Japanese Subtitles on the Netflix Service

Authors (alphabetical)
Cyril Concolato: Senior Software Engineer, Cloud Media Systems
David Ronca: Director, Encoding Technologies
Rohit Puri: Manager, Cloud Media Systems
Yumi Deeter: Manager, Catalog QC

Introduction

This document is a technical description of the work that we have done to support JA subtitles, intended for discussion at the W3C. The primary purpose of the document is to describe the features that we consider essential for the success of IMSCvNext, and how TTML2 tools are used by Netflix to enable those features.

In 2014, we were working on the technical features for the planned September, 2015 launch of Netflix in Japan. At the time, we were mindful that other streaming services that were operating in the Japanese market had received significant criticism for providing a substandard subtitle experience. That knowledge, and our desire to maintain Netflix' very high standards of quality, led us to establish a set of challenging requirements:

1. Proper support for all of the "essential" Japanese subtitle features; those features that are expected in premium Japanese video services.
2. Subtitles must be delivered to clients separate from the video (i.e. no burned-in subtitles).
3. All subtitle sources must be delivered to Netflix in text format, in order to be future-proof.

Essential Japanese Subtitle Features

Feature Overview

When considering a high-quality, Japanese subtitle experience, we identified five essential Japanese subtitle features. These features are rubies, boutens, vertical text, tate-chu-yoko (horizontal numbers in vertical text), and slanted text. These essential features are described in the following sections.

Rubies

Rubies explain the meaning of unfamiliar, foreign, or slang words/phrases AND/OR convey pronunciation of kanji characters that are rare and/or unknown. Rubies can help provide cultural context to a translation which allows the viewer to enjoy the content with a deeper understanding. Common practice is to place the ruby above the base character for single-line subtitles, and for the first line of two-line subtitles. Rubies are placed below the base character if appearing on the second line of a two-line subtitle. **Rubies should never be placed between two lines as it is difficult to discern which base character they should be associated with.** Figure 1 shows a ruby example for the dialogue *“All he ever amounted to was chitlins.”*

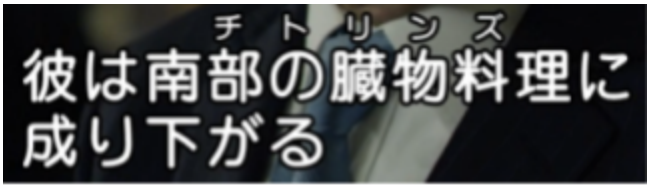


Figure 1: Rubies Example

The base text translates the word “chitlins”[1], while the ruby provides transliteration of the word “chitlins” so that viewers can more closely associate the keyword of the dialogue to the translation.

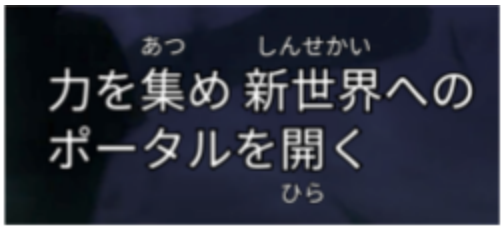


Figure 2: Proper ruby placement for a two-line subtitle

As mentioned above, rubies should never be placed in between two lines. Figure 2 shows the proper placement for rubies with two-line subtitles. In the unlikely event that a subtitle spans 3 lines, it is preferable to have the rubies on top of each line, except for the last line where they should be at the bottom.

Boutens

Boutens are dots placed above or below a word or phrase that act as literal points of emphasis, equivalent to the use of italics in English. Boutens can help express implied meanings which provide a richer and more dynamic translation. Figure 3 shows a bouten example for the dialogue*: “I need someone to talk to.”*

Figure 3: bouten example:

This subtitle has boutens above the word for “talk”. In the context of this scene, placing emphasis on this word allows the viewer to understand the implication that the speaker needs someone to provide him/her with privileged information.

Vertical Subtitles

Vertical subtitles are generally used to avoid overlap with on-screen text present in the video. This is the Japanese equivalent to putting subtitles at the top of the screen. This is illustrated in Figure 4.

