Chapter 13

Special Areas and Format Characters

This chapter describes several kinds of characters that have special properties as well as areas of codespace that are set aside for special purposes:

- Control Codes
- Layout Controls
- Deprecated Format Characters
- Surrogates
- Private Use Area
- Special Characters

In addition to regular characters, the Unicode Standard contains a number of characters that are not normally rendered directly, but that influence the layout of text or otherwise affect the operation of text processes. They are called format characters.

The Unicode Standard contains code positions for the 64 control characters and the DEL character found in ISO standards and many vendor character sets. The choice of control function associated with a given character code is outside the scope of the Unicode Standard, with the exception of those control characters specified in this chapter.

Layout controls are not themselves rendered visibly, but influence the behavior of algorithms for line breaking, word breaking, glyph selection, and bidirectional ordering.

Surrogate characters are reserved and are to be used in pairs to access 1,048,544 characters.

Private use characters are reserved for private use. Their meaning is defined by private agreement.

The Specials block contains characters that are neither graphic characters nor traditional controls.

13.1 Control Codes

C0 Control Codes: U+0000–U+001F

ASCII C0 Control Codes and Delete. The Unicode Standard makes no specific use of these control codes, but it provides for the passage of the numeric code values intact, neither adding to nor subtracting from their semantics. The semantics of the C0 controls (U+0000..U+001F) and *delete* (U+007F) are generally determined by the application with which they are used. However, in the absence of specific application uses, they may be interpreted according to the semantics specified in ISO/IEC 6429. U+0009 HORIZONTAL TAB is widely used with a consistent set of semantics; therefore these semantics are recognized in the Unicode Standard. (For more information on control codes, see *Section 2.8, Controls and Control Sequences*.)

There is a simple one-to-one mapping between 7-bit (and 8-bit) control codes and Unicode control codes: every 7-bit (or 8-bit) control code is simply zero-extended to a 16-bit code. For example, if LINE FEED (0A) is to be used for terminal control, then the text "WX<LF>YZ" would be transmitted in plain Unicode text as the following 16-bit values: "0057 0058 000A 0059 005A." Any interpretation of these control codes is outside the scope of the Unicode Standard; programmers should refer to a relevant standard (for example, ISO/IEC 6429) that specifies control code interpretations.

Newline Function. One or more of the control codes U+000A LINE FEED, U+000D CARRIAGE RETURN, or the Unicode Equivalent of EBCDIC NL encode a *newline function*. A newline function can act like a *line separator* or a *paragraph separator*, depending on the application. See *Section 13.2, Layout Controls*, for information on how to interpret a line or paragraph separator. The exact encoding of a newline function depends on the application domain. For information on how to identify a newline function, see Unicode Technical Report #13, "Unicode Newline Guidelines," on the CD-ROM or the up-to-date version on the Unicode Web site.

C1 Control Codes: U+0080–U+009F

In extending the 7-bit encoding system of ASCII to an 8-bit system, ISO/IEC 4873 (on which the 8859 family of character standards are based) introduced 32 additional control codes in the range 80–9F hex. Like the C0 control codes, the Unicode Standard makes no specific use of these C1 control codes, but provides for the passage of their numeric code values intact, neither adding to nor subtracting from their semantics. The semantics of the C1 controls (U+0080..U+009F) are generally determined by the application with which they are used. However, in the absence of specific application uses, they may be interpreted according to the semantics specified in ISO/IEC 6429.

13.2 Layout Controls

Layout Controls

The effect of layout controls is specific to particular text processes. As much as possible, layout controls are transparent to those text processes for which they were not intended. In other words, their effects are mutually orthogonal.

Line and Word Breaking

U+00A0 NO-BREAK SPACE has the same width as U+0020 SPACE, but the NO-BREAK SPACE indicates that, under normal circumstances, no line-breaks are permitted between it and surrounding characters, unless the preceding or following character is a line or paragraph separator. U+00A0 NO-BREAK SPACE behaves like the following coded character sequence: U+FEFF ZERO WIDTH NO-BREAK SPACE + U+0020 SPACE + U+FEFF ZERO WIDTH NO-BREAK SPACE. For a complete list of space characters in the Unicode Standard, see *Table 6-1*.

