# Preprocessing XBRL files

The purpose of preprocessing XBRL files is to ensure that clean text is sent to the machine translation engine. This helps with producing high quality machine translations.

When identifying editing points, it’s important to keep in mind that the XBRL file will first be uploaded to CAT software, go through the designated segmentation process (splitting the document into individual sentences), then sent to the machine translation API. We will mostly be focused on preprocessing XBRL files before they are uploaded to CAT software as the files are often messy. Although there are also segmentation issues, these can get pretty complex. For now, just read about the [Pragmatic Segmenter](https://github.com/diasks2/pragmatic_segmenter) to get an idea of how segmentation works.

After a document goes through the segmentation process, the CAT software creates tags to note any changes in font, and the tags occasionally confuse the machine translation engine.

Font changes are particularly frequent in Japanese documents as each Japanese character requires two spaces (full-width), while each English character requires only one space (half-width). Document creators will often mix the full and half-width characters such as when referring to a foreign product name in English, using an acronym, or in some cases writing numbers with the keyboard input set to English (half-width). The issue is exacerbated as the Japanese character format includes the option to type full-width alphabet as well.

This will be a live document where I take note of common errors, their location, and characteristics. I will broadly categorize them into two categories: Easy edits and Hard edits. Easy edits will be quick-to-implement, “find and replace” level edits. Hard edits will probably require some level of programming.

### Character format examples

#### Vowels

Japanese (full-width): あいうえお

English (half-width): aiueo

#### Acronyms

Japanese (full-width): ＳＡＰ

English (half-width): SAP

#### Numbers

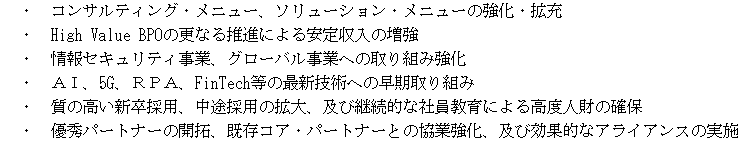
Japanese (full-width): １２３

English (half-width): 123

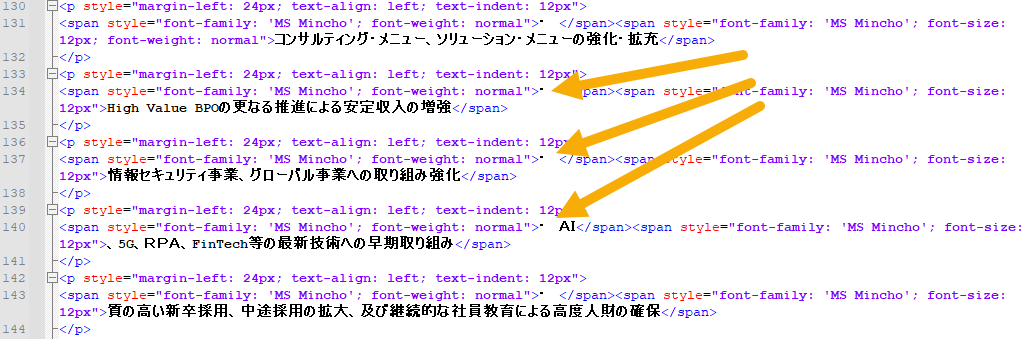
### Machine translation output examples

#### Full-width bullet point vs. half-width dash

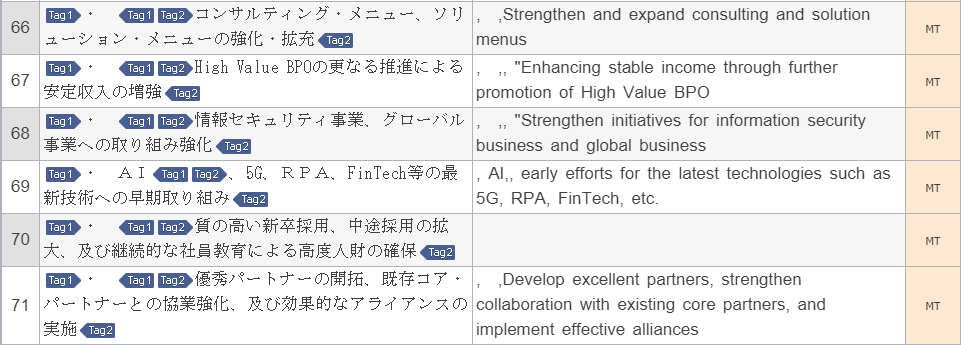
The following section found in Business Brain Showa-Ota’s FY03/20 Annual Securities Report contains full-width bullet points:



