L2/12-355

[contents]

This document is an editor's copy. It supports markup to identify changes from a previous version. Two kinds of changes are highlighted: new, added text, and ~~deleted text~~.

Unicode in XML and other Markup Languages

Unicode Technical Report #20

W3C Working Group Note (Editor's Draft) 12 October 2012

This version:
http://www.unicode.org/reports/tr20/tr20-9.html
http://www.w3.org/International/docs/unicode-xml/
Latest version:
http://www.unicode.org/reports/tr20/
http://www.w3.org/TR/unicode-xml/
Previous version:
http://www.unicode.org/reports/tr20/tr20-8.html
http://www.w3.org/TR/2007/NOTE-unicode-xml-20070516/
Date (Unicode):
2012-10-12
Revision (Unicode):
9

Authors:
Martin Dürst (duerst@it.aoyama.ac.jp)
Asmus Freytag (asmus@unicode.org)
Editor:
Richard Ishida, W3C

Abstract

This document contains guidelines on the use of the Unicode Standard in conjunction with markup languages such as XML.

Status of This Document (common)

This is a proposed update to a Technical Report published jointly by the Unicode Technical Committee and by the W3C Internationalization Working Group/Interest Group (W3C Members only) in the context of the W3C Internationalization Activity. This is a draft document which may be updated, replaced, or superseded by other documents at any time. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.

The base version of the Unicode Standard for this document is Version ~~5~~6.2. For more information about versions of the Unicode Standard, see http://www.unicode.org/standard/versions/. Both the Unicode Standard and markup technologies are evolving. When appropriate, a new version of this document may be published.

Please mail corrigenda and other comments to the authors or use the reporting form.

Status of This Document (Unicode Consortium)

This document is a proposed update of a previously approved **Unicode Technical Report**. Publication does not imply endorsement by the Unicode Consortium.

A Unicode Technical Report (UTR) contains informative material. Conformance to the Unicode Standard does not imply conformance to any UTR. Other specifications, however, are free to make normative references to a UTR.

For a list of current Unicode Technical Reports see http://www.unicode.org/reports.

Status of This Document (W3C)

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a proposed update to a Note that has been previously endorsed by the W3C Internationalization Working Group/Interest Group, but has not been reviewed or endorsed by W3C Members.

This W3C Working Group Note was produced by the i18n Core Working Group, part of the Internationalization Activity. Please send comments related to this document to ~~www-i18n-comments@w3.org (public archive)~~www-international@w3.org (public archive). Use "[unicode-xml]" in the subject line of your email.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. At the time of publication, work on this document was considered complete and no further revisions are anticipated. It is a stable document and may be used as reference material or cited from another document. However, this document may be updated, replaced, or made obsolete by other documents at any time.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction

The Unicode Standard [Unicode] defines the universal character set. Its primary goal is to provide an unambiguous encoding of the content of plain text, ultimately covering all languages in the world, but also major text-based notational systems for science, technology, music, and scholarship.

Currently in its ~~fifth~~ sixth major version, Unicode contains a large number of characters covering most of the currently used scripts in the world. It also contains additional characters for interoperability with older character encodings, and characters with control-like functions included primarily for reasons of providing unambiguous interpretation of plain text. Unicode provides specifications for use of all of these characters.

For document and data interchange, the Internet and the World Wide Web make extensive use of marked-up text such as HTML4.01 and XML. In many instances, markup provides the same, or essentially similar features to those provided by format characters in the Unicode Standard for use in plain text. Another special character category provided by Unicode are compatibility characters. While there may be valid reasons to support these characters and their specifications in plain text, their use in marked-up text can conflict with the rules of the markup language. Formatting characters are discussed in Section 3, *Characters not Suitable for Use With Markup* and Section 4, *Format Characters Suitable for Use With Markup,* compatibility characters in Section 5,*Characters with Compatibility Mappings* . Section 6 briefly discusses noncharacters, and Section 7 is devoted to white space.

~~Issues resulting from canonical equivalences and Normalization [Normalization] as well as t~~The interaction of character encoding and methods of escaping characters in markup are discussed in the Character Model for the World Wide Web [Charmod]~~ and [Charmodnorm]~~.

The issues of using Unicode characters with marked-up text depend to some degree on the rules of the markup language in question and the set of elements it contains. In a narrow sense, this document concerns itself only with XML, and to some extent HTML. However, much of the general information presented here should be useful in a broader context, including some page layout languages.

> **Note:** Many of the recommendations of this report depend on the availability of particular markup or styling. Where possible, appropriate DTDs or Schemas should be used or designed to make such markup or styling available, or the DTDs or Schemas used should be appropriately extended. The current version of this document makes no specific recommendations for the design of DTDs or Schemas, or for the use of particular DTDs or Schemas, but the information presented here may be useful to designers of DTDs and Schemas, and to people selecting DTDs or Schemas for their applications.

Note: The recommendations of this report do not apply in the case of XML used for blind data transport and similar cases.

1.1 Notation

This report uses XML [XML] as a prominent and general example of markup. The XML namespace notation [Namespace] is used to indicate that a certain element is taken from a specific markup language. As an example, the prefix 'xhtml:' indicates that this element is taken from [XHTML]. This means that the examples containing the namespace prefix 'xhtml:' are assumed to include a namespace declaration of xmlns:xhtml="..."

Characters are denoted using the notation used in the Unicode Standard, that is, an optional U+ followed by their hexadecimal number, using at least 4 digits, such as "U+1234" or "U+10FFFD". In XML or HTML this could be expressed as "&#x1234;" or "&#x10FFFD;".

2. General Considerations

There are several general points to consider when looking at the interaction between character encoding and markup.

- Linearity of text vs. hierarchy of markup structure
- Overlap of control codes and markup semantics
- Markup *vs.* Styling
- Coincidence of semantic markup and functions
- Extensibility of markup

2.1 Linearity versus Structure

Encoding text as a sequence of characters without further information leads to a linear sequence, commonly called plain text. Character follows character, without any particular structure. Markup, on the other hand, defines a hierarchical structure for the text or data. In the case of XML and most other, similar markup languages, the markup defines a tree structure. While this tree structure is linearized for transmission in the XML document, once the document has been parsed, the tree is available directly.

Operations that are easy to perform on trees are often difficult to perform on linear sequences and vice versa. By separating functionality between character encoding and markup appropriately, the architecture becomes simpler, more powerful and longer-lasting.

In particular, operations on hierarchical structures can easily make sure that information is kept in context. Attributes assigned to parts of a document are moved together with the associated part of the document. Assigning an attribute to a part of a document limits the scope of the attribute to that part of the document. Performing the same operations on linear sequences of characters using control codes to set attributes and to delimit their scope requires much more work and is error prone. Locating the start or end of a span of text of the same attribute requires scanning backwards and forwards for the embedded delimiter or control code. Moving or editing text often results in mismatched control codes, so that an attribute might suddenly apply to text it was not intended for.

2.2 Overlap of Control Code and Markup Semantics

When markup is not available, plain text may require control characters. This is usually the case where plain text must contain some scoping or attribute information in order to be legible, *i.e.* to be able to transmit the same content between originator and receiver. Many of these control characters have direct equivalents in particular markup languages, since markup handles these concerns efficiently. If both characters and their markup equivalents may be present in the same text, the question of priority is raised. Therefore it is important to identify and resolve these ambiguities at the time markup is first applied.