U+00AD SOFT HYPHEN indicates a hyphenation point, where a line-break is preferred when a word is to be hyphenated. Depending on the script, the visible rendering of this character when a line-break occurs may differ (for example, in some scripts it is rendered as a hyphen -, while in others it may be invisible). Contrast this usage with U+2027 HYPHENATION POINT, which is used for a visible indication of the place of hyphenation in dictionaries. For a complete list of dash characters, including all the hyphens, in the Unicode Standard, see *Table 6-2*.

There are two nonbreaking hyphen characters in the Unicode Standard: U+2011 NON-BREAKING HYPHEN and U+0F0C TIBETAN MARK DELIMITER TSHEG BSTAR. See *Section 9.13, Tibetan*, for more discussion of the Tibetan-specific line-breaking behavior.

Zero Width No-Break Space. In addition to the meaning of *byte order mark*, the code value U+FEFF possesses the semantics of ZERO WIDTH NO-BREAK SPACE.

AS ZERO WIDTH NO-BREAK SPACE, U+FEFF behaves like U+00A0 NO-BREAK SPACE in that it indicates the absence of word boundaries; however, the former has no width. For example, this character can be inserted after the fourth character in the text "base+delta" to indicate that there should be no line-break between the "e" and the "+". The ZERO WIDTH NO-BREAK SPACE can be used to prevent line breaking with other characters that do not have nonbreaking variants, such as U+2009 THIN SPACE or U+2015 HORIZONTAL BAR, by bracketing the character.

Zero Width Space. The U+200B ZERO WIDTH SPACE indicates a word boundary, except that it has no width. Zero-width space characters are intended to be used in languages that have no visible word spacing to represent word breaks, such as in Thai or Japanese. When text is justified, ZWSP has no effect on letter spacing—for example, in English or Japanese usage.

There may be circumstances with other scripts, such as Thai, where extra space is applied around ZWSP as a result of justification, as shown in *Figure 13-1*. This approach is unlike the use of fixed-width space characters, such as U+2002 EN SPACE, that have specified width

and should not be automatically expanded during justification (see *Section 6.1, General Punctuation*).

Figure 13-1. Letter Spacing

Type	Justification Examples	Explanation
Memory	the ISP® ZWSP Charts	The ZWSP is inserted to allow linebreak after ®
Display 1	the ISP®Charts	Without letter spacing
Display 2	t h e I S P ® C h a r t s	Normal letter spacing
Display 3	t h e I S P® C h a r t s	"Thai-style" letter spacing
Display 4	t h e I S P ®C h a r t s	ZWSP incorrectly inhibiting letter spacing

Zero-Width Spaces and Joiner Characters. The zero-width spaces are not to be confused with zero-width joiner characters. U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER have no effect on word boundaries, and ZERO WIDTH NO-BREAK SPACE and ZERO WIDTH SPACE have no effect on joining or linking behavior. In other words, the zero-width joiner characters should be ignored when determining word boundaries; ZERO WIDTH SPACE should be ignored when determining cursive joining behavior. See *Cursive Connection* below.

Line and Paragraph Separator. The Unicode Standard provides two unambiguous characters, U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR, to separate lines and paragraphs. They are considered the canonical form of denoting line and paragraph boundaries in Unicode plain text. A new line is begun after each LINE SEPARATOR. A new paragraph is begun after each PARAGRAPH SEPARATOR. As these characters are separator codes, it is not necessary either to start the first line or paragraph or to end the last line or paragraph with them. Doing so would indicate that there was an empty paragraph or line following. The PARAGRAPH SEPARATOR can be inserted between paragraphs of text. Its use allows the creation of plain text files, which can be laid out on a different line width at the receiving end. The LINE SEPARATOR can be used to indicate an unconditional end of line.

A paragraph separator indicates where a new paragraph should start. Any interparagraph formatting would be applied. This formatting could cause, for example, the line to be broken, any interparagraph line spacing to be applied, and the first line to be indented. A *line separator* indicates that a line-break should occur at this point; although the text continues on the next line, it does not start a new paragraph: no interparagraph line spacing or paragraphic indentation is applied.