In an HTML editor (Notepad++), the same section appears as below. The orange arrows point to a few of the full-width bullet points:

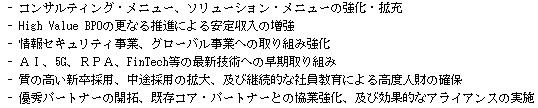


Here’s a screenshot from Wordfast Pro 6 after DeepL machine translations have been populated:

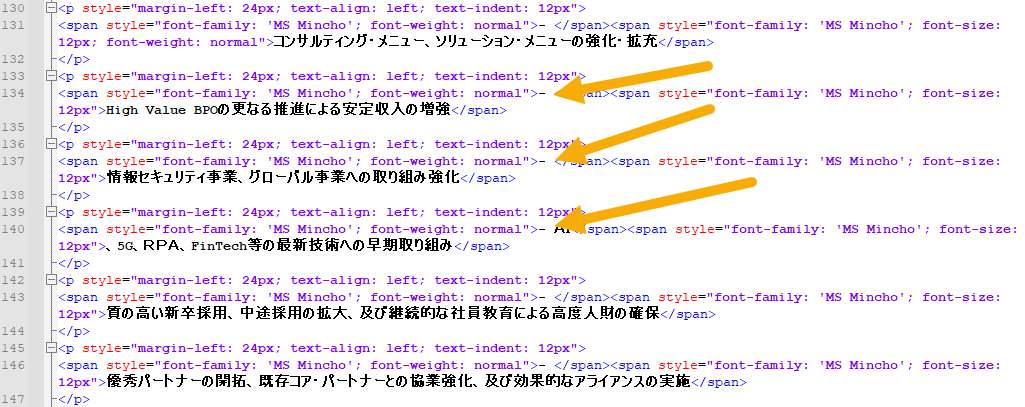


All translated segments contain mysterious commas and two segments contain open quotes. Segment 70 did not populate the first time around (the second try populated a reasonable translation with mysterious commas). None of the translations contain bullet points or dashes. Segment 69 contains a translation error caused by change in font in the source (Japanese) text (“AI” is in full-width, “5G” in half-width, “RPA” in full-width, “FinTech” in half-width)

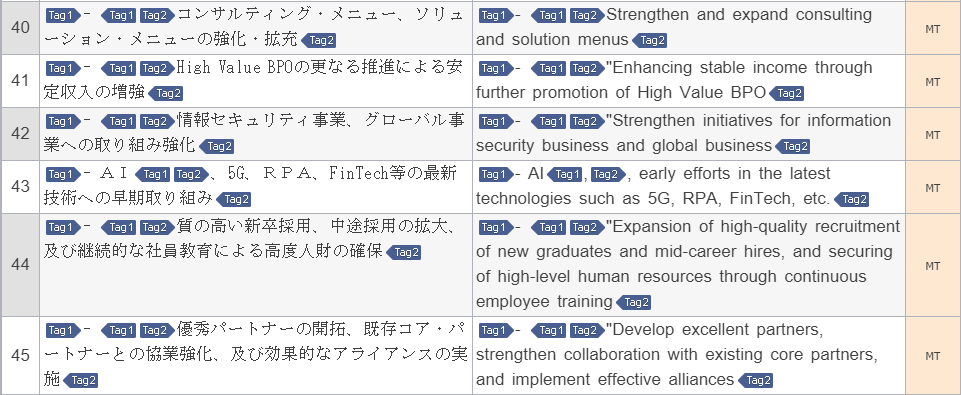
Here’s what the file looks like after finding and replacing full-width bullet points with a half-width dash in Notepad++.



In Notepad++:



In WFP6:



All translated segments contain a dash, 4 translated segments contain open quotes.

No mysterious commas. The 4 open quotes should not be there, but the error is non-critical.

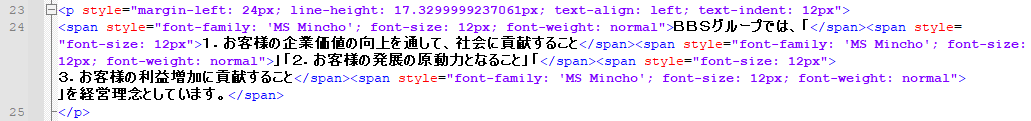
#### Japanese quotes (「」) vs. English quotes (“”)

Japanese quotes are also a common cause for error for machine translation engines, probably because it is unique to the Japanese language. The tricky part here is that Japanese documents occasionally contain separate quotes in succession with no separator (i.e., a comma).