Figure 4: Vertical subtitle overlap with credits

Tate-chu-yoko

In Japanese typography, vertical text often includes short runs of horizontal numbers or Latin text. This is referred to as tate-chu-yoko. Instead of stacking the characters vertically, half-width characters are placed side-by-side to enhance legibility and allow more characters to be placed on a single subtitle line. This is illustrated in the figure to the right, for the dialogue, "*It’s as if we are still 23 years old*". In this example, subtitle, the number “23” uses half-width numeric characters,
and employ the tate-chu-yoko functionality.

[1] Slang word used in “House of Cards”, also known as “Chitterlings”, southern US food usually made from the small intestines of a pig. Such word is unlikely to be known to the audience.

Slanted Text

Slanted text is used in similar fashion as italics/oblique text in other languages - for narration, off-screen dialogue, and forced narratives. One unique feature in Japanese subtitles however is that italics slant in different directions for horizontal vs. vertical subtitles; furthermore, the angle of the slant is not necessarily constant, but may vary. This is illustrated in Figure 6.

Figure 6: Example of horizontal and vertical slanted text.

LambdaCAP: The only option.

From the first subtitle asset, Netflix has always required text sources (vs. image). There are several reasons for this requirement. First, different clients have different subtitle capabilities, requiring us to be able to produce many variations of client assets from a single source. In addition, text subtitle sources are future-proof. That is, as new device capabilities emerge, we can apply those to our large back-catalog of subtitle assets. As an example, when displaying subtitles on an HDR device playing HDR content, it is useful to specify the luminance gain so that white text is not a max-white specular highlight. With text sources, we can easily go back and reprocess to produce subtitles for a client profile that supports #luminanceGain. If we had ingested image sources, on the other hand, it would have been difficult to add this sort of functionality to client assets.

With text sources as a "must-have" requirement, we reviewed the available options for Japanese, and LambdaCAP was chosen as the only workable model for JA subtitles. There were several reasons for this decision. From our analysis, we determined that LambdaCAP:

- is reasonably open, allowing us to develop our own tools and workflows.
- is currently the most common subtitle format that Japanese subtitle tools can support. This was a key driver of our decision because it meant that the established JA subtitle industry could produce subtitles for Netflix.
- is the most common archive format for existing Japanese subtitles. Another key driver, because supporting LambdaCAP meant that we could ingest existing assets without any transformation requirements.
- supports the essential Japanese features as described above.
- has been widely used in the industry to create image-based subtitle files for burn-in. Thus, it is well-tested.

While the LambdaCAP model was suitable for our JA launch, it is not a great long-term option. It is not an industry standard, and there are some ambiguities in the specification. Thus, while we chose LambdaCAP as an interim source format, we do not feel that it is suitable as a long-term subtitle. In addition, we chose to not use LambdaCAP as a client model for JA subtitles. That is, we ingest LambdaCAP assets and create derivative subtitle assets in TTML2 and WebVTT, but we do not deliver LambdaCAP to any Netflix clients.

Mapping of Japanese Features to TTML2

The following table summarizes the features of TTML2 that are considered by Netflix as essential for the support of Japanese subtitling. It also shows with usage statistics that most of these features are already used, sometimes significantly, in today's Japanese subtitles. The other features not yet used or used significantly are expected to be used more widely in the future[2]. The following sections provide details on each feature, in particular regarding the supported values.

Japanese Feature	TTML2 Feature	Percentage of subtitle events using the feature	Notes
Slanted Text	tts:fontShear	8.2%	n.a.
Rubies	tts:ruby	1.9% (including 0.2% in vertical text)	n.a.
	tts:rubyAlign	Used with its default value 'auto' when tts:ruby is used	Only 'auto'
	tts:rubyPosition	Used with its default value when tts:ruby is used	'outside' is most useful. 'auto' should take same behavior as 'outside'.
	tts:rubyReserve	Not ingested yet	Only 'auto'
Tate-chu-yoko	tts:textCombine	0.1%	n.a.
Boutens	tts:textEmphasis	0.1%	n.a.
Vertical Subtitles	tts:writingMode	5.9%	n.a.

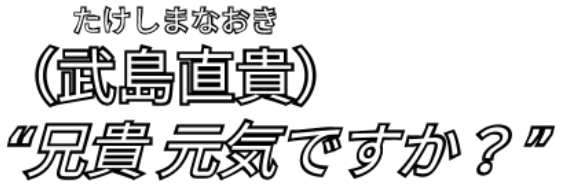
Ruby

tts:ruby

This styling attribute specifies structural aspects of ruby content. The range of values associated with tts:ruby maps to corresponding HTML markup elements, but for subtitling we think that nested rubies and the notions of baseContainer, textContainer and delimiters are not required.