Cursive Connection

The Non-joiner and Joiner. In some fonts for some scripts, consecutive characters in a text stream may be rendered via adjacent glyphs that cursively join to each other, so as to emulate connected handwriting. For example, cursive joining is implemented in nearly all fonts for the Arabic scripts and in a few handwriting-like fonts for the Latin script.

Cursive rendering is implemented by joining glyphs in the font, plus using a process that selects the particular joining glyph to represent each individual character occurrence, based on the joining nature of its neighboring characters. This glyph selection is implemented in some combination between the rendering engine and the font itself.

In cases where cursive joining is implemented, on occasion an author may wish to override the normal automatic selection of joining glyphs. Typically, this choice is made to achieve one of the following effects:

- Cause nondefault joining appearance (for example, as is sometimes required in writing Persian using the Arabic script).
- Exhibit the joining-variant glyphs themselves in isolation.

The Unicode Standard provides a means to influence joining glyph selection, by means of the two characters U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER. Logically, these characters do not modify the contextual selection process itself, but rather they *change the context* of a particular character occurrence. By providing a non-joining neighbor character where otherwise the neighbor would be joining, or vice versa, they deceive the rendering process into selecting a different joining glyph. This process can be used in two ways:

1. Prevent joining appearance. For example,

ص	U+0635 ARABIC LETTER SAD
ZW NJ	U+200C ZERO WIDTH NON-JOINER
ل	U+0644 ARABIC LETTER LAM

would be rendered as صل (that is, the normal cursive joining of the interior *sad* and *lam* is overridden). Without the ZERO WIDTH NON-JOINER, it would be rendered as ص ل

2. Exhibit joining glyphs in isolation. For example,

ZW J	U+200D ZERO WIDTH JOINER
غ	U+063A ARABIC LETTER GHAIN
ZW J	U+200D ZERO WIDTH JOINER

would be rendered as غ (that is, the medial glyph form of the *ghain* appears in isolation). Without the ZERO WIDTH JOINER before and after, it would be rendered as غ

The preceding examples are adapted from the Iranian national coded character set standard, ISIRI 3342, which defines these characters as “pseudo space” and “pseudo connection,” respectively.

The ZERO WIDTH JOINER does have specific interpretations in certain scripts as specified in this standard. For example, in Indic scripts it provides an invisible neighbor to which a dead consonant may join to induce a half-consonant form (see *Section 9.1, Devanagari*). It is not to be used for arbitrary “glueing” of textual elements together, such as for ligatures or marking index items.

ZERO WIDTH NON-JOINER or ZERO WIDTH JOINER are format control characters. As with other such characters, they should be ignored by processes that analyze text content. For example, a spelling-checker or find/replace operation should filter them out. (See *Section 2.7, Special Character and Noncharacter Values,* for a general discussion of format control characters.)

The effect of these characters in display depends on the context in which they are found. Adding a ZERO WIDTH JOINER between characters that are already cursively connected will

have no effect. Adding a ZERO WIDTH NON-JOINER between characters that are unconnected will also have no effect. For example, any number of ZERO WIDTH NON-JOINER or ZERO WIDTH JOINER characters sprinkled into an English text stream will have no effect on its appearance when rendered in a typical noncursive Latin font.

Controlling Ligatures

Although ZERO WIDTH JOINER and ZERO WIDTH NON-JOINER should not affect ligating behavior, in some systems they may break up ligatures by interrupting the character sequence required to form the ligature. For example, a cursive Latin font would produce the results shown in *Figure 13-2*.

Figure 13-2. Ligature Example

Memory Representation	Rendering	
f i s h	f- -i- -s- -h	
	fi- -s- -h	optionally using a ligature
f ZWJ i s h	f- -i- -s- -h	
	fi- -s- -h	optionally using a ligature
f ZWNJ i s h	f -i- -s- -h	
f ZWJ ZWNJ i s h	f- -i- -s- -h	
f ZWNJ ZWJ i s h	f -i- -s- -h	

Usage of optional ligatures such as *fi* is not currently controlled by any codes within the Unicode Standard but is determined by protocols or resources external to the text sequence where hyphens indicate cursive joining.