The following section found in Business Brain Showa-Ota’s FY03/20 Annual Securities Report contains Japanese quotes in succession:



In Notepad++, this section appears as below:



Here’s a screenshot from Wordfast Pro 6 after DeepL machine translations have been populated:



DeepL returned a strange string of characters and no translation, making this a critical error. This happens from time to time when dealing with too many Japanese quotes.

To fix this, here’s what I did in Notepad++:

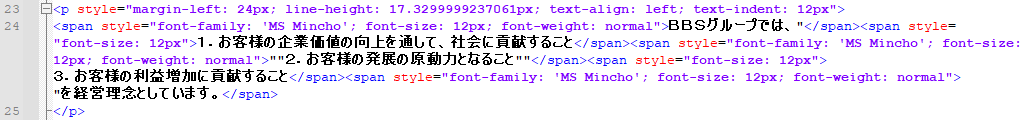
1. Find (」「) and replace (“”)
2. Find (「) and replace (“)
3. Find (」) and replace (“)

The same portion in the revised file looks like this:

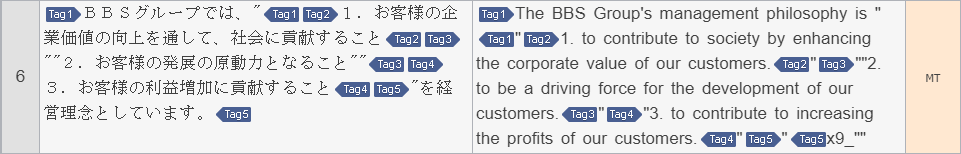


Not pretty in Japanese, but easier for computers to read.

In Notepad++, the revised section appears as below:



In WFP6:



Mysterious string of characters appeared at the end of the machine translation output, but this error is non-critical. Overall, the sentence is coherent (and accurately translated), although stylistically poor (non-critical).

The following is a non-exhaustive list of edits to consider for preprocessing. They are broadly categorized into easy edits (think: find & replace) and hard edits (think: regex or programming).

## Easy edits

| Find | Desc. | Replace | Desc. |
| --- | --- | --- | --- |
| ・ | Full-width bullet point + space | - | Half-width dash + space |
| 」「 | Japanese close quote + Japanese open quote | ““ | Close quote + open quote |
| 「 | Japanese open quote | “ | Open quote |
| 」 | Japanese close quote | “ | Close quote |

## Hard edits

### Hard returns inside table cells of XBRL files

All Japanese Annual Securities Report filings seem to have the same issue of containing hard returns within a cell, rather than wrapping the text, when the content of the cell does not fit into the designated space. This is an issue because when we feed one sentence that is broken up in pieces on different lines to the machine translation engine, the machine translation engine views each piece as a separate sentence. This in turn generates incorrect translations and messy formatting.

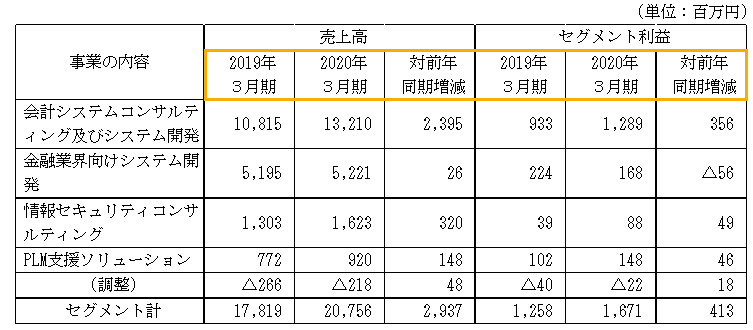
While in many cases, the fix simply involves removing all hard returns inside a cell, there are also cases where hard returns are placed to split a phrase or sentence. In the latter case, we would be concatenating sentences that should not be concatenated, which will produce translation errors.

First, we will go over a simple case.

