1/HT (horizontal tab; ↦; ^I; 09_{16}; $000{,}1001_2$) causes the cursor to move horizontally to the next predefined tab stop position. On some systems, these may be predefined to be at multiples of some fixed amount of space from the left margin. The precise behavior depends on the application. HT characters (also called TAB) are usually treated as whitespace characters which are stored along with printable text in a data file. Often, the Tab or ↹ key on the keyboard causes this character to be inserted.

FE2/LF (line feed; ↓; ^J; $0A_{16}$; $000{,}1010_2$) was originally intended to cause teleprinters to advance the paper by one line of text without moving the print head. In some modern systems (particularly Unix systems) it has acquired the more abstract meaning of NL (newline), causing the equivalent of a CR/LF sequence. In other words, it serves the role of a line-break character on these systems. However, other systems interpret it as moving the cursor down one line without causing a carriage return. Therefore, care must be taken when porting files containing LF characters between different systems.

FE3/VT (vertical tab; ↧; ^K; $0B_{16}$; $000{,}1011_2$) is the analogue of HT for text formatting systems that also provide vertical tab stops (*e.g.*, for advancing the cursor vertically to the next predefined region on a preformatted page). However, it is rarely used today. A sensible modern use in text entry might be to advance the cursor to the first line of the next paragraph of text.

FE4/FF (form feed; ⇟; ^L; $0C_{16}$; $000{,}1100_2$) is intended to advance the cursor or print head to the first line of the next page of text. It can be included in text files to serve as a page-break character. In some CRT terminals and terminal emulators, printing this character clears the display. In some text editors and word processors, the Page Down key enters this character, advancing the cursor to the next page or inserting a page break.

FE5/CR (carriage return; ↵; ^M; $0D_{16}$; $000{,}1101_2$) was originally intended to return the teletype print head to the start of the current line. As with LF, some modern systems have co-opted it to serve as a newline or line-break character. Some text display systems might advance the cursor to the next line when CR is printed; others might not. Care must be taken when porting files containing CR characters between different systems. In text entry systems, CR is usually generated by the Return, Enter or ↵ key, and causes the cursor to advance to the start of the next line or a line-break character sequence to be inserted in the text file. However, it may be translated to a CR/LF sequence internally.

SP (space; ␣; 20_{16}; $010{,}0000_2$) is simply an invisible space character that is used as whitespace to separate words. In text entry systems, it is entered with the spacebar key, inserting a space and moving the cursor to the right, and it is stored within text files and transmitted just like as with normal printable characters. However, in some contexts (notably, web URLs), space characters are not allowed, and must be represented with predefined printable character sequences instead.

DEL (delete; ⌦; ^?; $7F_{16}$; $111{,}1111_2$) was originally defined as a RUBOUT character (▩) which was used to manually obliterate characters already printed on punch cards or punched tape by punching out all of the unpunched holes within the character code. It was defined to be interpreted like NUL, that is, as a meaningless pause in the data stream. However, the meaning has since evolved, and now it is used primarily as a text-entry character which usually means, “delete the character immediately following the insertion point.” It is rarely found inserted into text files, but when it is, its meaning is application-dependent. ANSI classifies it as “other control,” but here we classify it as a format effector.

2.5 Use of the Shift characters

LS1/SO (⊙), LS0/SI (⊗)

The characters LS1/SO (shift out; ⊙; ^N; $0E_{16}$; $000{,}1110_2$) and LS0/SI (shift in; ⊗; ^O; $0F_{16}$; $000{,}1111_2$) are intended to extend the capabilities of the graphic character set. In a transmission or data file, characters with codes 21_{16} through $7E_{16}$ that appear between an SO and subsequent SI may be given an alternate interpretation and visual appearance; for example, they might be mapped to characters in some extended character set. ASCII itself does not define what these alternate characters will be. In text entry systems, SO could represent the pressing of a modifier key such as Shift, Ctrl, Alt, or Meta, and SI the later releasing of that key. (However, in practice most systems process keypresses using proprietary keycodes, and only use ASCII for character storage or transmission.) In 8-bit ASCII, the SO and SI characters are called LS1 (Locking Shift 1) and LS0 (Locking Shift 0) respectively, and their defined behavior is slightly different.