Bidirectional Ordering Codes

These codes are used in the bidirectional algorithm, described in *Chapter 3, Conformance*. Systems that handle bidirectional scripts (Arabic and Hebrew) should be sensitive to these codes. The codes appear in *Table 13-1*.

Table 13-1. Bidirectional Ordering Codes

Code	Name	Abbrev.
U+200E	LEFT-TO-RIGHT MARK	LRM
U+200F	RIGHT-TO-LEFT MARK	RLM
U+202A	LEFT-TO-RIGHT EMBEDDING	LRE
U+202B	RIGHT-TO-LEFT EMBEDDING	RLE
U+202C	POP DIRECTIONAL FORMATTING	PDF
U+202D	LEFT-TO-RIGHT OVERRIDE	RLO
U+202E	RIGHT-TO-LEFT OVERRIDE	LRO

As with the other zero-width character codes, except for their effect on the layout of the text in which they are contained, the bidirectional ordering characters can be ignored by the processing software. For nonlayout text processing, such as sorting, searching, and so on, the zero-width layout characters may be ignored. However, operations that modify text must maintain these characters correctly, because the matching pairs of zero-width formatting characters must be coordinated (see *Chapter 3, Conformance*).

U+200E LEFT-TO-RIGHT MARK and U+200F RIGHT-TO-LEFT MARK have the semantics of an invisible character of zero width, except that these characters have strong directionality. They are intended to be used to resolve cases of ambiguous directionality in the context of bidirectional texts. Unlike U+200B ZERO WIDTH SPACE, these characters carry no word-break semantics. (See *Section 3.12, Bidirectional Behavior*, for more information.)

13.3 Deprecated Format Characters

Deprecated Format Characters: U+206A–U+206F

Three pairs of deprecated format characters are encoded in this block:

- Symmetric swapping format characters used to control the glyphs that depict characters such as "(". (The default state is *activated.*)
- Character shaping selectors used to control the shaping behavior of the Arabic compatibility characters. (The default state is *inhibited.*)
- Numeric shape selectors used to override the normal shapes of the Western Digits. (The default state is *nominal.*)

The use of these character shaping selectors and codes for digit shapes is *strongly* discouraged in the Unicode Standard. Instead, the appropriate character codes should be used with the default state. For example, if contextual forms for Arabic characters are desired, then the nominal characters should be used, and not the presentation forms with the shaping selectors. Similarly, if the Arabic digit forms are desired, then the explicit characters should be used, such as U+0660 ARABIC-INDIC DIGIT ZERO.

Symmetric Swapping. The symmetric swapping format characters are used in conjunction with the class of left- and right-handed pairs of characters (symmetric characters), such as parentheses. The characters thus affected are listed in *Section 4.7, Mirrored—Normative.* They indicate whether the interpretation of the term LEFT or RIGHT in the character names should be interpreted as meaning *opening* or *closing*, respectively. They do not nest. The default state of symmetric swapping may be set by a higher-level protocol or standard, such as ISO 6429. In the absence of such a protocol, the default state is *activated*.

From the point of encountering U+206A INHIBIT SYMMETRIC SWAPPING format character up to a subsequent U+206B ACTIVATE SYMMETRIC SWAPPING (if any), the symmetric characters will be interpreted and rendered as left and right.

From the point of encountering U+206B ACTIVATE SYMMETRIC SWAPPING format character up to a subsequent U+206A INHIBIT SYMMETRIC SWAPPING (if any), the symmetric characters will be interpreted and rendered as opening and closing. This state (*activated*) is the default state in the absence of any symmetric swapping code or a higher-level protocol.

Character Shaping Selectors. The character shaping selector format characters are used in conjunction with Arabic presentation forms. During the presentation process, certain letterforms may be joined together in cursive connection or ligatures. The shaping selector codes indicate that the character shape determination (glyph selection) process used to achieve this presentation effect is to be either activated or inhibited. The shaping selector codes do not nest.

From the point of encountering a U+206C INHIBIT ARABIC FORM SHAPING format character up to a subsequent U+206D ACTIVATE ARABIC FORM SHAPING (if any), the character shaping determination process should be inhibited. If the backing store contains Arabic presentation forms (for example, U+FE80..U+FEFC), then these forms should be presented without shape modification. This state (*inhibited*) is the default state in the absence of any character shaping selector or a higher-level protocol.

