Ruby Annotation

W3C Recommendation 31 May 2001 (Markup errors corrected 25 June 2008)

This version:
http://www.w3.org/TR/2001/REC-ruby-20010531
(ZIP archive)
Latest version:
http://www.w3.org/TR/ruby
Previous version:
http://www.w3.org/TR/2001/PR-ruby-20010406

Editors:
Marcin SAWICKI (until 10 October, 1999)
Michel SUIGNARD, Microsoft
Masayasu ISHIKAWA (石川 雅康), W3C
Martin DÜRST, W3C
Tex TEXIN, Progress Software Corp.
(See Acknowledgements for additional contributors)

Abstract

"Ruby" are short runs of text alongside the base text, typically used in East Asian documents to indicate pronunciation or to provide a short annotation. This specification defines markup for ruby, in the form of an XHTML module [XHTMLMOD].

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this series of documents is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced as part of the W3C Internationalization Activity by the Internationalization Working Group (I18N WG, members only) with the help of the Internationalization Interest Group (I18N IG). Comments should be sent to the publicly archived mailing list www-i18n-comments@w3.org. Comments in languages other than English, in particular Japanese, are also welcome. Public discussion of this document takes place on the www-international@w3.org mailing list (see archive).

Due to its subject matter, and to make the examples more realistic, this document includes examples using a wide range of characters. Not all user agents may be able to display all characters. Depending on the user agent, changing the configuration can improve the situation. Also, great care has been taken to serve this document in various character encodings to cover a wide range of user agents and configurations.

Information related to this document can be found on the public ruby page (http://www.w3.org/International/O-HTML-ruby). This includes translations of this specification as well as potential errata. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

There have been no declarations regarding patents related to this specification within the Internationalization Working Group.

Contents

1. Introduction

This section is *informative*.

This document presents an overview of ruby annotation and defines the markup for it. Several examples are provided. However, this document does not specify any mechanisms for presentation or styling of ruby annotation; this is part of the respective style sheet languages.

This document is organized as follows:

Section 1.1 gives an overview of ruby annotation.

Section 1.2 gives an overview of the markup for ruby annotation.

Section 2 provides the normative definition of ruby markup.

Section 3 discusses typical rendering and styling of ruby text.

Section 4 provides conformance criteria.

1.1 What is ruby?

Ruby is the term used for a run of text that is associated with another run of text, referred to as the *base text*. Ruby text is used to provide a short annotation of the associated base text. It is most often used to provide a *reading* (pronunciation guide). Ruby annotations are used frequently in Japan in many kinds of publications, including books and magazines. Ruby is also used in China, especially in schoolbooks.

Ruby text is usually presented alongside the base text, using a smaller typeface. The name "ruby" in fact originated from the name of the 5.5pt font size in British printing, which is about half the 10pt font size commonly used for normal text. Figure 1.1 shows an example, with three ideographs (kanji) as base text, and six hiragana giving the reading (shinkansen - Japanese bullet train).

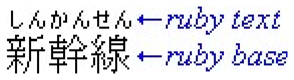

Figure 1.1: Ruby text giving the reading of each character of the base text.

East Asian typography has developed various features that do not appear in western typography. Most of these can be addressed appropriately with style sheet languages such as CSS or XSL. However, additional markup is required to define the association between base text and ruby text.

This specification defines such markup, designed to be usable with XHTML, so that ruby text is available on the Web without using special workarounds or graphics. Although this specification gives examples of actual rendering to make it easier for most readers to understand the markup, all such examples are informational only. This document does not specify any mechanisms for presentation or styling; this is part of the respective style sheet languages.

Sometimes more than one ruby text is associated with the same base text. A typical example is to indicate both meaning as well as reading for the same base text. In such cases, ruby texts may appear on both sides of the base text. Ruby text before the base text is often used to indicate reading; ruby text after the base text is often used to indicate meaning. Figure 1.2 shows an example of base text with two ruby texts, giving reading using hiragana and Latin letters.

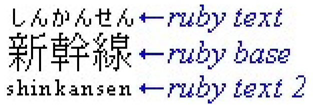

Figure 1.2: Two ruby texts applied to the same base text.

In addition, each ruby text may be associated with different, but overlapping, parts of the base text, such as in the following example:

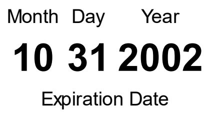

Figure 1.3: Base text with two ruby texts using different associations

In this example, the base text is the date "10 31 2002". One ruby text is the phrase "Expiration Date". This ruby text is associated with the entire base text. The other ruby text has 3 parts: "Month", "Day" and "Year". Each part is associated with a different part of the base text. "Month" is associated with "10", "Day" is associated with "31", and "Year" is associated with "2002".