2.3 Markup and Styling

Besides the basic character encoding and text markup there is a third contributor to text functionality, namely styling. Markup is concerned with the logical structure of the text or data, *e.g.* to indicate sections, subsections, and headers in a document, or to indicate the various fields of an address record. Styling is used to present the information in various ways, *e.g.* in different fonts, different type styles (italic, bold), different colors, *etc.* Some character codes do not encode a generic character, but a styled character. Where these characters are used, styling information is frozen, *i.e.* it is no longer possible to alter the appearance of the text by applying style information. However, there are many examples where a historically free stylistic variation has over time become a semantic distinction that is properly encoded as plain text. Sometimes, what is a free variation in some contexts, implies strict semantic differentiation in others. In all such instances, altering the appearance of the text by styling information would irreparably alter the content of the text. This is of particular concern with mathematical notation or systems for phonetic and phonemic transcription which make extensive semantic use of styles on a character by character basis.

2.4 Coincidence of Markup and Functions

Dealing with various functionalities on the markup level has the additional advantage that in most cases, text portions that need some particular attribute (or styling) are actually those text portions identified by markup. A paragraph may be in French, a citation may need a bidi embedding, a keyword may be in italics, a list number may be circled, and so on. This makes it very efficient to associate those attributes with markup.

However, where local or point-like functionality is needed, markup is *not* very efficient and its main benefit, easy manipulation of scope, is not required. On the contrary, the intrusion of markup in the middle of words can make search or sort operations more difficult. For these cases expressing the information as character codes is not only a viable, but often the preferred alternative, which needs to be considered in the design of markup languages.

2.5 Extensibility of Markup

Character encoding works with a range of integers used as character codes. This is extremely efficient, but has some limitations. Markup, on the other hand, is much more extensible. Using technologies such as XML Namespaces [Namespace] and their application in schema languages like [XML Schema], various vocabularies can be mixed.

2.6 Suitability of Characters in Markup

The suitability of a particular character for markup depends on its status in the Unicode Standard, the nature of its behavior in text and the availability of equivalent markup. Many format characters that are needed for advanced plain text are not suitable for use with markup. Section 3 gives a list and detailed descriptions. However, not all format characters are unsuitable for use with markup. Section 4 provides a list of format characters that are suitable for use with markup and gives some discussion about their use. In addition to format characters, the Unicode Standard also has compatibility characters, some of which may be replaceable by suitable markup. These characters are discussed in Section 5.

3. Characters not Suitable for use With Markup

There are characters which are unsuitable in the context of markup in XML/HTML and whose use is discouraged, because one or more of the following conditions apply:

- They are deprecated in the Unicode Standard.
- They are unsupportable without additional data.
- They are difficult to handle because they are stateful.
- They are better handled by markup.
- They are undesirable because of conflict with equivalent markup.

Section 3.1 provides a list of such characters. Sections 3.2 through 3.10 discuss in more detail the following points for the discouraged characters.

- Short description of semantics
- Reason for inclusion in Unicode
- Specific problems when used with markup
- Other areas where problems may occur (*e.g.* plain text)
- What kind of markup to use instead
- What to do if detected in a particular context

3.1 Table of Characters not Suitable for use With Markup

The following table contains the characters currently considered not suitable for use with markup in XML or HTML. (See however the note in the Introduction.) They may also be unsuitable for other markup or page layout languages. For determining possible conflict this report uses the markup available in HTML.

Table 3.1 Characters not suitable for use with markup

Codepoints	**Names/Description**	**Short Comment**
U+0340..U+0341	Clones of grave and ~~accent~~acute	Deprecated in Unicode
U+17A3, U+17D3	Obsolete characters for Khmer	Deprecated in Unicode
U+2028..U+2029	Line and paragraph separator	use <xhtml:br />, <xhtml:p></xhtml:p>, or equivalent
U+202A..U+202E	BIDI embedding controls (LRE, RLE, LRO, RLO, PDF)	Strongly discouraged in [HTML4.01]
U+206A..U+206B	Activate/Inhibit Symmetric swapping	Deprecated in Unicode
U+206C..U+206D	Activate/Inhibit Arabic form shaping	Deprecated in Unicode
U+206E..U+206F	Activate/Inhibit National digit shapes	Deprecated in Unicode
U+FFF9..U+FFFB	Interlinear annotation characters	Use ruby markup [Ruby]
U+FEFF	as ZWNBSP	Use U+2060 Word Joiner instead
	as Byte Order Mark	Use only at the start of a file, not as part of markup
U+FFFC	Object replacement character	Use markup, e.g. HTML <object> or HTML <img>
U+1D173..U+1D17A	Scoping for Musical Notation	Use an appropriate markup language
U+E0000..U+E007F	Language Tag code points	Use xhtml:lang or xml:lang

Except for Line and Paragraph Separator, or the Byte Order Mark, it is acceptable for browsers and similar user agents to ignore the presence of discouraged characters in HTML or XML. It is up to authoring tools to ensure proper conversion between these characters and equivalent markup where it exists.

3.2 Line and Paragraph Separator, U+2028..U+2029

Short description: The line and paragraph separator provide unambiguous means to denote hard line breaks and paragraph delimiters in plain text.

Reason for inclusion: These characters were introduced into the Unicode Standard to overcome the ambiguous and widely divergent use of control codes for this purpose. See *Section 5.8, Newline Guidelines,* in [Unicode].

Problems when used in markup: Including these characters in markup text does not work where it would duplicate the existing markup commands for delimiting paragraphs and lines.

Problems with other uses: The separator characters can also problematic when used in plain text, because legacy data is usually converted code point for code point into Unicode and all receivers of Unicode plain text have to effectively be able to interpret the existing use of control codes for this purpose. As a result, fewer Unicode implementations support these characters, than would be the case otherwise.

Replacement markup: In HTML, use <xhtml:br /> instead of U+2028 and surround paragraphs by <xhtml:p> and </xhtml:p> instead of separating them with U+2029.

What to do if detected: In a browser context, treat as white space, or ignore. When received in an editing context, replace the character by the corresponding markup.

3.3 Bidi Embedding Controls (LRE, RLE, LRO, RLO, PDF), U+202A..U+202E

Short description: The bidi embedding controls are required to supplement the Unicode Bidirectional Algorithm in plain text

Reason for inclusion: The Unicode Bidirectional algorithm unambiguously resolves the display direction for bidirectional text. It does so by assigning all characters directional categories and then resolving these in context. In a small number of circumstances this *implicit* method does not produce satisfactory results and embedding controls are needed to ensure that sender and receiver agree on the display direction for a given text. See Unicode Technical Report #9, The Bidirectional Algorithm [UAX 9].

Problems when used in markup: These characters duplicate available markup, which is better suited to handle the stateful nature of their effect.

Problems with other uses: The embedding controls introduce a state into the plain text, which must be maintained when editing or displaying the text. Processes that are modifying the text without being aware of this state may inadvertently affect the rendering of large portions of the text, for example by removing a PDF.