From the point of encountering a U+206D ACTIVATE ARABIC FORM SHAPING format character up to a subsequent U+206C INHIBIT ARABIC FORM SHAPING (if any), any Arabic

presentation forms that appear in the backing store should be presented with shape modification by means of the character shaping (glyph selection) process.

The shaping selectors have no effect on nominal Arabic characters (U+0660..U+06FF), which are always subject to character shaping (glyph selection) and which are unaffected by these formatting codes.

Numeric Shape Selectors. The numeric shape selector format characters allow the selection of the shapes in which the digits U+0030..U+0039 are to be rendered. These format characters do not nest.

From the point of encountering a U+206E NATIONAL DIGIT SHAPES format character up to a subsequent U+206F NOMINAL DIGIT SHAPES (if any), the European digits (U+0030..U+0039) should be depicted using the appropriate national digit shapes as specified by means of appropriate agreements. For example, they could be displayed with shapes such as the ARABIC-INDIC DIGITS (U+0660..U+0669). The actual character shapes (glyphs) used to display national digit shapes are not specified by the Unicode Standard.

From the point of encountering a U+206F NOMINAL DIGIT SHAPES format character up to a subsequent U+206E NATIONAL DIGIT SHAPES (if any), the European digits (U+0030..U+0039) should be depicted using glyphs that represent the nominal digit shapes shown in the code tables for these digits. This state (*nominal*) is the default state in the absence of any numeric shape selector or a higher-level protocol.

13.4 Surrogates Area

Surrogates Area: U+D800–U+DFFF

The Surrogates Area consists of 1,024 low-half surrogate code values and 1,024 high-half surrogate code values, which are interpreted in pairs to access more than 1 million code points. *Except for private use, there are no such characters currently assigned in this version of this standard.* For the formal definition of a *surrogate pair* and the role of surrogate pairs in the Unicode Conformance Clause, see *Section 3.7, Surrogates,* and *Section 5.4, Handling Surrogate Pairs*.

The use of surrogate pairs in the Unicode Standard is formally equivalent to the Universal Transformation Format-16 (UTF-16) defined in ISO 10646. (For a complete statement of the UTF-16 extension mechanism, see *Appendix C, Relationship to ISO/IEC 10646*.)

High-Surrogate. The high-surrogate code values are assigned to the range U+D800.. U+DBFF. The high-surrogate code value is always the first element of a surrogate pair.

Low-Surrogate. The low-surrogate code values are assigned to the range U+DC00.. U+DFFF. The low-surrogate code value is always the second element of a surrogate pair.

Private-Use High-Surrogates. The high-surrogate code values from U+DB80..U+DBFF are private-use high-surrogate code values (a total of 128 code values). Characters represented by means of a surrogate pair, where the high-surrogate code value is a private-use high-surrogate, are private-use characters. This mechanism allows for a total of 131,068 ($= 128 \times 1024 - 4$) private-use characters representable by means of surrogate pairs. (For more information on private-use characters, see the discussion of the Private Use Area.)

The code tables do not have charts or name list entries for the range D800..DFFF because individual, unpaired surrogates merely have code values.

13.5 Private Use Area

Private Use Area: U+E000–U+F8FF

The Private Use Area is reserved for use by software developers and end users who need a special set of characters for their application programs. The code points in this area are reserved for private use and do not have defined, interpretable semantics except by private agreement.

There are no charts for this area, as any character encoded in this area is privately defined.

There are also private-use characters defined by means of surrogate pairs. (See the Surrogates Area description for a specification of how those private-use characters are encoded.)

Encoding Structure. By convention, the Private Use Area is divided into a Corporate Use subarea, starting at U+F8FF and extending downward in values, and an End User subarea, starting at U+E000 and extending upward.

