ISO/IEC JTC1/SC2/WG2 **N 2208**

Unicode in XML and other Markup Languages

Interim draft for non-public committee review by W3C i18n-ig and UTC. *(The status sections are worded as they would appear as result of a status change following this review). To go to the latest public version, click here.*

REVIEWERS: Comments or questions marked like this in bold red. I have flagged all text has some issues, or needs further updates like this. ***Text like*** this ***is new text*** *(except for whole sections, which I have listed in the change history)*. ***Text like*** this ***contains as yet unimplemented suggestions or comments from the e-mail that will not be part of the final text.***

You may either mail me back an edited version of this file with your revisions marked so I can consolidate them, or send suggestions in plain text. (AF).

(to become) DRAFT Unicode Technical Report #20

W3C Working Draft XX-xxxx-2000

Revision (Unicode):
2.1d4
This version:
http://www.unicode.org/unicode/reports/tr20/tr20-2.html
http://www.w3.org/TR/1999/WD-unicode-xml-19990928
Latest version:
http://www.unicode.org/unicode/reports/tr20
http://www.w3.org/TR/unicode-xml
Previous version:
http://www.unicode.org/unicode/reports/tr20/tr20-1.html
(no previous public version for W3C)
Date (Unicode):
2000-03-09
Authors:
Martin Dürst (mduerst@w3.org) and Asmus Freytag (*asmus@unicode.org*)

Summary

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

Status of this document (Unicode Consortium)

This draft is published for review purposes. This draft has been considered by the Unicode Technical Committee and approved as draft for public review. At its next meeting, the Unicode Technical Committee may approve, reject, or further amend this document. This document is a joint Unicode - W3C document.

The content of technical reports must be understood in the context of the latest version of the Unicode Standard. See http://www.unicode.org/unicode/standard/versions/ for more information.

This document does not, at this time, imply any endorsement by the Consortium's staff or member organizations. Please mail comments to unicore@unicode.org.

Status of this document (W3C)

This is a W3C Working Draft worked on jointly by the W3C Internationalization Working Group/Interest Group (Members only) and the Unicode Technical Committee. For public discussion of this working draft, please use the mailing lists www-international@w3.org and unicode@unicode.org (please crosspost to both lists). In order to post to unicode@unicode.org, you must join the list by following the simple instructions on http://www.unicode.org/unicode/email.html. For internal discussions, please use the relevant mailing list (again with crossposting). Please send editorial comments to the authors.

The material in this draft now covers all affected characters in the Unicode Standard, Version 3.0. Some of the detailed recommendations need additional discussion, and the descriptions may need additional clarification in some instances. It is not exactly clear yet how this document will be related to other W3C specifications. One potential way to proceed is to work towards publishing this document as a Note, and to reference it, normatively or otherwise, from the Character Model [CharMod] document.

Publication as a Working Draft does not imply endorsement by the W3C membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite W3C Drafts as other than "work in progress". A list of current W3C working drafts can be found at http://www.w3.org/TR.

Table of Contents

1. Introduction

The Unicode Standard [Unicode] is 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. Currently in its third 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 are more and more making use of marked-up text. In many instances, markup provides the same, or essentially similar features to those provided by formatting characters in the Unicode Standard for use in plain text. 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.

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.

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 'html:' indicates that this element is taken from [XHTML]. This means that the examples containing the namespace prefix 'html:' are assumed to include a namespace declaration of xmlns:html="..."

Characters are denoted using the notation used in the Unicode Standard, i.e. U+ followed by their hexadecimal number such as "U+1234". In XML or HTML this would be expressed as "&#x1234;".

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
- Coincidence of semantic markup and functions
- Extensibility of markup
- Markup *vs.* Styling

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 markup, since markup handles these concerns efficiently. If both may be present in the same text, the question of priority is raised: Which of the settings has priority and in which case? 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.

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, *etc.*. This makes it very efficient to associate those attributes with markup.

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 [Namespace] various vocabularies can be mixed. This is important for functionality that depends on complex context.

2.6 Local Context

Where local, or point like context is required, 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.

Reviewers: Please pay attention to the proposed action requirements "What to do if.." below in each subsection.

3. List of Characters

The following table contains the characters currently considered **not suitable** for use with markup in XML/HTML, and may also be unsuitable for other markup or page layout languages. Each category is further discussed below.