Replacement markup: The following table gives the replacement markup:

Unicode	**Equivalent markup**	**Comment**
RLO	<xhtml:bdo dir = "rtl">	
LRO	<xhtml:bdo dir = "ltr">	
PDF	</xhtml:bdo>	when used to terminate RLO or LRO only, otherwise ignore
RLE	dir = "rtl"	attribute on block or inline element
LRE	dir = "ltr"	attribute on block or inline element

For details on bidi markup, please see Section 8.2 of HTML [HMTL 4.0-8.2]. The text of HTML 4.0 gives this recommendation:

> ***Using HTML directionality markup with Unicode characters.*** *Authors and designers of authoring software should be aware that conflicts can arise if the `dir` attribute is used on inline elements (including `BDO`) concurrently with the corresponding [UNICODE] formatting characters. Preferably one or the other should be used exclusively. The markup method offers a better guarantee of document structural integrity and alleviates some problems when editing bidirectional HTML text with a simple text editor, but some software may be more apt at using the [UNICODE] characters. If both methods are used, great care should be exercised to insure proper nesting of markup and directional embedding or override, otherwise, rendering results are undefined.*

This document goes beyond HTML and recommends that *only* the markup should be used.

> **Note:** The interpretation of how to handle directionality markup for block level elements differs in different versions of [CSS].

What to do if detected: In a browser context, ignore. When received in an editing context, replace the characters by the appropriate markup.

3.4 Deprecated Formatting Characters, U+206A..U+206F

Short description: These characters are deprecated. They were originally intended to allow explicit activation of contextual shaping, numeric digit rendering and symmetric swapping.

Reason for inclusion: These characters were retained from draft versions of ISO 10646.

Problems when used in markup: The processing model for these characters is not supported in markup.

Problems with other uses: The Unicode Standard requires that symmetric swapping, contextual shaping, and alternate digit shapes are enabled by default and no longer supports inhibiting any of them by use of these character codes. The most likely effect of their occurrence in generated text would be that of a 'garbage' character.

Conversion for use with markup: Apply the appropriate conversion to bring the data stream in line with the Unicode text model for bidirectional text and cursively-connected scripts.

What to do if detected: When received by a browser as part of marked up text, they may be ignored. When received in an editing context, they may be removed, possibly with a warning. Alternatively, an appropriate conversion from the legacy text model may be provided. This will most likely be limited to applications directly interfacing with and knowledgeable of the particular legacy implementation that inspired these characters.

3.5 Byte Order Mark, ZWNBSP, U+FEFF

Short description: U+FEFF has two functions. It is formally known as ZERO WIDTH NO-BREAK SPACE (ZWNBSP), and can act as a word joiner, but its primary use is as *byte order mark (BOM)*, to indicate in a file signature at the start of a file that a file is in a particular Unicode encoding form and of a particular byte order. Using U+FEFF as a word joiner in new data is deprecated as of [Unicode3.2] in favor of U+2060 WORD JOINER (WJ). The use as byte order mark remains unaffected.

Reason for inclusion: Originally included in Unicode for the sole purpose of indicating byte order or use in file signatures, the character acquired the ZWNBSP semantics as part of the merger between ISO/IEC 10646 and Unicode. When used as a byte order mark the character is placed at the beginning of a file. If a recipient views it as FEFF then the byte order between sender and receiver match. If the recipient views it as FFFE (a non-character code point) then the sender used opposite byte order from the recipient, and the recipient needs to invert the byte order or refuse to read the file. When used as a ZWNBSP the character is intended to prevent breaks between adjacent characters. This function is now provided by U+2060 WORD JOINER (WJ) making it unnecessary to insert U+FEFF in the middle of a file. For more information see Chapter 16 of [Unicode].

Problems when used in markup: Using U+FEFF as ZWNBSP makes it impossible to distinguish it from the case where a byte order mark was left in the middle of a file inadvertently due to incorrect splicing. U+FEFF can and in some cases (XML encoded in UTF-16) must be used at the start of a file containing markup, but as a signature, this is not part of actual markup or marked-up content. Some older versions of browsers and parsers may not correctly recognize U+FEFF at the start of a file encoded in UTF-8. For details of how U+FEFF participates in encoding detection of XML files, see Appendix F of [XML 1.0].

Problems with other uses: The use of byte order mark as ZWNBSP is also problematic when used in plain text, and has been deprecated for that purpose in favor of U+2060 WORD JOINER. The use of U+FEFF in file signatures to indicate byte order is the only recommended use of this character.

Replacement markup: None. In locations other than the beginning of a text file, U+FEFF can be removed or replaced by U+2060 in an editing environment.

What to do if detected: When received by a browser as part of marked-up text, treat depending on location. At the start of an external entity, treat as byte order mark (i.e. as part of the character encoding, not as part of the parsed character stream, see e.g. Section 4.3.3 of [XML 1.0]). Otherwise, assume it is older data using it as ZWNBSP. When receiving plain text in an editing environment, editors may take one or more of several actions: replace ZWNBSP in the middle of a file with WJ or issue a warning to the user.

3.6 Interlinear Annotation Characters, U+FFF9-U+FFFB

Short description: The interlinear annotation characters are used to delimit interlinear annotations in certain circumstances. They are intended to provide text anchors and delimiters for interlinear annotation for in-process use and are not intended for interchange.

Reason for inclusion: The interlinear annotation characters were included in Unicode only in order to reserve code points for very frequent application-internal use. The interlinear annotation characters are used to delimit interlinear annotations in contexts where other delimiters are not available, and where non-textual means exist to carry formatting information. Many text-processing applications store the text and the associated markup (or in some cases styling information) of a document in separate structures. The actual text is kept in a single linear structure; additional information is kept separately with pointers to the appropriate text positions. This is called out-of-band information. The overall implementation makes sure that these two structures are kept in sync. If the text contains interlinear annotations, it is extremely helpful for implementations to have delimiters in the text itself; even though delimiters are not otherwise used for style markup. With this method, and unlike the case of the object replacement character, all textual information can remain in the standard text stream, but any additional formatting information is kept separately. In addition, the Interlinear Annotation Anchor serves as a placeholder for formatting information for the whole annotation object, the same way a paragraph mark can be a placeholder to attach paragraph formatting information.

Problems when used in markup: Including interlinear annotation characters in marked-up text does not work because the additional formatting information (how to position the annotation,...) is not available.

Problems with other uses: The interlinear annotation characters are also problematic when used in plain text, and are not intended for that purpose. In particular, on older display systems that simply ignore or replace the Interlinear Annotation Characters, the meaning of the text may be changed.

Replacement markup: The markup to be used in place of the Interlinear Annotation Characters depends on the formatting and nature of the interlinear annotation in question. For ruby, please see [Ruby].

What to do if detected: When received by a browser as part of marked-up text, they may be ignored. When receiving plain text in an editing environment, editors may take one or more of several actions: remove U+FFF9 together with removing all characters between U+FFFA and following U+FFFB; ignore U+FFF9 and turn U+FFFA and U+FFFB into "[" and "]" respectively, or into similar characters; issue a warning to the user; or tentatively convert into appropriate ruby markup for further editing and formatting by the user.

3.7 Object Replacement Character, U+FFFC

Short description: The object replacement character is used to stand in place of an object (e.g. an image) included in a text.

Reason for inclusion: The object replacement character was included in Unicode only in order to reserve a codepoint for a very frequent application-internal use. Many text-processing applications store the text and the associated markup (or in some cases styling information) of a document in separate structures. The actual text is kept in a single linear structure; additional information is kept separately with pointers to the appropriate text positions. The overall implementation makes sure that these two structures are kept in sync. If the text contains objects such as images, it is extremely helpful for implementations to have a sentinel in the text itself; any additional information is kept separately.