1.2 Ruby markup overview

The markup defined in this specification is designed to cover all the above cases, namely markup for one or two ruby texts associated with the same base text and markup for associations of substrings of the ruby text(s) with components of the base text.

There are two variants of ruby markup, called *simple ruby markup* and *complex ruby markup*. Simple ruby markup associates a single ruby text with a run of base text. Simple ruby markup can also specify a fallback mechanism to allow display of ruby text by (older) browsers that do not know about ruby markup. Complex ruby markup can associate two ruby texts with one base text, and can define a more fine-grained association between components of the ruby text and the base text. However, complex ruby markup does not provide a fallback mechanism for browsers that do not understand ruby markup.

This section gives an overview of the markup for ruby defined in this specification. A full formal definition can be found in Section 2.

1.2.1 Simple ruby markup

In the simplest case, ruby markup defines a `ruby` element which contains one `rb` element for the base text and one `rt` element for the ruby text. This `ruby` element therefore creates an association between the base text and the ruby text, and is sufficient for most cases. Here is an example of simple ruby markup:

```
<ruby>
  <rb>WWW</rb>
  <rt>World Wide Web</rt>
</ruby>
```

Figure 1.4: Example of simple ruby markup

This may be rendered as follows:

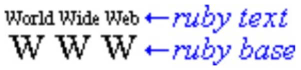

Figure 1.5: Example of rendering for simple ruby markup in Figure 1.4

***Note**: The name of this enclosing element, "<`ruby`>", should be interpreted to mean that its contents are associating ruby text with base text. It must not be misunderstood to mean that everything inside, including the base text, is ruby. The name of the enclosing element was chosen to compactly and clearly identify the function of the markup construct; the names for the other elements were chosen to keep the overall length short.*

1.2.2 Simple ruby markup with parentheses

Some user agents might not understand ruby markup, or may not be able to render ruby text appropriately. In either situation, it is generally preferable to render ruby text, so that information is not lost. A generally acceptable fallback is to place the ruby text immediately after the base text, and to enclose the ruby text in parentheses. The parentheses reduce the potential for confusing the ruby text with other text. (It should be noted that text in parentheses in Japanese typography is never called "ruby".)

For compatibility with older user agents that do not understand ruby markup and simply render the content of elements they do not understand, `rp` elements can be added to simple ruby markup to distinguish ruby text.

The element name `rp` stands for "ruby parenthesis". The `rp` elements and the parentheses (or other characters) inside them are provided as a fallback mechanism only. User agents that ignore unknown elements, but render their contents, will display the contents of each `rp` element. Therefore the `rp` element can be used to denote both the beginning and end of ruby text.

User agents that do know about ruby markup will recognize the `rp` element, and intentionally not display its contents. Instead, they will render the simple ruby markup in a more appropriate way.

The following example demonstrates the use of the `rp` element:

```
<ruby>
  <rb>WWW</rb>
  <rp>(</rp><rt>World Wide Web</rt><rp>)</rp>
</ruby>
```

Figure 1.6: Example of simple ruby markup including `rp` elements for fallback

User agents that either:

- do not know about ruby markup but render the contents of unknown elements, or
- cannot render the ruby text alongside the base text,

will render the above markup as:

WWW (World Wide Web)

Figure 1.7: Rendering of simple ruby markup using fallback parentheses

User agents that do know about ruby markup, and that have more sophisticated presentation styles for ruby text, will choose to not render the parentheses. For example, the markup of [figure 1.6]() can be rendered as shown in the next figure.

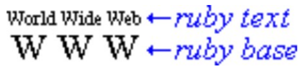

Figure 1.8: `rp` element ignored in favor of more sophisticated rendering

1.2.3 Complex ruby markup

Complex ruby markup is used to associate more than one ruby text with a base text, or to associate parts of ruby text with parts of base text.

Complex ruby markup provides for multiple `rb` and `rt` elements. This specification defines container elements that make the association between the individual elements clear. The ruby base container element, `rbc`, encloses `rb` elements. There can be one or two ruby text container elements, `rtc`, that enclose `rt` elements. This allows association of two ruby text containers with the same base text. With complex ruby markup it is also possible to associate parts of the base text with parts of a ruby text by using a number of `rb` elements, and a corresponding number of `rt` elements. In addition, the `rt` element may use the `rbspan` attribute to indicate that a single `rt` element spans (is associated with) multiple `rb` elements. This is similar to the `colspan` attribute of the `th` and `td` elements in tables ([HTML4], section 11.2.6).

Where and how each part of complex ruby markup is rendered is defined as part of the respective style sheet languages; see also section 3 for further information.