Table 3.1 Characters not suitable for use with markup

Codepoints	**Names/Description**	**Short Comment**
U+2028 .. U+2029	Line and paragraph separator	use <html:br />, <html:p></html:p>, or equivalent
U+202A .. U+202E	BIDI embedding controls (LRE, RLE, LRO, RLO, PDF)	Strongly discouraged in [HTML 4.0];
U+206A .. U+206B	Activate/Inhibit Symmetric swapping	Deprecated in Unicode 3.0
U+206C .. U+206D	Activate/Inhibit Arabic form shaping	Deprecated in Unicode 3.0
U+206E .. U+206F	Activate/Inhibit National digit shapes	Deprecated in Unicode 3.0
U+FFF9 .. U+FFFB	Interlinear annotation characters	Use ruby markup [Ruby]
U+FFFC	Object replacement character	Use markup, e.g. HTML <object> or HTML <img>
U-000E0000 .. U-000E007F	Language Tag codepoints (not part of Unicode 3.0)	Use html:lang or xml:lang

For each of the characters categories the following sections discuss these points:

- 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 in place
- What to if detected (remove/ignore/replace/complain,...)

The following table contains characters that despite their apparent relation to characters in table 3.1 are currently considered **suitable** for use with markup.

Table 3.2: Some characters that affect text format but are suitable for use with markup

Code points	Names/Description	Short Comment
U+00A0	No-break space	In Latin-1
U+00AD	Soft Hyphen	In Latin-1
U+070C	Syriac Abbreviation Mark (SAM)	
U+0F0C	Tibetan tsheg mark	
U+180B..U+180E	Mongolian Variant Selectors(FVS1.. FVS3, MVS)	Required for Mongol
U+200C..U+200D	Zero-width Joiners (ZWJ and ZWNJ)	Required for Persian
U+200E..U+200F	Implicit directional marks (LRM and RLM)	LRM and RLM are all
U+2011	Non breaking Hyphen	
U+202F	Narrow No-break Space	
U+2FF0..U+2FFB	Ideographic description	graphic characters (
U+303E	Ideographic variation indicator	graphic character (n

3.1 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 Unicode Technical Report #13, Unicode Newline Guidelines [UTR13].

Problems when used in markup: Including these characters in markup text does not work because 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.

Replacement markup: Use <html:br /> instead of U+2028 and surround paragraphs by <html:p> and </html:p> instead of separating them with U+2029.

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

3.2 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 [UTR 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 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	<BDO dir = "RTL">	
LRO	<BDO dir = "LTR">	
PDF	</BDO>	when used to terminate RLO or LRO only
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.*

What to do if detected: In a proxy context context, ignore. In a browser context, treat as ... When received in an editing context, replace the character by the appropriate markup for that object.

3.3 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: In a proxy context or browser context ignore. When received in an editing context, remove, possibly with a warning. Alternatively, provide appropriate conversion from the legacy text model.

This will most likely be limited to applications directly interfacing with and knowledgeable of the particular legacy implementation that inspired these characters.

3.4 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 place holder 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 markup 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 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: In a proxy context or browser context, remove U+FFF9 and remove all characters between U+FFFA and following U+FFFB, alternatively, ignore U+FFF9 and turn U+FFFA and U+FFFB to "[" and "]" respectively. When received in an editing context, either remove in the same manner, maybe with a warning to the user, or convert into appropriate ruby markup for further editing and formatting by the user.

3.5 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 <html:img src'...' />, <html:object ...>, or <html:applet ...>. These constructs allow to provide all additional information needed to identify and use the object in question.

What to do if detected: In a proxy context context, ignore. In a browser context, treat as either a missing image, or a REPLACEMENT CHARACTER. When received in an editing context, if the actual object is accessible, replace the character by the appropriate markup for that object. Otherwise remove, ideally providing a warning.

3.6 Language Tag Characters

Short description: A series of characters from U-000E0000 .. U-000E007F that can be used to express language tags, based on existing standards for language tags.

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 specifically included for the benefit of .... and to avoid the use of other 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 don't expect them will treat them as garbage characters.

Replacement markup: Replace with equivalent language markup <html:lang>

What to do if detected: In a proxy context or browser context ignore/remove/illegal. When received in an editing context, remove and replace by equivalent markup.