Problems when used in markup: Including an object replacement character in markup text does not work because the additional information (what object to include,...) is not available.

Problems with other uses: The object replacement character is also problematic when used in plain text, because there is no way in plain text to provide the actual object information or a reference to it.

Replacement markup: The markup to be used in place of the Object Replacement Character depends on the object in question and the markup context it is used in. Typical cases are <xhtml:img src='...' />, <xhtml:object ...>, or <html:applet ...>. These constructs allow providing all additional information needed to identify and use the object in question.

What to do if detected: Browsers may ignore this character. When received in an editing context, if the actual object is accessible, editors may either replace the character by the appropriate markup for that object, or otherwise remove it, ideally providing a warning.

3.8 Musical Controls, U+1D173..U+1D17A

Short description: A series of characters for controlling scope in musical notation.

Reason for inclusion: These characters designate the start and end of common musical constructs. Full musical layout depends on additional information, for example pitch, that cannot be encoded using Unicode. However, many musical symbols may be depicted in isolation (and without assigning pitch) as part of a textual discussion of music. Plain text use of Unicode characters is primarily intended for this latter purpose. The scoping operators can be used to support limited renderings of beams, slurs, phrases, etc. in this context. However, in the context of markup languages, musical scoring calls for a dedicated markup language (analogous to MathML) which would be expected to contain markup for these constructs.

Problems when used in markup: These characters duplicate information that can in principle be expressed in markup.

Problems with other uses: Their special code range allows them to be easily filtered, but applications that do not expect them will treat them as garbage characters.

Replacement markup: Replace with equivalent markup if available.

What to do if detected: Browsers may ignore these characters. When received in an editing context, editors may remove or replace them by equivalent markup.

3.9 Language Tag Characters, U+E0000..U+E007F

Short description: A series of characters for expressing language tags, based on existing standards for language tags using the rules in Chapter 16 of [Unicode].

Reason for inclusion: These characters allow in-band language tagging in situations where full markup is not available, while allowing easy filtering by applications that do not support them. They were solely included for the benefit of those Internet protocols, such as ACAP, which require a standard mechanism for marking language in UTF-8 strings, and at the same time to avoid the use of other tagging schemes that relied on specific details of the encoding form used.

Problems when used in markup: These characters duplicate information that can be expressed in markup.

Problems with other uses: Their special code range allows them to be easily filtered, but applications that do not expect them will treat them as garbage characters.

Replacement markup: Replace with equivalent language markup. XML and XHTML have the xml:lang attribute. HTML has the lang attribute. These attributes follow different scoping rules than the tag characters, therefore this replacement will generally not be a simple 1:1 substitution.

What to do if detected: Browsers may ignore these characters. When received in an editing context, editors may remove or replace them by equivalent markup.

3.10 Other Characters Deprecated in Unicode

Short description: The Unicode Character Database [UnicodeData] lists all characters that have been deprecated in [Unicode]. This list may grow (slowly) over time. Deprecated characters remain valid characters forever, but their use is strongly discouraged. Deprecation of characters is applied only in exceptional circumstances. It is never the result of historical changes of a writing system: characters no longer in current, modern use are retained in Unicode, as they are needed for the representation of historical documents.

Reason for inclusion: Usually, characters that are deprecated were never needed, but were inadvertently added to the Unicode Standard, perhaps based on incomplete information available at the time of encoding.

Problems when used in markup: Except where noted elsewhere in this document, their presence in markup presents the same problems as in plain text, usually that of an unnecessary duplicate encoding.

Problems with other uses: Depends on the character and the reason for its deprecation. For more information see [Unicode].

Conversion for use with markup: For deprecated characters not discussed elsewhere in this document, see the relevant descriptions of those characters in [Unicode] for information on the recommended alternatives.

What to do if detected: Unless a specific recommendation is given elsewhere, deprecated characters are not ignored; where possible, in an editing environment, a preferred alternate encoding may be substituted.

4. Format Characters Suitable for Use with Markup

The following table contains format characters that do not exhibit the problems discussed at the start of Section 3. Despite their apparent relation to or similarity with characters in table 3.1, they are considered suitable for use with markup. It is not acceptable for user agents to ignore the characters in table 4.1. For a description of these characters see [Unicode].

Table 4.1: Some characters that affect text format but are suitable for use with markup

Code points	**Names/Description**	**Short Comment**
U+00A0	No-break Space	Line break control
U+00AD	Soft Hyphen	Line break control
U+034F	Combining Grapheme Joiner	Used in sorting
U+0600	Arabic Number Sign	Subtending mark
U+0601	Arabic Sign Sanah	Subtending mark
U+0602	Arabic Footnote Marker	Subtending mark
U+0603	Arabic Sign Safha	Subtending mark
U+06DD	Arabic End of Ayah	Enclosing mark
U+070F	Syriac Abbreviation Mark (SAM)	Supertending mark
U+0F0C	Tibetan Mark Delimiter Tsheg Bstar	Non-breaking form of 0F0B
U+115F..U+1160	Hangul Jamo Fillers	Filler
U+180B..U+180E	Mongolian Variation Selectors(FVS1..FVS3), Mongolian Vowel Separator	Required for Mongolian
U+200B	Zero-width Space	Line break control
U+200C..U+200D	Zero-width Join Controls (ZWJ and ZWNJ)	Required for a.o. Persian and many Indic scripts
U+200E..U+200F	Implicit Directional Marks (LRM and RLM)	LRM and RLM are allowed
U+2011	Non-breaking Hyphen	Line break control
U+202F	Narrow No-break Space	Line break control/Mongolian
U+2044	Fraction Slash	Or use markup (MathML)
U+2060	Word Joiner	Use for that purpose instead of U+FEFF ZWNBSP
U+2061..U+2064	Invisible Mathematical Operators	Mathematical use
U+2FF0..U+2FFB	Ideographic Character Description	Graphic characters (not controls)
U+303E	Ideographic Variation Indicator	Graphic character (not a control)
U+FF80	Halfwidth Hangul Filler	Filler, not generally required
FE00..FE0F	Variation Selectors	Modify graphic characters
E0100..E01DF	Variation Selectors	Modify graphic characters

The following subsections briefly discuss some of the characters from the above list, particularly those that affect more than their immediately adjacent neighbors. Please see the Unicode Standard [Unicode] for full details.

4.1 Subtending Marks

Subtending marks are needed to represent a common feature in the Arabic and Syriac scripts where a mark can be placed below a range of characters, for example below a sequence of digits, to indicate a year. The Syriac abbreviation mark is placed above a series of characters, making it technically a supertending mark, and the ARABIC END OF AYAH is an enclosing mark. In the character stream, a subtending mark precedes the affected characters. The end of affected range of characters is defined implicitly, usually by the first non-alphanumeric character.

Unlike subtending marks, the scope of combining enclosing marks, such as COMBINING ENCLOSING CIRCLE, is limited to the preceding default grapheme cluster. For details on grapheme clusters see Unicode Standard Annex #29: "Text Boundaries", [UAX 29] .

There is currently no existing markup that can represent the scoping and layout functions defined by these characters, so they cannot be substituted. It is unresolved to what degree intervening markup affects the scope of these marks.