The following example shows all these features.

```
<ruby>
  <rbc>
    <rb>10</rb>
    <rb>31</rb>
    <rb>2002</rb>
  </rbc>
  <rtc>
    <rt>Month</rt>
    <rt>Day</rt>
    <rt>Year</rt>
  </rtc>
  <rtc>
    <rt rbspan="3">Expiration Date</rt>
  </rtc>
</ruby>
```

Figure 1.9: Complex ruby markup to associate two ruby texts with different parts of the same base text.

In this example, the first ruby text container encloses 3 components ("Month", "Day", "Year"). Each of these components is associated with a corresponding component in the base text ("10", "31", "2002"). The second ruby text container ("Expiration Date") consists of a single ruby text, and is associated with the entire base text ("10 31 2002"). It may be rendered as shown in figure 1.10.

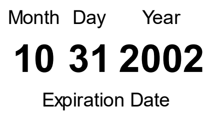

Figure 1.10: Rendering of the complex ruby markup in figure 1.9

The example shows that the association of ruby text with base text can be more or less granular as needed. For example, the ruby text can be associated with the entire base text in cases where:

- a more detailed relationship is unknown, or
- when the reading or annotation only applies to the whole unit and cannot be split into pieces.

More fine-grained associations can also be made when the relationships are known. For these situations, an improved rendering can therefore be provided. For example, a person's name can be decomposed into family name and given name, or a kanji compound or phrase can be decomposed into semantic subparts or individual characters. With either fine or course granularity, the spans of the ruby text can be set with the corresponding spacing in the base text, and better readability and a more balanced layout may be achieved.

The `rp` element is not available in the case of complex ruby markup. There are two reasons for this. First, the `rp` element is only a fallback mechanism, and it was considered that this is much more important for the more frequent simple case. Second, for the more complex cases, it is difficult to come up with a reasonable fallback display, and constructing markup for such cases can be even more difficult if not impossible.

1.2.4 Summary

In summary, the `ruby` element serves as a container for one of the following:

- a combination of `rb`, `rt` and possibly `rp` elements (**simple ruby markup**) for:
    - Association of a single ruby text with a single base text
    - Fallback in case the ruby markup is not understood.
- a combination of a single `rbc` and one or two `rtc` container elements (**complex ruby markup**) for:
    - Associating two ruby texts with the same base text
    - Defining more fine-grained associations between parts of a ruby text and parts of the base text.

2. Formal definition of ruby markup

This section is *normative*.

This section contains the formal syntax definition and the specification of the functionality of the ruby markup. Some familiarity with the XHTML Modularization framework, in particular the "*Modularization of XHTML*" [XHTMLMOD] specification, is assumed.

2.1 Abstract definition of ruby markup

The following is the abstract definition of the elements for ruby markup, which is consistent with the XHTML Modularization framework [XHTMLMOD]. Further definitions of XHTML abstract modules can be found in [XHTMLMOD].

Elements	Attributes	Minimal Content Model
ruby	Common	`(rb, (rt	(rp, rt, rp)))`
rbc	Common	`rb+`
rtc	Common	`rt+`
rb	Common	`(PCDATA | Inline - ruby)*`
rt	Common, rbspan (CDATA)	`(PCDATA | Inline - ruby)*`
rp	Common	`PCDATA*`

The maximal content model for the `ruby` element is as follows:

```
((rb, (rt | (rp, rt, rp))) | (rbc, rtc, rtc?))
```

The minimal content model for the `ruby` element corresponds to simple ruby markup. The `(rbc, rtc, rtc?)` alternative of the maximal content model for the `ruby` element corresponds to complex ruby markup.

An implementation of this abstract definition as an XHTML DTD module can be found in Appendix A. An XML Schema [XMLSchema] implementation is being worked on (see [ModSchema]).

2.2 The `ruby` element

The `ruby` element is an inline (or text-level) element that serves as an overall container. It contains either the `rb`, `rt` and optional `rp` elements (simple ruby markup) or the `rbc` and rtc elements (complex ruby markup).

In the case of simple ruby markup, the ruby element contains either an rb element followed by an rt element, or a sequence of an rb element, an rp element, an rt element and another rp element. The content of the rt element is taken as ruby text and associated with the content of the rb element as the base text. The content of the rp elements, if present, is ignored.

In the case of complex ruby markup, the ruby element contains an rbc element followed by one or two rtc elements. The content of the subelements of each rtc element is taken as ruby text and associated with the content of the subelements of the rbc element as the base text.

The ruby element has common attributes only. Examples of common attributes include: id, class or xml:lang. Common attributes depend on the markup language with which ruby markup is used. In the case of [XHTML 1.1], these are defined in XHTML Modularization, Section 5.1 [XHTMLMOD].

2.3 The rbc element

The rbc (ruby base container) element serves as the container for rb elements in the case of complex ruby markup. Only one rbc element may appear inside a ruby element.