[2] Other TTML2 JA features (such as tts:rubyOverhang) not listed in the table are not necessary for the support of Japanese features in the Netflix use-case.

Sample rendering



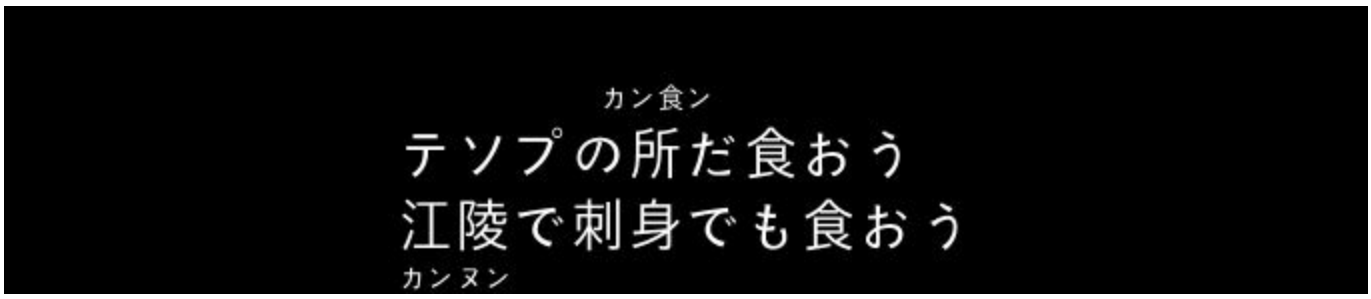
tts:rubyPosition

For the Japanese use-case, tts:rubyPosition="top" and tts:rubyPosition="bottom" are less than ideal because they do not provide for unanticipated word wrapping, where the second line of text should have rubies below. In general, the behavior tts:rubyPosition="auto" would be preferred. However, the behavior of 'auto' is only specified for exactly two-line events, which will not cover the use case of unanticipated wrapping of a two-line event. We believe that that the current behavior described for 'outside' is the correct model, and that 'auto' should have the same behavior as 'outside'.

TTML snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<tt xmlns="http://www.w3.org/ns/ttml" xmlns:tt="http://www.w3.org/ns/ttml"
xmlns:ttm="http://www.w3.org/ns/ttml#metadata" xmlns:ttp="http://www.w3.org/ns/ttml#parameter"
xmlns:tts="http://www.w3.org/ns/ttml#styling" ttp:tickRate="10000000" ttp:version="2" xml:lang="ja">
    <head>
        <styling>
            <initial tts:backgroundColor="transparent" tts:color="white" tts:fontSize="6.000vh"/>
            <style xml:id="style0" tts:textAlign="center"/>
            <style xml:id="style1" tts:textAlign="start"/>
            <style xml:id="style2" tts:ruby="container" tts:rubyPosition="auto"/>
            <style xml:id="style3" tts:ruby="base"/>
            <style xml:id="style4" tts:ruby="text"/>
            <style xml:id="style5" tts:ruby="text"/>
        </styling>
        <layout>
            <region xml:id="region0" tts:displayAlign="after"/>
        </layout>
    </head>
    <body xml:space="preserve">
        <div>
            <p xml:id="subtitle1" begin="18637368750t" end="18676157500t" region="region0"
style="style0"><span style="style1">テソプ<span style="style2"><span style="style3">の所だ</span><span
style="style4">カン食ン</span></span>食おう<br/><span style="style2"><span style="style3">江陵</span><span
style="style5">カンヌン</span></span>で刺身でも食おう</span></p>
        </div>
    </body>
</tt>
```

Sample rendering



tts:rubyAlign

Netflix's prefered value of tts:rubyAlign is "center".

TTML snippet