4.2 Fraction Slash

The fraction slash is used between sequences of decimal digits to form fractions. Whether the resulting fraction has a horizontal or diagonal fraction line is unspecified. The fallback is to leave the digits unchanged and display a regular slash. In order to separate a digit from a following fraction, as in 1¾, the use of U+2009 THIN SPACE is recommended.

For better control of fractions the use of [MathML] is suggested where appropriate.

4.3 Variation Selectors

A variation selector is intended to cause a specific variant form (or range of variant forms) when applied to a base character. For a variation selector to have an effect it must immediately follow its base character. Only pre-determined combinations of selected base characters and specific variation selectors have a defined effect. All other combinations are ill-formed and are to be ignored. The list of standardized combinations is documented in the Unicode Character Database, see [Variants]. In addition to the 256 generic variation selectors, there are 3 Mongolian *free variation selectors*. They function in all other ways like variation selectors, except they only apply to base characters from the Mongolian script. Since Mongolian, like Arabic, has positional character shapes, the variations are limited to particular shaping contexts.

4.4 Ideographic Description Characters

Ideographic Description Characters are included in the Unicode Standard as a means to indicate the composition of ideographs from a combination of pieces (terms), where each piece or term is either a Unicode character or composed. Ordinarily the result would be a human readable description of a character, perhaps one for which a font is not available. However, at least some vendors are interested in automatic conversion of these sequences into single ideographs.

4.5 Invisible Mathematical Operators

These characters are needed to convey the intended meaning of a mathematical expression to an automated parser whenever two elements are simply written next to each other. See Unicode Technical Report #25: "Unicode Support for Mathematics" [UTR25] for more details.

4.6 Line Break Controls

Most of these characters prevent line breaks adjacent to them, but ZWSP and SHY provide invisible line break opportunities. The detailed function of these characters is described in Unicode Standard Annex #14: "Line Breaking Properties" [UAX14]. While high-end applications may be able to deduce line breaking opportunities automatically solely with the help of very generic markup or styling properties, the use of these characters currently provides the most reliable and straight-forward way to control line breaking and hyphenation. Note that [HTML4.01] uses U+00A0 NO-BREAK SPACE also as a "hard space" (i.e. a space with a fixed width), something that is not part of its character semantics in [Unicode].

U+2011 NON-BREAKING HYPHEN (NBHY) is used to encode a hyphen that does not provide a line break opportunity. In several languages, the sequence <SHY, NBHY> may be used to handle special line breaking behavior for explicit hyphens, see [UAX14].

4.7 Hangul Fillers

These should not be needed except for texts that need to have a fixed number of jamos per Korean syllable block. See the description of Korean Syllable Blocks in [Unicode].

5. Characters with Compatibility Mappings

The Unicode Standard provides compatibility mappings for a number of characters. Compatibility mappings indicate a relationship to another character, but the exact nature of the relationship varies. In some cases the relationship means "is based on" in some other cases it denotes a property. When plain text is marked up, it may make sense to map some of these characters to a combination of their compatibility equivalents and suitable markup. It is important to understand the nature of the distinctions between characters and their compatibility equivalents and the context in which these distinctions matter. It is never advisable to apply compatibility mappings indiscriminately. This section provides guidance on when and how to apply compatibility mappings in the case of importing text from non-XML (non-marked-up) sources. The section is organized by the "compatibility tag" associated with each compatibility mapping.

5.1 Overview

The following table gives an overview of the various compatibility characters, organized by "compatibility tag". The first column, *Tag value,* contains the value of the "compatibility tag" from the Unicode Character Database [UnicodeData]. Although these tags use "<" and ">", they do not appear as such in markup and should not be confused with XML tags. *Code range* indicates a further break down by code points. *Action* summarizes the recommended action to be taken whenever markup is first applied to non-XML text. Each entry indicates whether the characters can be substituted using the compatibility equivalent according to Normalization Form KC of [UAX 15], can be replaced by equivalent markup where available, or should be retained. For some cases, instead of or in addition to markup, style information [CSS] is needed. *Description and usage* provides additional information. Sections 5.3 through 5.6 provide additional information for some of these sets of compatibility characters including detailed recommended actions.

Table 5.1 Characters with compatibility mappings

Tag value	Code range	Action	Description and usage
<circled>	all	retain	Circled letters and digits used for list item markers, and in running text
<compat>	2002..200A	retain	Fixed width spaces
	2100..2101	retain	Variant letter forms that are used as symbols
	2105..2106	retain	Variant letter forms that are used as symbols
	2121, 213B	retain	For use as single code point in vertical layout
	2160..217F	retain, or use list item marker style, or normalize	For use as single code point in vertical layout, or as list item marker
	2474..249B	retain, or use list item marker style, or normalize	Parenthesized or dotted number used as list item marker
	249C..24B5	retain, or use list item marker style, or normalize	Parenthesized letters used as list item markers
	3131..318E	retain	Compatibility Hangul Jamo. These do not conjoin
	3200..3229	retain, or use list item marker style, or normalize	Parenthesized characters used as list item markers

	322A..3243	retain	Parenthesized characters used as symbols in vertical layout
	32C0..32CB	retain	String used as single code point in vertical layout
	all other	retain	Maintain, semantic distinctions apply
<final>	all	normalize	Arabic Presentation forms
<font>	all	retain	Variant letter forms that are used as symbols
<fraction>	all	normalize	As long as fraction slash is supported!
<initial>	all	normalize	Arabic Presentation forms
<isolated>	all	normalize	Arabic Presentation forms
<medial>	all	normalize	Arabic Presentation forms
<narrow>	all	retain	Half-width characters
<noBreak>	all	retain	The compatibility mapping merely indicates the equivalent breaking character. The noBreak distinction must be preserved
<small>	all	retain	Precise usage unknown. Maintain, but do not generate
<square>	3300..3357	retain	Single display cell cluster containing multiple lines of kana for vertical layout
	3358..337D	retain	For use as single code point in vertical layout
	33E0..33FE	retain	For use as single code point in vertical layout
	all other	retain	Variant letter form used as symbol in vertical layout
<sub>	2080..208E	retain, or use markup	Subscript digits 0-9, as well as minus, plus, equal and parens
	all other	retain	Subscript characters, usually used as modifier letters in phonetic notation
<super>	00B2..00B3	retain, or use markup	Superscript digits 0-9, as well as minus, plus, equal and parens
	00B9		
	2070		
	2074..207E		
	all other	retain	Superscript characters, usually used as modifier letters in phonetic notation
<vertical>	all	normalize	East Asian Presentation forms
<wide>	all	retain	Full-width characters

Note: Some symbols used in vertical layout exist as single code points in legacy systems, but can also be composed on the fly by more advanced display engines. There are currently no style properties that could be used to express squared Kana clusters (*kumimoji*) or horizontal in vertical writing mode (*tate-chu-yoko*).

5.2 Generating New Text

Presentation forms and characters for which adequate representation exists as marked up text should never be entered into new data. Many of the characters with <font> tag are however suitable for new data, as long as they are used in the manner they are intended, that is as symbols, with definite semantic differentiation between the different forms. The largest set of these characters exists to carry essential semantic distinctions in mathematical notation, where the any loss of markup during text export would compromise the meaning of the text. Most of the characters with <super> and <sub> tag have been encoded for use in phonetic or phonemic transcriptions, where they act as ordinary letters and the use of style markup is therefore deemed inappropriate. However, it is inappropriate to use any of these classes of characters to create the appearance of styled text runs.