The rbc element has common attributes only.

2.4 The rtc element

The rtc (ruby text container) element serves as the container for rt elements in the case of complex ruby markup. One or two rtc elements may appear inside a ruby element to associate ruby texts with a single base text, represented by an rbc element. More than two rtc elements MUST NOT appear inside a ruby element.

The rtc element has common attributes only.

2.5 The rb element

The rb (ruby base) element serves to markup the base text. For simple ruby markup, only one rb element may appear. For complex ruby markup, multiple rb elements may appear inside an rbc element. Each rb element is associated with a corresponding rt element, for fine-grained control of ruby presentation.

The rb element may contain inline elements or character data as its content, but the ruby element is not allowed as its descendant element.

The rb element has common attributes only.

2.6 The rt element

The rt element is the markup for ruby text. For simple ruby markup, only one rt element may appear. For complex ruby markup, multiple rt elements may appear inside an rtc element, and each `rt` element contains the ruby text for the relevant base text, represented by the corresponding `rb` element.

The `rt` element may contain inline elements or character data as its content, but the `ruby` element is not allowed as its descendant element.

The `rt` element has common attributes and the `rbspan` attribute. In complex ruby markup, the `rbspan` attribute allows an `rt` element to span multiple `rb` elements. The value shall be an integer value greater than zero ("0"). The default value of this attribute is one ("1"). The `rbspan` attribute should not be used in simple ruby markup, and user agents should ignore the `rbspan` attribute when it appears in simple ruby markup.

2.7 The `rp` element

The `rp` element can be used in the case of simple ruby markup to specify characters that can denote the beginning and end of ruby text when user agents do not have other ways to present ruby text distinctively from the base text. Parentheses (or similar characters) can provide an acceptable fallback. In this situation, ruby text will only degrade to be rendered inline and enclosed in the fallback parentheses. This is the least inappropriate rendering under the condition that only inline rendering is available. The `rp` element cannot be used with complex ruby markup.

The `rp` element has common attributes only.

Using parentheses for the fallback may lead to confusion between runs of text intended to be ruby text and other runs that happen to be enclosed within parentheses. The document or style sheet author should be aware of the potential for that confusion and is advised to choose an unambiguous delimiter for the fallback.

3. Rendering and styling considerations

This section is *informative*.

This section discusses various aspects of rendering and styling in the context of ruby markup as defined in this document. However, this document does not specify any mechanisms for presentation/styling; this is left to the respective style sheet languages. Formatting properties for styling ruby are under development for CSS and XSL. See for example "*CSS3 module: Ruby*" [CSS3-RUBY] (*work in progress*) for more details.

Details of ruby formatting in a Japanese print context can be found in JIS-X-4051 [JIS4051].

3.1 Ruby on the Web vs. traditional typographic usage

The term "ruby" in Japanese is only used for text visually rendered alongside the base text. Considerations for such cases are given in section 3.2 (font size), section 3.3 (positioning), and section 3.4 (presentation of ruby markup). This kind of presentation should be used wherever possible. However, introducing ruby to the Web may lead to some phenomena and problems that are not present in traditional typography. Structural markup for ruby, as defined in this specification, cannot guarantee that ruby text will always be rendered alongside the base text. There are a very wide variety of current and future output devices for documents marked up with XHTML. The following are possible scenarios and reasons for different rendering:

- On non-visual user agents such as voice browsers and braille user agents, only sequential rendering is possible. See section 3.5 for more consideration on non-visual rendering.
- On display devices with low resolution, displaying ruby text at the usual size may not be feasible. Fallbacks may be used. See section 3.6 for additional details.
- For educational purposes, it may in some cases be interesting to hide the ruby text and make it available as a pop-up. This is impossible on paper, but easily possible on a dynamic display device.

3.2 Font size of ruby text

In typical usage, the font size of ruby text is normally about half the font size of the base text. In fact, the name "ruby" originated from the name of the 5.5pt font size in British printing, which is about half the 10pt font size commonly used for normal text.

3.3 Positioning of ruby text

There are several positions where the ruby text can appear relative to its base text. Because East Asian text may be rendered vertically as well as horizontally, the terms "before" and "after" are used here rather than "above" and "below" or "right side" and "left side". The words "before" and "after" should be understood as "before"/"after" the line containing the base text. The correspondence is shown in the following table:

terminology	**Horizontal Layout** (left-to-right, top-to-bottom)	**Vertical Layout** (top-to-bottom, right-to-left)
before	above	right-side
after	below	left-side

Ruby texts are most frequently placed before the base text (see figure 1.1 and figure 3.2). Sometimes, especially in horizontal educational documents, ruby text may appear after the base text, i.e. below (see figure 3.1). In Chinese, it is rather common that Pinyin ruby text appears after the base text. Ruby text may also appear after the base text in vertical layout (see figure 3.3). In all these cases, the writing direction of the ruby text is the same as that of its base text, that is vertical if the base text is vertical, and horizontal if the base text is horizontal.