#### Simple case

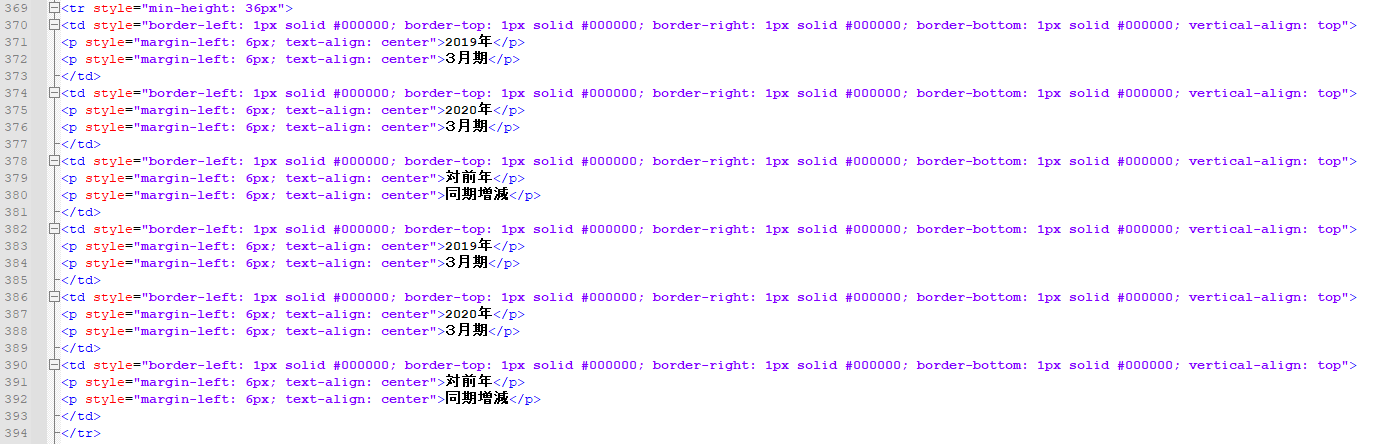
Here is an example from Business Brain Showa-Ota’s FY03/20 Annual Securities Report (filename: [0102010\_honbun\_jpcrp030000-asr-001\_E04869-000\_2020-03-31\_01\_2020-06-25\_ixbrl](https://drive.google.com/file/d/1eidykMOI080yP8WZUYdJV7tOrzDAEw7h/view?usp=sharing))

The 1st table in the above file looks like this (in your browser):

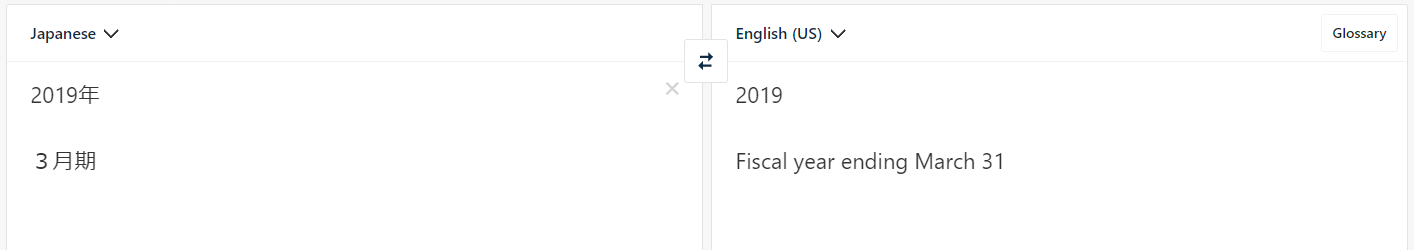


The area highlighted in orange is where we will focus on. This table shows sales and profit by business segment. The first two cells in the highlighted area (from left to right), shows the fiscal years, with the 3rd cell showing YoY change. Then the 4th and 5th cells show fiscal years, followed by YoY change in the 6th cell.

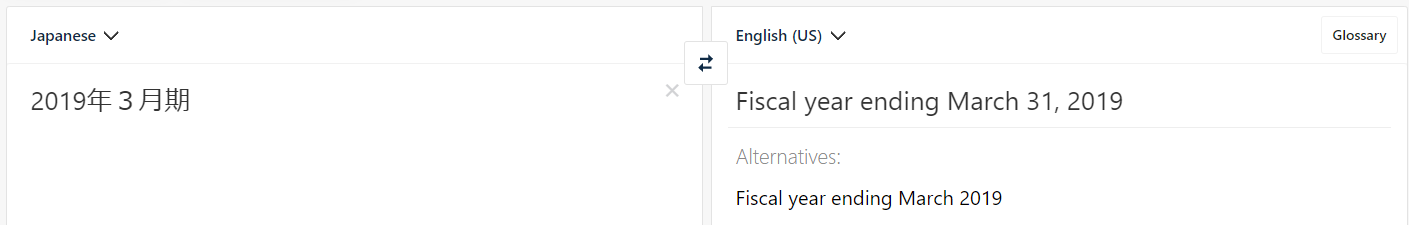
When opening the same file in Notepad++, the section for this table starts at line 348, and the area highlighted in orange starts at line 369.