2.6. Use of the Device Control characters

DC1 (◎), DC2 (⊛), DC3 (⊜), DC4 (⊖)

These characters are intended for controlling a remote device. This could be an output device such as a printer or terminal emulator, or some other predefined device associated with the receiving node, such as for example a remote device that is transmitting data back to the current node. DC1 (device control 1; ◎; ^Q; 11_{16}; $001{,}0001_2$) is intended to cause the remote device to turn on or resume normal operation. In the XON/XOFF flow control standard, it is the XON character which means that the flow of data back from the remote device should be resumed. DC2 (device control 2; ⊛; ^R; 12_{16}; $001{,}0010_2$) means that the remote device should turn on and go into some predefined special mode. This code is rarely used. DC3 (device control 3; ⊜; ^S; 13_{16}; $001{,}0011_2$) means that the remote device should perform a secondary type stop, such as wait, pause, stand-by, or halt. In XON/XOFF, it causes the remote terminal to pause in its sending of data. Finally, DC4 (device control 4; ⊖; ^T; 14_{16}; $001{,}0100_2$) is a primary stop signal that interrupts the remote device and/or causes it to turn off. All of the DCn character codes are permitted to be used for device-control purposes other than these, if they are not needed for the specific purposes listed. The precise meaning depends on the system.

2.7. Use of Miscellaneous Characters

CAN (✖), EM (■), SUB (�),ESC (⇑)

CAN (cancel; ✖; ^X; 18_{16}; $001{,}1000_2$) means that the preceding data is in error and should be ignored. As the ASCII specification is ambiguous as to how much data is indicated, the precise meaning is up to the application. Since BS already provides a means for character-by-character backwards deletion, a sensible meaning would be for cancel to terminate and disregard the entire current message, transmission block, or transmission.

EM (end medium; ■; ^Y; 19_{16}; $001{,}1001_2$) marks the end of a physical storage medium or the end of the meaningful data stored on that medium. For example, after overwriting the initial portion of a tape, we may terminate the new data with ■ to indicate that the subsequent data is just old left-over content that should be ignored. If we later append more data, it would be appended starting at the ■'s position.

SUB (substitute; �; ^Z; $1A_{16}$; $001{,}1010_2$) is a placeholder for corrupted characters. For example, suppose we are embedding 7-bit ASCII into 8-bit octets where the high bit is used as a parity bit for error detection. If the receiver of a message finds a mismatch in the parity bit, then it knows that the character received is not the one that was sent. It then automatically replaces the corrupted character with � before passing it along to whatever application or user is reading the message. This tells the application or user that this character was bad, and it can then cope with the error at some higher level.

ESC (escape; ⇑; ^[; $1B_{16}$; $001{,}1011_2$) is a character that is used to extend the set of control characters. It is followed by subsequent characters that specify additional codes. ASCII itself does not define the syntax and meaning of these extended codes, but other specifications such as ISO 2022 do. ESC is usually generated by the "Esc" key on a keyboard. Many keyboard-controlled systems interpret it as a command to abort or terminate the current mode or application.

2.8. Use of the Information Separator characters

IS1/US (∫), IS2/RS (∮), IS3/GS (∯), IS4/FS (∰)

These characters are intended to be used for delimiting hierarchically structured data. For example, suppose that the content or body of a message is followed by attachments consisting of a set of database files. Then, individual files within the body may be separated from each other by IS4/FS (file separator; ∰; ^\; $1C_{16}$; $001{,}1100_2$) characters. Within a given database file, there may be several different tables, or in other words groups of records. The different groups would be separated by IS3/GS (group separator; ∯; ^]; $1D_{16}$; $001{,}1101_2$) characters. Within each group, there would be a sequence of records. Records within a group would be separated by IS2/RS (record separator; ∮; ^^; $1E_{16}$; $001{,}1110_2$) characters. Finally, within each data record would be several fields or units of data. The different fields within a record would be separated by IS1/US (unit separator; ∫; ^_; $1F_{16}$; $001{,}1111_2$) characters. So, a given message containing such files might look something like this:

```
⌜From: ∫larry∮
To: ∫mike∮
Subject: ∫HR databases⊥
Mike,  Take  a  look  at  this  file.   It  contains  the
updated employee databases.
∰Last∫First∫SSN∫Date Hired∫Salary∮
Smith∫John∫123-45-6789∫9/11/01∫$50,000∮
Doe∫Jane∫987-65-4321∫10/31/69∫$100,000∮
Gates∫Bill∫666-66-6666∫01/01/40∫$10,000,000∯
Emp SSN∫Manager SSN∮
123-45-6789∫987-65-4321∮
987-65-4321∫666-66-6666∮
666-66-6666∫666-66-6666⌟
```

Here we see a file separator used to separate the message text from a file that contains two tables, or groups of records. The first line (record) in each table gives the names of the fields in that group. The subsequent records in the group give the data. The tables are separated from each other using the group separator character. Finally, the ETX character ends the whole message.

3. Conclusion

In this document, we have summarized, to the best of our knowledge to date, the proper intended (or in some cases, *de facto*) interpretation of the nonprintable ASCII characters. The discussion can likely be further improved after the assimilation of more of the available standards documents.