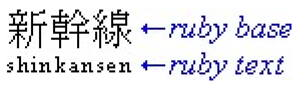

Figure 3.1: Ruby text (Latin letters) after/below the base text (Japanese ideographs)

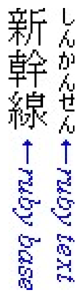

Figure 3.2: Ruby text in vertical writing (before/to the right)

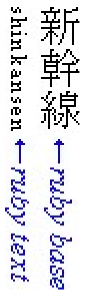

Figure 3.3: Ruby text in vertical writing (after/to the left).

In traditional Chinese texts, "Bopomofo" ruby text can appear along the right side of the base text even in horizontal layout.

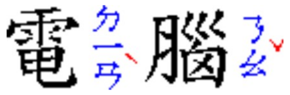

Figure 3.4: "Bopomofo" ruby text in traditional Chinese (ruby text shown in blue/red for clarity) in horizontal layout

Note that Bopomofo tone marks (in the above example shown in red for clarity) seem to appear in a separate column (along the right side of the Bopomofo ruby text) and therefore might be seen as "ruby on ruby". However, they are simply encoded as part of the ruby text. The details of this encoding are not addressed in this document.

3.4 Presentation of ruby markup

This specification does not prescribe how ruby markup will be displayed. Style sheets, in general, will be used to specify the exact behavior of ruby markup.

Note. *Although the rendering of the ruby texts should be controlled by style sheets, in case no style information is provided by the author or the user, it is recommended that visual user agents place the ruby text before the base text when only one ruby text is used. This is also the case for simple ruby. When there are two ruby texts, the first ruby text should be placed before the base text, and the second ruby text should be placed after the base text. A sample user agent default style sheet which describes this formatting will be provided by [CSS3-RUBY] or its successor document.*

For non-visual rendering, in the absence of style sheet information, it is recommended that both the base text and the ruby text(s) should be rendered, with an indication (e.g. different voice, different pitch, ...) of the status of each.

In order for style sheets to be able to apply styling, or for other mechanisms to render ruby text appropriately, it is very important to provide enough information on the function of each component. The following example illustrates the use of the class attribute to allow style sheets to define the exact presentation of the ruby text. The class "`reading`" is used for a ruby text that indicates reading. The class "`annotation`" is used to indicate ruby text that is used for annotation. The `xml:lang` attribute indicates the language of the text.

```
<ruby xml:lang="ja">
  <rbc>
    <rb>斎</rb>
    <rb>藤</rb>
    <rb>信</rb>
    <rb>男</rb>
  </rbc>
  <rtc class="reading">
    <rt>さい</rt>
    <rt>とう</rt>
    <rt>のぶ</rt>
    <rt>お</rt>
  </rtc>
  <rtc class="annotation">
    <rt rbspan="4" xml:lang="en">W3C Associate Chairman</rt>
  </rtc>
</ruby>
```

Figure 3.5: Ruby markup with `class` and `xml:lang` attributes.

Using a style sheet specifying horizontal text, rendering of the reading before the base text, and rendering of the annotation after the base text, the markup above could be rendered like this:

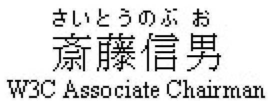

Figure 3.6: Horizontal rendering of two ruby texts associated with a single base text.

3.5 Considerations for non-visual rendering

Documents containing ruby markup may in some cases need to be rendered by non-visual user agents such as voice browsers and braille user agents. For such rendering scenarios, it is important to understand that:

- Depending on the user and the situation, different ways of rendering may be appropriate.
- Ruby text that represents reading may have to be treated differently from ruby text that contains other information.
- For appropriate non-visual rendering, it is important to indicate the function of each ruby text.
- There often are some differences between the reading indicated by the ruby text and the actual pronunciation.
- The reader may be interested in getting information about the (ideographic) base text.

Depending on a user's needs, the way a text should be read may vary from very quick and 'cursory' reading to very careful and detailed reading. This may lead to different ways of treating ruby text in non-visual rendering, from skipping ruby text in fast reading to detailed exploration of the ruby structure and the actual characters used in careful reading.

In the frequent case that ruby texts represent reading, rendering both the base text and the ruby text may produce annoying duplications. A speech synthesizer may be able to correctly pronounce the base text based on a large dictionary, or it may in other cases be able to select the right pronunciation based on the reading given by the ruby text.