As you can see, there are technically two paragraphs in each cell. The machine translation engine treats each paragraph as separate entries, which means there will technically be two entries for each cell. To replicate the problem, copy the content of the first cell in your browser, then paste it into DeepL.



To see what it’s supposed to look like, remove the hard returns and see the output:

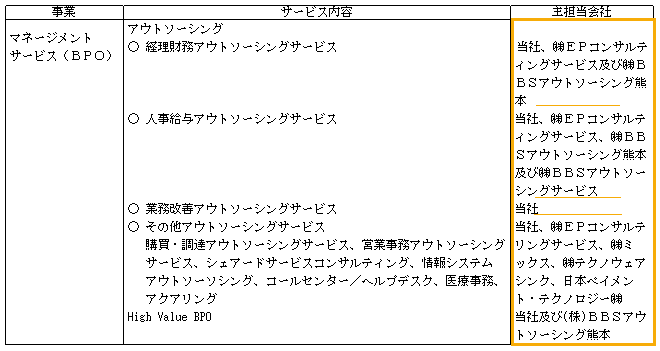


Next, we will go over the complex case

#### Complex case

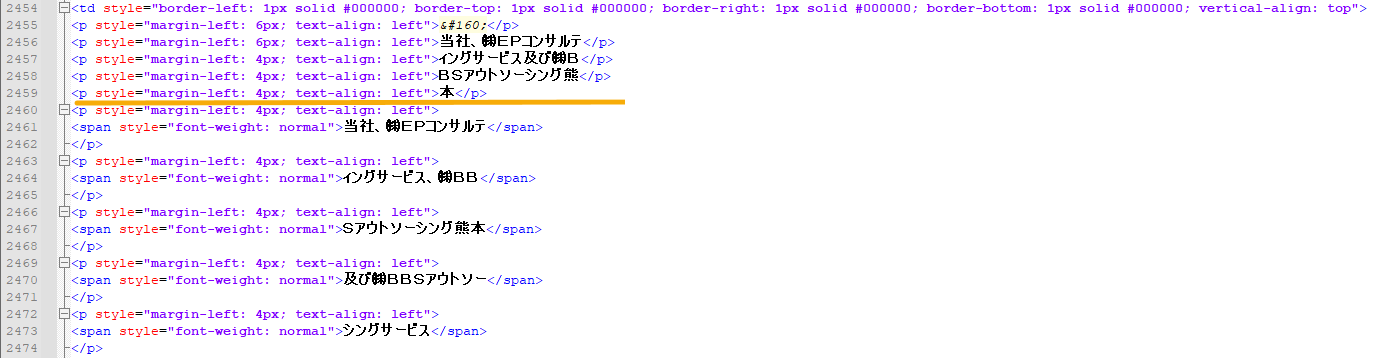
Here is an example from Business Brain Showa-Ota’s FY03/20 Annual Securities Report (filename: [0101010\_honbun\_jpcrp030000-asr-001\_E04869-000\_2020-03-31\_01\_2020-06-25\_ixbrl](https://drive.google.com/file/d/19-6WdHOufW3q5iRUpi054uFNRFp9cZZP/view?usp=sharing))

The 5th table in the above file looks like this:



The area highlighted in orange is where we will focus on. The orange lines within the cell are places where hard returns are actually necessary.

When opening the same file in Notepad++, the section for this table starts at line 2,389, and the area highlighted in orange starts at line 2,454.



The orange line is where the hard return is actually necessary. I have no idea how to approach this issue and there is seemingly no way to programmatically distinguish where a hard return is necessary/unnecessary.

One roundabout way of approaching this might be to capture the width of the cell (from <thead> element), estimate how many characters would fit in a single line based on font style (full-width vs. half-width) and font size, then for each hard return, check whether the previous line is filled edge to edge. Presumably, if the previous line is filled edge to edge, the line following will be a continuation of the previous line. If the previous line is not filled edge to edge, then it’s highly likely that the following line is a new phrase and not a continuation of the previous line.