Corporate Use Subarea. Systems vendors and/or software developers may need to reserve some private-use characters for internal use by their software. The Corporate Use subarea is the preferred area for such reservations. Assignments of character semantics in this subarea could be completely internal, hidden from the end users, and used only for vendor-specific application support, or they could be published as vendor-specific character assignments available to applications and end users. An example of the former case would be the assignment of a character code to a system support operation such as <MOVE> or <COPY>; an example of the latter case would be the assignment of a character code to a vendor-specific logo character such as Apple's *apple* character.

End User Subarea. The End User subarea is intended for private-use character definitions by end users or for scratch allocations of character space by end-user applications.

Allocation of Subareas. Vendors may choose to reserve private-use codes in the Corporate Use subarea and make some defined portion of the End User subarea available for completely free end-user definition. This convention is for the convenience of system vendors and software developers. No firm dividing line between the two subareas is defined, as different users may have different requirements. No provision is made in the Unicode Standard for avoiding a "stack-heap collision" between the two subareas in the Private Use Area.

Promotion of Private-Use Characters. In future versions of the Unicode Standard, some characters that have been defined by one vendor or another in the Corporate Use subarea may be encoded elsewhere as regular Unicode characters if their usage is widespread enough that they become candidates for general use. The code positions in the Private Use Area are permanently reserved for private use—no assignment to a particular set of characters will ever be endorsed by the Unicode Consortium.

13.6 Specials

Specials: U+FEFF, U+FFF0–U+FFFF

The Specials block contains code values that are interpreted neither as control nor graphic characters but that are provided to facilitate current software practices. The 14 Unicode values from U+FFF0..U+FFFD are reserved for special character definitions. In addition to these 14 positions, 2 code values are specified here for use not as characters but as special signaling devices (described below).

Byte Order Mark (BOM)

There are two circumstances where the character U+FEFF can have a special interpretation that is different from the normal semantics of a *zero width no-break space* (see *Section 13.2, Layout Controls*):

1. Unmarked Byte Order. Some machine architectures use the so-called big-endian byte order, while others use the little-endian byte order. When Unicode text is serialized into bytes, the bytes can go in either order, depending on the architecture. However, sometimes this byte order is not externally marked, which causes problems in interchange between different systems.
2. Unmarked Character Set. In some circumstances, the character set information for a stream of coded characters (such as a file) is not available. The only information available is that the stream contains text, but the precise character set is not known.

In these two cases, the character U+FEFF can be used as a signature to indicate the byte order and the character set by using the UTF-16 serialization described in *Section 3.8, Transformations*. Because the byte-swapped version U+FFFE is always an illegal Unicode value, when an interpreting process finds U+FFFE as the first character, it signals either that the process has encountered text that is of the incorrect byte order or that the file is not valid Unicode text.

In the UTF-16 serialization, U+FEFF at the very beginning of a file or stream explicitly signals the byte order.

The byte sequence FE_{16} FF_{16} may serve as a signature to identify a file as containing Unicode text. This sequence is exceedingly rare at the outset of text files using other character encodings, whether single- or multiple-byte, and therefore not likely to be confused with real text data. For example, in systems that employ ISO Latin 1 (ISO/IEC 8859-1) or the Microsoft Windows ANSI Code Page 1252, the byte sequence FE_{16} FF_{16} constitutes the string *thorn, y diaeresis* "þÿ"; in systems that employ the Apple Macintosh Roman character set or the Adobe Standard Encoding, this sequence represents the sequence *ogonek, hacek* "˛ˇ"; in systems that employ other common IBM PC Code Pages (for example, CP 437, 850), this sequence represents *black square, no-break space* "■ ".

In UTF-8, the BOM corresponds to the byte sequence EF_{16} BB_{16} BF_{16}. Although there are never any questions of byte order with UTF-8 text, this sequence can serve as signature for UTF-8 encoded text where the character set is unmarked. As with a BOM in UTF-16, this sequence of bytes will be extremely rare at the beginning of text files in other character encodings. For example, in systems that employ Microsoft Windows ANSI Code Page 1252, EF_{16} FF_{16} BF_{16} corresponds to the sequence *i diaeresis, guillemet, inverted question mark* "ï » ¿".