Not all ruby texts represent pronunciations. Authors should distinguish ruby texts used for different purposes by using the `class` attribute. This is demonstrated above by using `class="reading"` for ruby text used to indicate reading.

Ruby text indicating reading may not produce the correct pronunciation even in cases where the script used at first glance seems perfectly phonetic. For example, Bopomofo is associated independently for each character of the base text; context-dependent sound or tone changes are not reflected. Similarly, in Japanese, spelling irregularities can occur, such as using "は" (hiragana ha) for the topic suffix pronounced "わ" (wa), or using vowels for indicating lengthening. For such cases, authors may want to supply the actual pronunciation with special markup designed for that purpose, or may rely on the aural rendering system being able to handle such cases correctly.

3.6 Alternatives to the `rp` element

If the author is not concerned about fallbacks for user agents that neither know about ruby markup nor support CSS2 [CSS2] or XSL [XSL] style sheets, then the `rp` elements are not needed.

Nevertheless, it is possible to parenthesize ruby text as a fallback if for example the device resolution is not appropriate for traditional ruby rendering. Using [CSS2], the parentheses can be generated using the 'content' property ([CSS2], section 12.2) with the :before and :after pseudo-elements ([CSS2], section 12.1), as for example in the following style fragment:

```
rt:before { content: "(" }
rt:after { content: ")" }
```

Figure 3.8: CSS2 style fragment to generate parentheses around an `rt` element

In the above example, parentheses will be automatically generated around the `rt` element. It is assumed that the above style rules are used together with style rules that position the ruby text inline. Generation of parentheses is straightforward with XSLT [XSLT].

4. Conformance Criteria

This section is *normative*.

Within the context of this specification, conformance can be claimed for markup, document types, module implementations, documents, generators, and interpreters. In most of these cases, two levels of conformance are available: simple conformance and full conformance. Simple conformance means that the conforming object supports the minimal content model of the ruby element in section 2.1, i.e. only simple ruby markup. Full conformance means that the conforming object supports the maximal content model of the ruby element in section 2.1, i.e. that both simple and complex ruby markup are supported.

Markup is *conforming simple ruby markup* if it contains one or more `ruby` elements and the content of all those elements (including their children) conforms to the minimal content model in section 2.1 (i.e. only simple ruby markup is allowed). Markup is *conforming full ruby markup* if it contains one or more `ruby` elements and the content of all those elements (including their children) conforms to the maximal content model in section 2.1 (i.e. both simple and complex ruby markup is allowed).

A document type is a *conforming simple ruby markup document type* if it integrates conforming simple ruby markup by adding the `ruby` element to the appropriate elements (such as inline elements) and by defining the necessary elements and attributes. A document type is a *conforming full ruby markup document type* if it integrates conforming full ruby markup by adding the `ruby` element to the appropriate elements (such as inline elements) and by defining the necessary elements and attributes.

A module implementation (e.g. with DTD or XML Schema technology) is a *conforming simple ruby module implementation* if it is designed to integrate simple ruby markup with other modules into document types as described above. A module implementation is a *conforming complex ruby module implementation* if it is designed to integrate full ruby markup with other modules into document types as described above. A module implementation is a *conforming full ruby module implementation* if it is designed to integrate either simple or full ruby markup with other modules into document types as described above (e.g. by providing a switch, or by providing two separate module implementations).

A document is a *conforming simple ruby markup document* if it contains conforming simple ruby markup and does not contain complex ruby markup or non-conforming ruby markup. A document is a *conforming full ruby markup document* if it contains conforming full ruby markup and does not contain non-conforming ruby markup.

A generator is a *conforming simple ruby markup generator* if it generates conforming simple ruby markup and does not generate complex ruby markup or non-conforming ruby markup. A generator is a *conforming full ruby markup generator* if it generates conforming full ruby markup and does not generate non-conforming ruby markup.

An interpreter is a *conforming simple ruby markup interpreter* if it rejects nonconforming simple ruby markup, accepts conforming simple ruby markup, and, where it interprets ruby markup, does so in accordance with this specification. An interpreter is a *conforming full ruby markup interpreter* if it rejects nonconforming ruby markup, accepts conforming full ruby markup, and, where it interprets ruby markup, does so in accordance with this specification. Examples of interpreters are server-side analysis or transformation tools and renderers.

For XHTML Modularization conformance, please see section 3 of [XHTMLMOD].

Appendices

A. Ruby module for XHTML

This appendix is *informative*.

The following is a link to the Ruby DTD module that is used in XHTML 1.1 [XHTML11].

- XHTML Ruby Module

B. Notes on design decisions

This appendix is *informative*. This appendix contains some notes on design decisions, based on questions and comments received during the Last Call review.