```
<?xml version="1.0" encoding="utf-8"?>
<tt xmlns="http://www.w3.org/ns/ttml" xmlns:tt="http://www.w3.org/ns/ttml"
xmlns:ttm="http://www.w3.org/ns/ttml#metadata" xmlns:ttp="http://www.w3.org/ns/ttml#parameter"
xmlns:tts="http://www.w3.org/ns/ttml#styling" ttp:tickRate="10000000" ttp:version="2" xml:lang="ja">
  <head>
    <styling>
      <initial tts:backgroundColor="transparent" tts:color="white" tts:fontSize="6.000vh"
tts:lineHeight="7.500vh" tts:opacity="1.000" tts:showBackground="whenActive" tts:writingMode="lrtb"
tts:rubyAlign="center"/>
      <style xml:id="style0" tts:textAlign="center"/>
      <style xml:id="style1" tts:textAlign="start"/>
      <style xml:id="style2" tts:ruby="container"/>
      <style xml:id="style3" tts:ruby="base"/>
      <style xml:id="style4" tts:ruby="text" tts:rubyPosition="auto"/>
    </styling>
    <layout>
      <region xml:id="region0" tts:displayAlign="after" tts:extent="80.000% 30.000%" tts:position="center
bottom 10vh" tts:showBackground="whenActive"/>
    </layout>
  </head>
  <body xml:space="preserve">
<div>
<p xml:id="subtitle1" begin="18637368750t" end="18676157500t" region="region0" style="style0"><span
style="style1">テソプ<span style="style2"><span style="style3">の所だ</span><span style="style4">カン食ン
</span></span>食おう</span></p>
<p xml:id="subtitle2" begin="18676991666t" end="18717031666t" region="region0" style="style0"><span
style="style1">テソプ<span style="style2"><span style="style3">の所だ</span><span style="style4">カン食ンカン食
ン</span></span>食おう</span></p>
</div>
</body>
</tt>
```

Sample rendering

The illustrations below were obtained from the above TTML snippet and they serve to describe the behavior in two cases, when the base text is wider than the ruby text and vice-versa. In all cases, the base text corresponds to 'の所だ' (3 Unicode characters) and the ruby annotation is at "center".

Case 1

In this case, the width of the ruby text is smaller than the width of the base text.

Case 2

In this case, the width of the ruby text is greater than the width of the base text. As illustrated below, when there is a single ruby text, because the initial value of tts:rubyOverhang is “allow”, the ruby text is allowed to overhang the surrounding text. Note that we have never encountered a use case when there are 2 adjacent large ruby texts.

tts:rubyReserve

The intent of this feature is to maintain temporal consistency in placement of the base text along the block progression direction as we move from subtitles with only base text to those with base text that is annotated with rubies.

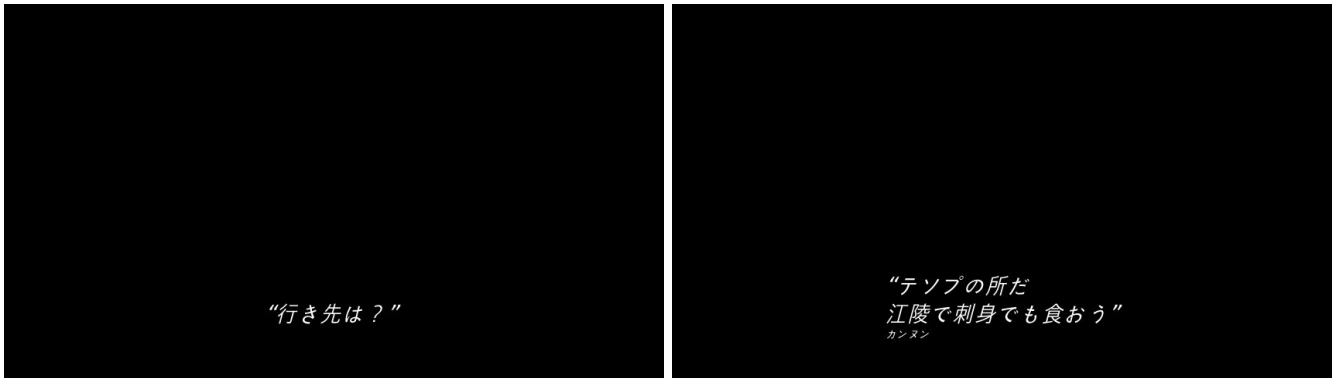
NOTE: This feature can also be used to preserve base text alignment across time when boutens are used.

TTML snippet

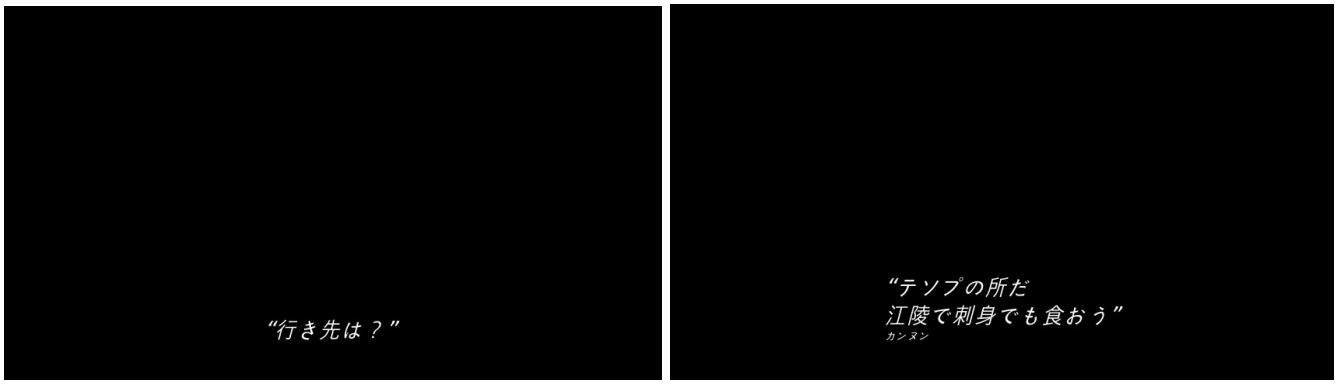
```
<?xml version="1.0" encoding="utf-8"?>
<tt xmlns="http://www.w3.org/ns/ttml" xmlns:tt="http://www.w3.org/ns/ttml"
xmlns:ttm="http://www.w3.org/ns/ttml#metadata" xmlns:ttp="http://www.w3.org/ns/ttml#parameter"
xmlns:tts="http://www.w3.org/ns/ttml#styling" ttp:tickRate="10000000" ttp:version="2" xml:lang="ja">
  <head>
    <styling>
<!-- set tts:rubyReserve in initial section -->
      <initial tts:backgroundColor="transparent" tts:color="white" tts:fontSize="6.000vh"
tts:lineHeight="7.500vh" tts:opacity="1.000" tts:showBackground="whenActive" tts:writingMode="lrtb"
tts:rubyReserve="auto"/>
      <style xml:id="style0" tts:fontShear="16.78842%" tts:textAlign="center"/>
      <style xml:id="style1" tts:fontShear="16.78842%" tts:textAlign="start"/>
      <style xml:id="style2" tts:fontShear="16.78842%" tts:ruby="container"/>
      <style xml:id="style3" tts:fontShear="16.78842%" tts:ruby="base"/>
      <style xml:id="style4" tts:fontShear="16.78842%" tts:ruby="text" tts:rubyPosition="auto"/>
    </styling>
    <layout>
```