Where the character set information is explicitly marked, such as in UTF-16BE or UTF-16LE, then all U+FEFF characters, even at the very beginning of the text, are to be interpreted as *zero width no-break spaces*. Similarly, where Unicode text has known byte order, initial U+FEFF characters are also not required and are to be interpreted as *zero width no-break spaces*. For example, for strings in an API, the memory architecture of the processor provides the explicit byte order. For databases and similar structures, it is much more efficient and robust to use a uniform byte order for the same field (if not the entire database), thereby avoiding use of the *byte order mark*.

Systems that use the *byte order mark* must recognize that an initial U+FEFF signals the byte order; it is not part of the textual content. It should be removed before processing, because otherwise it may be mistaken for a legitimate *zero width no-break space*. To represent an initial U+FEFF ZERO WIDTH NO-BREAK SPACE in a UTF-16 file, use U+FEFF twice in a row. The first one is a *byte order mark*; the second one is the initial *zero width no-break space*. See *Table 13-2* for a summary of encoding form signatures.

Table 13-2. Unicode Encoding Form Signatures

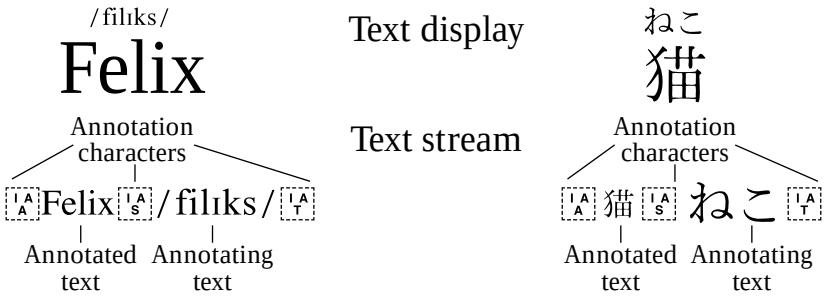
Encoding Form	Signature
UTF-8	EF BB BF
UTF-16 Big-Endian	FE FF
UTF-16 Little-Endian	FF FE
UCS-4[a] Big-Endian	00 00 FE FF
UCS-4 Little-Endian	FF FE 00 00

a. For UCS-4, see *Section C.2, Encoding Forms in ISO/IEC 10646*.

Annotation Characters

An *interlinear annotation* consists of *annotating text* that is related to a sequence of *annotated* characters. For all regular editing and text-processing algorithms, the annotated characters are treated as part of the text stream. The annotating text is also part of the content, but for all or some text processing, it does not form part of the main text stream. However, within the annotating text, characters are accessible to the same kind of layout, text-processing, and editing algorithms as the base text. The *annotation character*s delimit the annotating and the annotated text, and identify them as part of an annotation. See *Figure 13-3*.

Figure 13-3. Annotation Characters



The annotation characters are used in internal processing when out-of-band information is associated with a character stream, very similarly to the usage of the U+FFFC OBJECT REPLACEMENT CHARACTER. However, unlike the opaque objects hidden by the latter character, the annotation itself is textual.

Conformance. A conformant implementation that supports annotation characters interprets the base text as if it were part of an unannotated text stream. Within the annotating text, it interprets the annotating characters with their regular Unicode semantics.

U+FFF9 INTERLINEAR ANNOTATION ANCHOR is an anchor character, preceding the interlinear annotation. The exact nature and formatting of the annotation is dependent on additional information that is not part of the plain text stream. This situation is analogous to that for U+FFFC OBJECT REPLACEMENT CHARACTER.

U+FFFA INTERLINEAR ANNOTATION SEPARATOR separates the base characters in the text stream from the annotation characters that follow. The exact interpretation of this character depends on the nature of the annotation. More than one separator may be present. Additional separators delimit parts of a multipart annotating text.

U+FFFB INTERLINEAR ANNOTATION TERMINATOR terminates the annotation object (and returns to the regular text stream).

Use in Plain Text. Usage of the annotation characters in plain text interchange is strongly discouraged without prior agreement between the sender and the receiver because the content may be misinterpreted otherwise. Simply filtering out the annotation characters on input will produce an unreadable result or, even worse, an opposite meaning. On input, a plain text receiver should either preserve all characters or remove the interlinear annotation characters as well as the annotating text included between the INTERLINEAR ANNOTATION SEPARATOR and the INTERLINEAR ANNOTATION TERMINATOR.