There were proposals to change e.g. <rbc><rb>...</rbc> to <rb><rbc>...</rb> (and similar for rt/rtc). This looks in some way more natural. However, in XML, the content of an element is either mixed content (both character data and elements, without sequence or occurrence restrictions) or element content (only elements, with restrictions). This means that it is impossible to say that <rb> contains either only <rbc> elements or only character data and inline elements.

There were various proposals for removing the `rp` element from the minimal content model. They were considered, but rejected for the following reasons:

- Recognition and removal of the `rp` elements by a receiver understanding ruby markup is extremely simple to implement; the burden on implementations is minimal. Both CSS and XSL provide easy mechanisms to remove the `rp` elements or to avoid displaying them.
- Displaying ruby text in parentheses is not desirable, because it can be confused with ordinary parenthesized text, but such a confusion is highly preferable to the confusion created should ruby text appear inline, as part of the actual text, without any distinguishing features.

It was suggested to change the names of the elements, in particular to change <ruby> to <gloss>. However, while ruby markup is indeed in some way similar to the markup that would be needed for glosses, it is not designed for that purpose.

C. Notes on backwards compatibility

This appendix is *informative*.

For historical reasons, some authoring tools might generate ruby markup without the start and end tags of the `rb` element, like:

```
<ruby>
  A
  <rp>(</rp><rt>aaa</rt><rp>)</rp>
</ruby>
```

rather than the following:

```
<ruby>
  <rb>A</rb>
  <rp>(</rp><rt>aaa</rt><rp>)</rp>
</ruby>
```

The former markup is not conforming to this specification, but user agents that care about compatibility with documents generated by such authoring tools may treat the former markup as if it were written like the latter.

D. Glossary

This appendix is *informative*.

Base text
: Run of text that has a ruby text associated with it.

Bopomofo
: 37 characters and 4 tone marks used as phonetics in Chinese, especially standard Mandarin.

Complex ruby markup
: In this specification: Ruby markup that allows association of two ruby texts with a single base text as well as fine-grained associations between parts of the ruby texts and the base text.

Group ruby
: In Japanese typography: Ruby text associated with more than one character of the base text.

Hiragana
: Japanese syllabic script, or character of that script. Rounded and cursive in appearance. Subset of the Japanese writing system, used together with kanji and katakana. In recent times, mostly used to write Japanese words when kanji are not available or appropriate, and word endings and particles.

Ideograph
: A character that is used to represent an idea, word, or word component, in contrast to a character from an alphabetic or syllabic script. The most well-known ideographic script is used (with some variation) in East Asia (China, Japan, Korea,...).

Kana
: Collective term for hiragana and katakana.

Katakana
: Japanese syllabic script, or character of that script. Angular in appearance. Subset of the Japanese writing system, used together with kanji and hiragana. In recent times, mainly used to write foreign words.

Kanji
: Japanese term for ideographs; ideographs used in Japanese. Subset of the Japanese writing system, used together with hiragana and katakana.

Monoruby
: In Japanese typography: Ruby associated with a single character of the base text.

Reading
: For ideographs: Technical term; indication of possible pronunciation. Different from

pronunciation in various respects: script used may not be fully phonetic; actual pronunciation is speaker-dependent; pronunciation may not be realized when reading a text silently. In Chinese or Korean, some ideographs have several readings. In Japanese, most ideographs have at least two readings, and some have a lot more. Readings also may depend on context.

Ruby text
: Run of text that appears in the immediate vicinity of another run of text (called "ruby base") and serves as an annotation or a pronunciation guide associated with the base.

Simple ruby markup
: In this specification: Ruby markup that associates a single ruby text with a single ruby base, optionally providing some delimiters such as parentheses for fallback.

E. Changes from Proposed Recommendation

This appendix is *informative*.

Changes from the Proposed Recommendation (http://www.w3.org/TR/2001/PR-ruby-20010406):

- Added conformance section
- Streamlined terms 'ruby text' and 'base text'
- Changed 'part of' to 'used by' in Appendix A ([XHTML11] does not contain any module definitions)
- Added link to [ModSchema]
- Fixed Appendix numbering and table of contents
- Fixed link in Section 3.3 (to fig. 1.1 rather than 1.3)
- Added Japanese titles for [JIS4051] and [JIS4052]
- Fixed various typos and minor grammatical mistakes

Acknowledgements

This section is *informative*.

Takao SUZUKI (鈴木 孝雄) and Chris WILSON have contributed to previous drafts as editors.

This specification would not have been possible without the help from the members of the W3C I18N WG, in particular Mark DAVIS and Hideki HIURA (樋浦 秀樹), and the members of the W3C I18N IG.