For example to write *hello,* one should use <i>hello</i> and not the sequence of Unicode characters U+210E, U+212F, U+2113, U+2113, U+2134. Conversely, to indicate *Planck's constant* one should use U+210E and not <i>h</i>.

When style is applied across entire words, sentences or paragraphs, the use of markup is preferred. When style is applied to individual letters, especially to letters inside a word, giving them a particular interpretation, the use of character codes is preferred. See also Section 5.6.

5.3 List Item Marker Characters

Short description: Characters with a <circled> tag or characters with <compat> tag and compatibility mapping to a parenthesized string.

Reason for inclusion: They are most frequently used for marking enumerated list items, but the characters with a <circled> tag often occur as dingbats or footnote markers in tables. The same characters are used in regular text when citing an item from a corresponding ordered list.

Problems when used in markup: These characters do not cause undue interaction with markup

Problems with other uses: None

Replacement markup: (in text use) these characters are often used in running text; sometimes, but not exclusively, in situations where the text is to be associated with an item from a nearby numbered list. Replacement markup may not be available, and the support for such markup is much more limited today than was anticipated when this document was first written.

(list item style) When generating marked up text these characters occur only internal to the user agent when list item styles are rendered. When marking up plain text data they could be converted to suitable list item styles, if such use can be properly inferred. The default recommendation is to retain the original character.

(characters with compatibility mappings of the form "(*n*)" or "*n*." or roman numerals) Unlike circled characters, these could be rendered by sequences of regular characters. Using a list item marker style would in theory allow the support of longer lists (the Unicode characters are limited to the set (1) to (20) and "1." to "20."). Using regular character sequences would also allow the use of fonts that match the text of the list.

What to do if detected: No action needs to be taken by browsers. When received in an editing context, substitution of a list item marker style may be appropriate. However, the same characters are very often used as dingbat-like symbols in tables, or may appear in general text, whether or not referring to an item from a list. Therefore the user must have the choice of whether to replace the character.

5.4 Fractions

Short description: Single character fractions such as ½ or ¼.

Reason for inclusion: Subsets of these occur in practically all legacy character sets.

Problems when used in markup: The character repertoire is limited to a few common fractions. When used with more general methods of generating fractions such as MathML [MathML] the usual problem of dual representation arises.

Problems with other uses: Other than normalization issues, these characters present no undue problems in plain text. Where fraction slash is supported, these can be expressed by substituting their compatibility mappings.

Replacement markup: MathML can represent fractions unambiguously. When using fraction slash, care must be taken such that values like 3½ do not turn into 31/2 (=15.5).

What to do if detected: No action needs to be taken by browsers or editors, except when converting plain text to MathML.

5.5 Squared or Horizontal

Short description: Characters that are symbols composed of groups of typically kana or Latin letters, digits plus slash for use in a single display cell in vertical display of text.

Reason for inclusion: Many existing character sets contain these as precomposed characters since for simple implementations this is the only way to support the common use of providing metric units and other abbreviations in a single character cell for vertical text layout.

Problems when used in markup: Proposed markup, including CSS styling, would be able express an unbounded set of these abbreviations, obviating the need of cataloguing these in the character encoding standard and making them more directly accessible to text based processing, for example searching.

Problems with other uses: The repertoire of these legacy characters is limited; many more combinations are in actual use than are accounted for in character sets. Pre-composed symbols do not make their text content available to search engines. They also require re-encoding for text laid out horizontally.

Replacement markup: None available.

What to do if detected: No action required. (Subject to change pending the outcome of current proposals.)

5.6 Superscripts and Subscripts

Short description: Mainly super and subscript digits, but also signs, parentheses and a large number of letters.

Reason for inclusion: Super and subscripted letters and digits are quite common in some forms of phonetic or phonemic transcriptions, where the use of styles is both awkward and prone to data integrity issues when exported to plain text. For super or subscripted letters in phonetic transcription in particular, a change from superscript of subscript to regular style would alter the meaning. Note that such use in transcription is not limited to letters: superscripted small digits are often used to indicate tone. When used for these purposes, these characters should be retained and markup should *not* be used.

A few super and subscript characters, primarily the digits, also occur in many legacy character sets, including Latin-1. Their use in pure plain text is common for databases, e.g. including metric units for part descriptions (viz. cm^2) or for (usually simplified) formulae as occur in titles of scientific publications.

When used in mathematical context (MathML) it is recommended to consistently use style markup for superscripts and subscripts. This is because mathematical layout allows not just individual symbols, but entire expressions to be superscripted or subscripted in a regular, nested manner.

Problems when used in markup: Mixing direct use of these characters with the use of style markup provides multiple representations of the same text, leading to potentially different treatment by search and display engines.

However, when super and sub-scripts are to reflect semantic distinctions, it is easier to work with these meanings encoded in text rather than markup, for example, in phonetic or phonemic transcription. Otherwise, they would require markup in the middle of words, and they may also be inadvertently changed to normal style text, when exporting to plain text. This applies to the majority of super and subscripted characters in Unicode. On the other hand, some user agent may support certain superscripted or subscripted characters only when used as marked up text for example, because of lack of font support for them.

Problems with other uses: none

Replacement markup: Unless used as letters, <xhtml:sup> and <xhtml:sub> or <mathml:msup> and <mathml:msub> may be used.

What to do if detected: Both representations (with or without style markup) should be equivalent for search purposes. Input methods for mathematical texts might enforce the use of styles. If superscript characters are encountered during display of mathematical formulae, it is recommended that they be displayed in a manner indistinguishable from that achieved by using regular characters with corresponding style markup..

5.7 Other Characters Marked <compat>

Short description: The <compat> label was given to a set of compatibility characters whose further classification was not settled at the time the standard was created. The largest components are list item marker characters.

Reason for inclusion: These characters occur in many legacy character sets.

Problems when used in markup: none. There usually is no equivalent markup.

Problems with other uses: none

Replacement markup: none.

What to do if detected: No action required.

6. Noncharacters

The Unicode Standard defines 66 non-character code points, or *noncharacters*. These are the last two positions on each of the 17 planes, in other words, all characters whose code points end in ...FFFE or ...FFFF, as well as the 32 code points from U+FDD0 to U+FDEF. Applications are free to use any of these code points internally but should never attempt to interchange them. In effect, noncharacters can be thought of as application-internal private-use code points.

7. White Space

This section presents common issues with white space characters in markup languages, mostly based on their difference in function as part of the structure of the markup source (syntactic white space) on the one hand and as part of the document content on the other hand.

The set of characters in the Unicode standard that have the property "White_Space" (see 'White Space' in the [UCD]) is quite large. It includes white space characters with different line breaking properties, different ligating properties, and different widths. It is appropriate to use these characters as part of markup content for their very specific purpose. It is preferable to place them in the markup source so that they are surrounded by ordinary characters rather than line breaks for example. The set of white space characters defined by typical markup language specifications is a subset of the characters that are considered white space by [Unicode] .

Each markup language defines the set of characters that it accepts as part of the markup syntax, this is usually a very small set. The XML [XML1.0] and [XML1.1] specifications define white space as a combination of one or more of the following characters: U+0020 SPACE, carriage return (U+000D), line feed (U+000A), or tab (U+0009). [HTML4.01] adds to these the form feed character (U+000C), but that character cannot be used in any XHTML version.