4. 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 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 where these distinctions matter. It is never advisable to apply compatibility mappings indiscriminately. This section provides guidance on when and how to apply compatibility mappings. It is organized by the "compatibility tag" associated with each compatibility mapping.

4.1 Overview

The following table gives an overview of the various compatibility characters, organized by "compatibility tag". The first column contains the tag value of the "compatibility tag" from the Unicode database. Although these tags use "<" and ">", they should not be confused with XML tags. *Code range* indicates which code points the entry applies to. *Substitute* indicates whether the codes can be substituted using the compatibility equivalent according to Normalization Form KC of [UTR 15]. *Markup* indicates the available markup. For some cases, instead of or in addition to markup, style information [CSS2] is needed.

Table 4.1 Characters not suitable for use with markup

Tag value	Code range	Action	Description and usage
<circled>	all	retain	Circled letters and digits used for bullets
<compat>	2100-2101	retain	Variant letter forms that are used as symbols
<compat>	2105-2106	retain	Variant letter forms that are used as symbols
<compat>	2121	??	For use as single code point in vertical layout
<compat>	2160-2175	??	For use as single code point in vertical layout
<compat>	3131-318E	retain	Compatibility Hangul Jamo. These do not-conjoin
<compat>	2002-200A	retain	Fixed width spaces
<compat>	3200-3229	use bullet style	Parenthesized characters used as bullets
<compat>	322A-3243	??	Parenthesized characters used as symbols in vertical layout
<compat>	2474-249B	use bullet style	Parenthesized or dotted number used as bullet
<compat>	249C-24B5	use bullet style	Parenthesized letters used as bullets
<compat>	32C0-32CB	??	String used as single code point in vertical layout
<compat>	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	decompose	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
<no-break>	all	retain	The compatibility mapping is merely a way to indicate the equival that is not non-breaking. The distinction must be preserved
<small>	all	retain	Precise usage unknown. Maintain, but don't generate
<square>	3300-3357	??	Single display cell cluster containing multiple lines of kana for ver
<square>	3358-337D	??	For use as single code point in vertical layout
<square>	33E0-33FE	??	For use as single code point in vertical layout
<sqare>	all other	??	Variant letter form used as symbol in vertical layout
<super>	all	<sup>	Superscripted characters
<sub>	all	<sub>	Subscripted characters
<vertical>	all	normalize*	East Asian Presentation forms
<wide>	all	retain	Full-width characters

Notes:

*) use normalization form KC for these particular characters.

Some symbols used in vertical layout may also be accessible via bullet style(s).

At the time of this writing it was not known what the appropriate markup would be for squared kana clusters or horizontal in vertical symbols.

Is there markup/style that can be substituted?? I have this note: Markup for quared kana clusters: There is some proposal for styling, not markup. Michel Suignard should be most up to date on this.

4.2 Generating characters

Presentation forms and characters for which adequate representation exists as marked up text should never be generated for new data. Many of the characters with <font> tag are 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. They should not be used to create styled text, but styled text should not be used to carry the essential semantic distinction needed for example for mathematics.

4.3 Bullets

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 enumerated bullets, but the characters with a <circled> tag often occur as dingbats or footnote markers in tables.

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

Problems with other uses: None

Replacement markup: (bullet style) When generating marked up text these characters occur only internal to the user agent as bullet styles are rendered. When marking up plain text data they could be converted to suitable bullet styles, if such use can be properly inferred.

Compatibility mappings of the form (*n*) or (*n*.) can be kept as single characters, or replaced by bullet styles. A conversion to bullet styles allows a simple extension of the set to arbitrary numbers. This is in contrast to circled characters: Very few browsers can properly generate arbitrary circled numbers, therefore conversion to bullet styles does not easily allow an extension of the set of accessible circled numbers.

What to do if detected: In a proxy context or browser context no action needs to be taken. When received in an editing context, substitution of a bullet style may be appropriate. However, the same characters are very often used as dingbat-like symbols in tables, so the user should have the choice of whether to replace.

4.4 Fractions

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

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

Problems when used in markup: The 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.

What to do if detected: In a proxy context or browser context...... When received in an editing context,.... .

4.5 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 / styling ? can 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: unknown at this time

What to do if detected: In a proxy context or browser context...... When received in an editing context,.... .