When an output for plain text usage is desired and when the receiver is unknown to the sender, these interlinear annotation characters should be removed as well as the annotating text included between the INTERLINEAR ANNOTATION SEPARATOR and the INTERLINEAR ANNOTATION TERMINATOR.

This restriction does not preclude the use of annotation characters in plain text interchange, but it requires a prior agreement between the sender and the receiver for correct interpretation of the annotations.

Lexical Restrictions. If an implementation encounters a paragraph break between an *anchor* and its corresponding *terminator,* it shall terminate any open annotations at this point. Anchor characters must precede their corresponding terminator characters. Unpaired anchors or terminators shall be ignored. A *separator* occurring outside a pair of delimiters, shall be ignored. Annotations may be nested.

Formatting. All formatting information for an annotation is provided by higher-level protocols. The details of the layout of the annotation are implementation-defined. Correct formatting may require additional information not present in the character stream, but maintained out of band. Therefore, annotation markers serve as placeholders for an implementation that has access to that information from another source.

Collation. Except for the special case where the annotation is intended to be used as a sort-key, annotations are typically ignored for collation, or optionally preprocessed to act as tie breakers only. Importantly, annotation base characters are not ignored, but treated like regular text.

Replacement Characters

U+FFFC. The U+FFFC OBJECT REPLACEMENT CHARACTER is used as an insertion point for objects located within a stream of text. All other information about the object is kept outside the character data stream. Internally it is a dummy character that acts as an anchor point for the object's formatting information. In addition to assuring correct placement of

an object in a data stream, the object replacement character allows the use of general stream-based algorithms for any textual aspects of embedded objects.

U+FFFD. The U+FFFD REPLACEMENT CHARACTER is the general substitute character in the Unicode Standard. That character can be substituted for any "unknown" character in another encoding that cannot be mapped in terms of known Unicode values (see *Section 5.3, Unknown and Missing Characters*).

Noncharacters

U+FFFE. The 16-bit unsigned hexadecimal value U+FFFE is *not* a Unicode character value. Its occurrence in a stream of Unicode data strongly suggests that the Unicode characters should be byte-swapped before interpretation. U+FFFE should be interpreted only as an incorrectly byte-swapped version of U+FEFF ZERO WIDTH NO-BREAK SPACE, also known as the *byte order mark*.

U+FFFF. The 16-bit unsigned hexadecimal value U+FFFF is *not* a Unicode character value; it may be used by an application as a error code or other noncharacter value. The specific interpretation of U+FFFF is not defined by the Unicode Standard, so it can be viewed as a kind of private-use noncharacter.

This PDF file is an excerpt from *The Unicode Standard, Version 3.0*, issued by the Unicode Consortium and published by Addison-Wesley. The material has been modified slightly for this online edition, however the PDF files have not been modified to reflect the corrections found on the Updates and Errata page (see http://www.unicode.org/unicode/uni2errata/UnicodeErrata.html). More recent versions of the Unicode standard exist (see http://www.unicode.org/unicode/standard/versions/).

Dai Kan-Wa Jiten used as the source of reference Kanji codes was written by Tetsuji Morohashi and published by Taishukan Shoten.

ISBN 0-201-61633-5

This book is set in Minion, designed by Rob Slimbach at Adobe Systems, Inc. It was typeset using FrameMaker 5.5 running under Windows NT. ASMUS, Inc. created custom software for chart layout. The Han radical-stroke index was typeset by Apple Computer, Inc. The following companies and organizations supplied fonts:

Apple Computer, Inc.
Atelier Fluxus Virus
Beijing Zhong Yi (Zheng Code) Electronics Company
DecoType, Inc.
IBM Corporation
Monotype Typography, Inc.
Microsoft Corporation
Peking University Founder Group Corporation
Production First Software

Additional fonts were supplied by individuals as listed in the *Acknowledgments*.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information please contact:

Corporate, Government, and Special Sales
Addison Wesley Longman, Inc.
One Jacob Way
Reading, Massachusetts 01867

Visit A-W on the Web: http://www.awl.com/cseng/

First printing, January 2000.