```
      <region xml:id="region0" tts:displayAlign="after" tts:extent="80.000% 30.000%" tts:position="center
bottom 10vh" tts:showBackground="whenActive"/>
    </layout>
  </head>
  <body xml:space="preserve">
<div>
<!-- illustration of tts:fontShear -->
<p xml:id="subtitle1" begin="18623187916t" end="18635700416t" region="region0" style="style0"><span
style="style1">〝行き先は？〞</span></p>
<p xml:id="subtitle2" begin="18637368750t" end="18676157500t" region="region0" style="style0"><span
style="style1">〝テソプの所だ<br/><span style="style2"><span style="style3">江陵</span><span style="style4">カ
ンヌン</span></span>で刺身でも食おう〞</span></p>
</div>
</body>
</tt>
```

Sample Rendering



The above TTML snippet results in this rendering. We note that with tts:rubyReserve enabled, there is no relative movement of subtitles over time.



When tts:rubyReserve is not enabled, the base line of base text moves over time resulting in a jarring user experience.

Vertical Text

Basic vertical text

Netflix relies on the use of writingMode to indicate the vertical writing mode. Below is a sample illustration of vertical text (including specific vertical punctuation and symbols), and a TTML snippet using vertical text is provided below.

Ruby in vertical text

As shown in the following TTML snippet, the indication of ruby markup in vertical writing mode is no different from that in the horizontal writing mode.

TTML snippet