In addition, markup languages may use conventions for converting or removing some kinds of white space. XML processors replace some combinations of end-of-line characters by a single line feed character. [XML1.0] normalizes any two character sequences of (U+000D U+000A) or any U+000D not followed by U+000A to a single U+000A. [XML1.1] also normalizes NEL (U+0085) and U+2028 LINE SEPARATOR, but U+2029 PARAGRAPH SEPARATOR is not treated that way. Additional processing of white space before it is handled to an application also occurs for attribute values: line breaks are replaced by spaces, leading and trailing spaces are removed, and subsequent spaces are replaced by a single space.

In XML, white space is purely syntactic inside tags, for example, to separate the element name from attributes, and between elements in element content models (as they are typical for data-oriented applications). White space in element content models is used to lay out the markup source, using line breaks and indentation, to improve readability. The same use of white space is possible in many cases in mixed content (typical for text-oriented applications).

Because XML is used for a very wide range of applications, after the processing steps mentioned above it passes all white space to the application. Some XML applications such as [XHTML] may have their own white space processing rules when processing white space characters. Also, applications and software transforming XML (e.g. [XSLT]) have specific conventions of how they handle white space, and specific ways of how to control this behavior. To appropriately use white space characters, readers are advised to examine all involved standards and software.

If the characters U+2028 and U+2029 appear in text, they may be treated as zero-width characters without semantic meaning (see Section 3.2).

7.1 Converting Newline Functions to White Space

White space that is not purely syntactic, including control codes that define a newline function (see *Section 5.8, Newline Guidelines,* in [Unicode]), can be handled in three main ways.

1. For data-oriented applications, the textual content of elements is treated according to the needs of the data type in question. In many cases, processing by the application includes aspects similar to those of the processing of attribute values by the XML parser itself. For some types of data, in particular small data items, some applications may also simply prohibit the use of white space.
2. For running text in text-oriented applications, reflowing is used, i.e. the line breaks in the markup source are removed and the text is reflown into lines whose length is determined by the output medium and styling properties. In the context of Unicode, this reflowing process requires care; it is described in more detail below.
3. For preformatted text, such as program source code, line breaks must be preserved. Text-oriented applications usually contain special markup for preformatted text, e.g. <xhtml:pre>. XML itself defines an xml:space attribute that applications may use for a similar purpose.

When reflowing, line breaks and adjacent white space can be treated as space, removed, collapsed with adjacent control characters of the same type, or treated as zero-width space. Which choice is appropriate depends on the script of the surrounding text. The assumption is that line breaks and adjacent white space (in particular following white space, used for indentation) was added to make the markup source more readable, in particular to make each line fit on a line of a plain text editor. For scripts that use spaces, line breaks will have been inserted where there originally was a space; treating them as spaces therefore preserves the intended separation between words. For scripts which do not use spaces, such as Ideographic scripts or certain South East Asian scripts, such as Thai, line feeds should be removed, or replaced by U+200B zero width space. The choice of treatment can depend on the script value of the characters preceding and following the line feed character, assuming these characters belong to the same run of text.

> **Note:** The Unicode Standard [Unicode] specifies that the zero width space is considered a valid line-break point and that if two characters with a zero width space in between are placed on the same line they are placed with no space between them; and that if they are placed on two lines no additional glyph area is created at the line-break.

The details of reflowing are the responsibility of the various markup applications (e.g. [XHTML]). However, there is a tendency to move this functionality from markup applications to styling, so that it can be shared across applications.

Authors should be aware of the fact that the above script-specific treatment of line breaks when reflowing text is not yet available in all implementations (e.g. browsers). For scripts that do not use white space to separate words, it may therefore still be advisable to not split long lines.

Editing tools should try to support the user in the appropriate use of white space. Some white space characters cannot easily be entered via a keyboard, but some others, e.g. U+3000 Ideographic Space, can. Editing tools should try to make sure that only line breaks and white space that is accepted as syntactic white space by the relevant markup language are used to improve markup source readability.

While the styling possibilities provided by CSS and its implementations have not reached the level of professional typesetting systems, they offer a wide range of ways to control layout and spacing of text. A very simple example is text centering, which would have been done by inserting an appropriate number of spaces on each line in pure plain text.

8. Versioning

This report will be updated by the Unicode Technical Committee in cooperation with the W3C Internationalization Activity whenever the tables of characters in this document need to be updated as a result of the addition of characters to the Unicode Standard, as a result of a revised determination of the suitability of a given character for use with markup, or when additional background information or recommendations become available.

Each report carries a revision number, which may be used to refer to a specific version of the report. Older versions of the report will remain available. Each version of this report specifies the underlying version of the Unicode Standard.

For more information on the Unicode Standard and its versions, see:

- Versions of the Unicode Standard [UnicodeVersions]
- About the Unicode Character Database [UCD]

- Unicode Character Database [UnicodeData]

9. Conformance

In the context of the Unicode Standard, the material in this technical report is *informative.* However, other documents, particularly markup language specifications, may specify conformance including normative references to this document. Such references may have to be updated as a result of future updates to this report as discussed in Section 8, *Versioning.*

10. References

[Charmod]
Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf, Tex Texin, Eds., *Character Model for the World Wide Web 1.0: Fundamentals*, W3C Recommendation, 15-February-2005, <http://www.w3.org/TR/2005/REC-charmod-20050215/>.

~~**[Charmodnorm]**~~
~~François Yergeau, Martin J. Dürst, Richard Ishida, Addison Phillips, Misha Wolf, and Tex Texin, Eds., *Character Model for the World Wide Web 1.0: Normalization,* W3C Working Draft, 27-October-2005, <http://www.w3.org/TR/2005/WD-charmod-norm-20051027/>.~~

[CharReq]
Martin J. Dürst, *Requirements for String Identity and Character Indexing Definitions for the WWW*, W3C Working Draft, 10-July-1998, <http://www.w3.org/TR/WD-charreq>.

[CSS]
For information on cascading style sheet specifications, see <http://www.w3.org/Style/CSS/>.

[Feedback]
Reporting Errors and Requesting Information Online to the Unicode Consortium,<http://www.unicode.org/reporting.html>.

[HTML4.01]
Dave Raggett, Arnaud Le Hors, Ian Jacobs, Eds., *HTML 4.01 Specification*, W3C Recommendation, 18-Dec-1997 (revised on 24-Dec-1999), <http://www.w3.org/TR/1999/REC-html401-19991224/>.

[HTML 4.0 - 8.2]
Section 8.2 of [HTML4.0] *Specifying the direction of text and tables: the dir attribute* <http://www.w3.org/TR/1999/REC-html401-19991224/struct/dirlang.html#h-8.2>.

[MathML]
David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier, Eds., *Mathematical Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*, W3C Recommendation, 21-Oct-2003, <http://www.w3.org/TR/2003/REC-MathML2-20031021/>.

[Namespace]
Tim Bray, Dave Hollander, Andrew Layman, Eds., *Namespaces in XML (Second Edition)*, W3C Recommendation, 16-Aug-2006, <http://www.w3.org/TR/2006/REC-xml-names-20060816/>.