Additional contributors include Murray ALTHEIM, Laurie Anna EDLUND, Arye GITTELMA, Koji ISHII, Rick JELLIFFE, Eric LEVINE, Chris LILLEY, Charles MCCATHIENEVILE, Shigeki MORO (師 茂樹), Chris PRATLEY, Nobuo SAITO (斎藤 信男), Rahul SONNAD, Chris THRASHER.

The markup defined in this specification was coordinated with the ruby markup in [JIS4052], developed by WG 2 (Typesetting) of the Electronic Document Processing System Standardization Investigation and Research Committee of the Japanese

Standards Association. We would like to thank the members of WG 2, in particular Kohji SHIBANO (芝野 耕司, chair), and Masafumi YABE (家辺 勝文, liaison), for their collaboration. Technically, the markup for ruby in [JIS4052] differs from the markup in this specification in two points: First, there is an alternative form of markup not compatible with XML, based on special symbols, and second, the `rp` element is not permitted.

Valuable Last Call comments were also received from: The HTML WG, the CSS WG, the XSL WG, the WAI P&F WG, Steven PEMBERTON, Trevor HILL, Susan LESCH, and Frank Yung-Fong TANG. Akira UCHIDA (内田 明) provided feedback from a translator's viewpoint.

An earlier proposal for markup for ruby, using attributes, is described in [DUR97].

References

Normative References

[XHTML11]
"XHTML™ 1.1 - Module-based XHTML", W3C Recommendation
M. Altheim, S. McCarron, eds., 31 May 2001
Available at: http://www.w3.org/TR/2001/REC-xhtml11-20010531
The latest version is available at: http://www.w3.org/TR/xhtml11

[XHTMLMOD]
"Modularization of XHTML™", W3C Recommendation
M. Altheim *et al.*, eds., 10 April 2001
Available at: http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410
The latest version is available at: http://www.w3.org/TR/xhtml-modularization

[XML]
"Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation
T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, eds., 6 October 2000
Available at: http://www.w3.org/TR/2000/REC-xml-20001006
The latest version is available at: http://www.w3.org/TR/REC-xml

Informative References

[CSS2]
"Cascading Style Sheets, level 2 (CSS2) Specification", W3C Recommendation
B. Bos, H. W. Lie, C. Lilley and I. Jacobs, eds., 12 May 1998
Available at: http://www.w3.org/TR/1998/REC-CSS2-19980512
The latest version is available at: http://www.w3.org/TR/REC-CSS2

[CSS3-RUBY]
"CSS3 module: Ruby", W3C Working Draft
M. Suignard, ed., 16 February 2001
Available at: http://www.w3.org/TR/2001/WD-css3-ruby-20010216/
The latest version is available at: http://www.w3.org/TR/css3-ruby

[DUR97]
"Ruby in the Hypertext Markup Language", Internet Draft
Martin Dürst, 28 February 1997, *expired*
Available at: http://www.w3.org/International/draft-duerst-ruby-01

[HTML4]
: *"HTML 4.01 Specification", W3C Recommendation*
D. Raggett, A. Le Hors, I. Jacobs, eds., 24 December 1999
Available at: http://www.w3.org/TR/1999/REC-html401-19991224
The latest version is available at: http://www.w3.org/TR/html4

[JIS4051]
: *"Line composition rules for Japanese documents"* (日本語文書の行組版方法)
JIS X 4051-1995, Japanese Standards Association, 1995 (in Japanese)

[JIS4052]
: *"Exchange format for Japanese documents with composition markup"* (日本語文書の組版指定交換形式)
JIS X 4052:2000, Japanese Standards Association, 2000 (in Japanese)

[ModSchema]
: *"Modularization of XHTML™ in XML Schema", W3C Working Draft*
Daniel Austin and Shane McCarron, eds., 22 March 2001
Available at: http://www.w3.org/TR/2001/WD-xhtml-m12n-schema-20010322
The latest version is available at: http://www.w3.org/TR/xhtml-m12n-schema

[XMLSchema]
: *"XML Schema Part 1: Structures", W3C Recommendation*
H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, eds., 2 May 2001
Available at: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502
The latest version is available at: http://www.w3.org/TR/xmlschema-1
See also "*XML Schema Part 2: Datatypes*", available at: http://www.w3.org/TR/xmlschema-2

[XSL]
: *"Extensible Style Language (XSL)", W3C Candidate Recommendation*
S. Adler *et al.*, eds., 21 November 2000
Available at: http://www.w3.org/TR/2000/CR-xsl-20001121
The latest version is available at: http://www.w3.org/TR/xsl

[XSLT]
: *"XSL Transformations (XSLT) Version 1.0", W3C Recommendation*
J. Clark, ed., 16 November 1999
Available at: http://www.w3.org/TR/1999/REC-xslt-19991116
The latest version is available at: http://www.w3.org/TR/xslt