```
<?xml version="1.0" encoding="utf-8"?>
<tt xmlns="http://www.w3.org/ns/ttml" xmlns:ttm="http://www.w3.org/ns/ttml#metadata"
xmlns:ttp="http://www.w3.org/ns/ttml#parameter" xmlns:tts="http://www.w3.org/ns/ttml#styling"
ttp:frameRate="24" ttp:frameRateMultiplier="1000 1001" ttp:pixelAspectRatio="1 1" ttp:version="2"
tts:extent="1280px 720px" xml:lang="ja">
  <head>
    <styling>
      <initial tts:fontSize="6.0vh"/>
      <initial tts:lineHeight="7.5vh"/>
      <initial tts:showBackground="whenActive"/>
      <initial tts:textOutline="black 0.1em"/>
      <initial tts:fontFamily="notosans"/>
      <style xml:id="s1" tts:textCombine="all"/>
      <style xml:id="s2" tts:ruby="container"/>
      <style xml:id="s3" tts:ruby="base"/>
      <style xml:id="s4" tts:ruby="text" tts:rubyPosition="auto" tts:textOutline="black 0.1em"/>
      <style xml:id="s5" tts:textAlign="center"/>
      <style xml:id="s6" tts:textAlign="start"/>
      <style xml:id="s7" tts:fontSize="1.0em 1.5em"/>
      <style xml:id="s8" tts:fontShear="16.78842%"/>
      <style xml:id="s9" tts:fontShear="16.78842%" tts:textAlign="center"/>
<!-- set up tts:textEmphasis in initial section -->
      <style xml:id="s10" tts:textEmphasis="dot after"/>
    </styling>
    <layout>
      <region xml:id="横下" tts:displayAlign="after" tts:extent="80vw 30vh" tts:position="center bottom
10vh"/>
<!-- region for vertical writing mode -->
      <region xml:id="縦右" tts:extent="30vh 80vh" tts:position="right 10vw center"
tts:writingMode="tbrl"/>
    </layout>
  </head>
  <body region="横下" xml:space="preserve">
<div>
```

```
<!-- illustration of tts:textCombine in vertical writing mode -->
<p begin="00:00:37:07" end="00:00:40:00" region="縦右">まるで<span style="s1">23</span>歳のままだわ</p>
<!-- illustration of rubies in vertical writing mode -->
<p begin="00:05:46:20" end="00:05:49:00" region="縦右" style="s8">〝<span style="s2"><span style="s3">クリーンウォーター</span><span style="s4">ＣＷＩ</span></span>が<br/>労組を一掃〟</p>
<!-- illustration of tts:textEmphasis -->
<p begin="00:09:37:14" end="00:09:41:00" style="s5"><span style="s6">もし そうでも<br/><span style="s10">お相手</span>するって</span></p>
</div>
</body>
</tt>
```

Sample rendering

Tate-chu-yoko

tts:textCombine

tts:textCombine is used to realize the “tate-chu-yoko” feature. This feature helps increase legibility of subtitles.

TTML snippet

A ttml snippet for this feature can be found here.

Sample rendering

This is a rendering corresponding to the above TTML snippet.

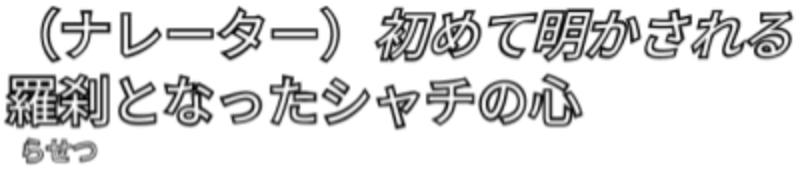
Slanted text

tts:fontShear

Japanese typography does not have italics fonts. This behavior is realized by performing geometric transformations on glyphs. The common value of tts:fontShear corresponds to a rotation by nearly 15°.

TTML snippet

A TTML snippet for this feature can be found here.



Font shear output

Boutens

tts:textEmphasis

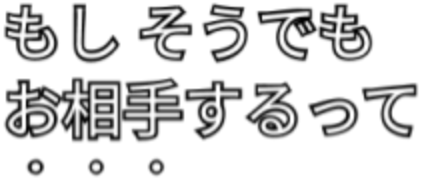
This is used for rendering boutens

TTML snippet

A TTML snippet for this feature can be found here.

Sample rendering

This is a rendering corresponding to the above TTML snippet



Boutens output