[Ruby]
Marcin Sawicki, Michel Suignard, Masayasu Ishikawa, Martin Dürst, Tex Texin, Eds., *Ruby Annotation*, W3C Recommendation, 31-May-2001, <http://www.w3.org/TR/2001/REC-ruby-20010531/>.

[UAX 9]
Mark Davis, *Unicode Standard Annex #9, The Bidirectional Algorithm*, <http://www.unicode.org/reports/tr9/>.

[UAX14]
Asmus Freytag,*Unicode Standard Annex #14, Line Breaking Properties* http://www.unicode.org/reports/tr14/

[UAX 15]
Mark Davis, Martin Dürst, *Unicode Standard Annex #15, Unicode Normalization Forms*, <http://www.unicode.org/reports/tr15/>.

[UAX 29]
Mark Davis,*Unicode Standard Annex #29*, *Text Boundaries*. <http://www.unicode.org/reports/tr29/>

[UCD]
~~About the~~ Unicode Standard Annex #44, Unicode Character Database, ~~<http://www.unicode.org/ucd/>.~~ <http://www.unicode.org/reports/tr44/>.

[Unicode]
~~The Unicode Consortium.*The Unicode Standard, Version 5.0* (Boston, MA, Addison-Wesley, 2007. ISBN~~

~~0-321-48091-0).~~
The Unicode Consortium.*The Unicode Standard* http://www.unicode.org/versions/latest/.

[Unicode3.2]
Unicode Standard Annex #28 Unicode 3.2, The Unicode Consortium, 2002.

~~**[Unicode40]**~~
~~*The Unicode Standard, Version 4.0*, *The Unicode Standard, Version 4.0,* (Reading, Massachusetts: Addison-Wesley Developers Press, 2003, ISBN 0-321-18578-1) or online as <http://www.unicode.org/versions/Unicode4.0.0/>.~~

~~**[Unicode50]**~~
~~The Unicode Consortium.*The Unicode Standard, Version 5.0* (Boston, MA, Addison-Wesley, 2007. ISBN 0-321-48091-0) or online as <http://www.unicode.org/versions/Unicode5.0.0/>~~

[UnicodeData]
Unicode Character Database, <http://www.unicode.org/Public/UNIDATA/UCD.html>.

[UnicodeVersions]
Versions of the Unicode Standard, <http://www.unicode.org/standard/versions/>.

[UTR25]
Asmus Freytag, Barbara Beeton, Murray Sargent, *Unicode Technical Report #25, Unicode Support for Mathematics, <http://www.unicode.org/reports/tr25/>*

[Variants]
Standardized Variants <http://www.unicode.org/Public/UNIDATA/StandardizedVariants.html>.

[XHTML]
Steven Pemberton, et al., Eds., *XHTML™1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4.0 in XML 1.0*, W3C Recommendation, 01-Aug-2002, <http://www.w3.org/TR/2002/REC-xhtml1-20020801/>.

[XML 1.0]
Tim Bray, Jean Paoli, Eve Maler, C. M. Sperberg-McQueen, François Yergeau, Eds., *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, 16-August-2006, <http://www.w3.org/TR/2006/REC-xml-20060816/>.

[XLST]
Michael Kay, Ed., *XSL Transformations (XSLT) Version 2.0*, W3C Recommendation, 23-January-2007, <http://www.w3.org/TR/2007/REC-xslt20-20070123/>

[XML 1.1]
Jean Paoli, Eve Maler, Tim Bray, C. M. Sperberg-McQueen, François Yergeau, John Cowan, Eds., *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C Recommendation 16-August-2006, <http://www.w3.org/TR/2006/REC-xml11-20060816/>.

[XML Schema]
Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, Eds., *XML Schema Part 1: Structures Second Edition*, W3C Recommendation 28-October-2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/> .

11. Acknowledgements

Mark Davis and Hideki Hiura contributed to the early drafts. Yukka Korpela and Felix Sasaki provided input to the current document.

12. Change History (last changes first)

Changes from http://www.unicode.org/reports/tr20/tr20-8.html : Fixed one typo: "accent" to "acute" in Table 3.1. Updated references to the Unicode Standard in the Introduction and References section. [RI]

Changes from http://www.unicode.org/reports/tr20/tr20-7.html : Added entries for new characters in Unicode 5.0. Updated references to use new chapter/section numbers in Unicode 5.0. Updated the discussion of superscript and subscript characters, accounting for the differences between their use in phonetic or phonemic transcription and mathematics. Added Section 3.10 and 4.5, 4.6 and 4.7. Added a Section 7 on handling white space. Updated references to W3C publications (AF). More work on white space section; moved everything about BOM to one place (MJD)

Changes from http://www.unicode.org/reports/tr20/tr20-6.html : Added entries for new characters in Unicode 4.0. Separated out, and extended, the discussion of format characters suitable for markup. This resulted in a new section 2.6, moving section 3.2 to 4, and renumbering, as well as new sections 4.1, 4.2, 4.3, 4.4. Added a discussion on noncharacters in a new section 6. Updated reference from Unicode 3.1 and 3.2 to Unicode 4.0. Improved the layout an description of what is now table 5.1. Changed the recommended action in 5.6 to none. Updated the Unicode status section. Changed http://www.unicode.org/unicode/reports/ to http://www.unicode.org/reports throughout to reflect the preferred style of URL (older style URLs continue to be valid). Updated references to W3C publications. (AF/MJD)

Changes from http://www.unicode.org/reports/tr20/tr20-5.html : Updated reference from Unicode 3.0 to 3.1 and 3.2 where appropriate. Added sections 3.6 and 3.9. Minor wording fixes in sections 2.3, 3.1, 3.2, 3.6, 3.10, 4.3, 4.5 and 5. (AF/MJD)

Changes from http://www.unicode.org/reports/tr20/tr20-4.html : Added a note to the introduction to limit the scope. Reorganized section 3 and clarified the language. Renamed some sections and tables. Updated the document to prepare for publication as Unicode Technical Report and W3C Note (AF/MJD). Minor editorial changes to the text, added section 4.7, fixed some dates, plus a few typos. (AF)

Changes from http://www.unicode.org/reports/tr20/tr20-3.html : Minor editorial changes to the introduction, fixed some references, links, and dates, plus a few typos. (AF/MJD)

Changes from http://www.unicode.org/reports/tr20/tr20-2.html : Added sections 2.1-2.6 (MJD), sections 3.1-3.5, and 3.8, as well as sections 4.4-4.6 and 8 (AF). Edited text for publication as DRAFT Unicode Technical Report. (AF)

Changes from http://www.unicode.org/reports/tr20/tr20-1.html : Completed references, linked TOC. Various wording changes. Added W3C WD stylesheet, logo, copyright, status of this document. Streamlined authors' section. (MJD) Added material on compatibility characters. (AF)

Changes from the initial draft: Fixed the header. Fixed the numbering. Fixed the title. Put references to final version of data files based on naming conventions. Minor wording changes. Added proposed language on annotation characters to match example on FFFC. Posted for internal review by UTC and W3C. (AF)

13. Copyright

> The Unicode Consortium makes no expressed or implied warranty of any kind, and assumes no liability for errors or omissions. No liability is assumed for incidental and consequential damages in connection with or arising out of the use of the information or programs contained or accompanying this technical report.
>
> Unicode and the Unicode logo are trademarks of Unicode, Inc., and are registered in some jurisdictions.