4.6 Super and subscripts

Short description:

Reason for inclusion: Super and subscript characters 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 (usually simplified) formulae as occur in titles of scientific publications.

Problems when used in markup: Using these characters directly in markup provides an alternate representation compared to marked up text, leading to different treatment by search engines.

Problems with other uses: none

Replacement markup:<xhtml:sup> and <xhtml:sub>

What to do if detected: In a proxy context or browser context...... When received in an editing context, substitute the corresponding markup.

4.7 [Template]

Short description:

Reason for inclusion:

Problems when used in markup:

Problems with other uses:

Replacement markup:

What to do if detected: In a proxy context or browser context...... When received in an editing context,.... .

5. Versioning

This technical report covers all relevant characters in the Unicode Standard, Version 3.0.

As the Unicode standard is updated and new characters get added, new characters that are not suitable for markup may also be added. However, the Unicode Technical Committee only introduces such characters where there is a strong requirement by industry or users of a script. As markup becomes more prevalent, the need for such characters may diminish.

This report will be updated by the Unicode Technical Committee in cooperation with the W3C i18n WG whenever the tables of characters need to be updated either as result of such changes in Unicode, as result of a revised determination of the suitability of a given character for use with markup, or when additional background information or recommendations will 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 will also specify the underlying version of the Unicode Standard.

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

- *Versions of the Unicode Standard (http://www.unicode.org/unicode/standard/versions)*
- *Unicode Character Database Format (ftp://ftp.unicode.org/Public/UNIDATA/UnicodeData.html)*
- *Unicode Character Database (ftp://ftp.unicode.org/Public/Public/UNIDATA/UnicodeData.txt)*

6. 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.

7. References

[Charmod]
Martin J. Dürst, *Character Model for the World Wide Web*, W3C Working Draft 25-Feb-1999, <http://www.w3.org/TR/WD-charmod>.

[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>.

[CSS2]
Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, Eds., *Cascading Style Sheets, level 2* (CSS2 Specification), W3C Recommendation 12-May-1998, <http://www.w3.org/TR/REC-CSS2>.

[HTML 4.0]
Dave Raggett, Arnaud Le Hors, Ian Jacobs, Eds., *HTML 4.0 Specification*, W3C Recommendation 18-Dec-1997 (revised on 24-Apr-1998), <http://www.w3.org/TR/REC-html40/>.

[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/html401/struct/dirlang.html#h-8.2>

[MathML]
Patrick Ion, Robert Miner, Eds., *Mathematical Markup Language (MathML™) 1.01 Specification* <http://www.w3.org/TR/REC-MathML/>

[Namespace]
Tim Bray, Dave Hollander, Andrew Layman, *Namespaces in XML*, W3C Recommendation 14-Jan-1999, <http://www.w3.org/TR/REC-xml-names/>.

[Ruby]

Marcin Sawicki, Michel Suignard, Masayasu Ishikawa, Martin Dürst, Eds., *Ruby*, W3C Working Draft 24-Sept-1999, <http://www.w3.org/TR/1999/WD-ruby-19990924/>.

[Unicode]
The Unicode Standard, *Version 3.0*, Addison Wesley, Reading MA, 2000, ISBN: 0-201-61633-5.

[UTR 9]
Mark Davis, *Unicode Technical Report #9, The Bidirectional Algorithm*, <http://www.unicode.org/unicode/reports/tr9/>.

[UTR 13]
Mark Davis, *Unicode Technical Report #13, Unicode Newline Guidelines*, <http://www.unicode.org/unicode/reports/tr13/>.

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

[XHTML]
XHTML™ 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4.0 in XML 1.0, <http://www.w3.org/1999/xhtml>.

[XML 1.0]
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eds., *Extensible Markup Language (XML) 1.0*, W3C Recommendation 10-February-1998, <http://www.w3.org/TR/REC-xml>.

8. Acknowledgements

Mark Davis (mark.davis@us.ibm.com), and Hideki Hiura (hideki.hiura@eng.sun.com) contributed to the early drafts.

9. Change History (last changes first)

Changes from *http://www.unicode.org/unicode/reports/tr20/tr20-2.html*: Added sections 2.1-2.6 (MJD), sections 3.1-3.3, and 3.6, 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/unicode/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)

10. Copyright

Unicode Home Page: *http://www.unicode.org*

Unicode Technical Reports: *http://www.unicode.org/unicode/reports/*