Kana Parser for LuaTEX

Greetings, reader. This document will describe this LuaTEX package in detail, providing all the information you need to start using this package. We will analyze the Japanese writing system and see how it relates to the Latin script. Then we will see how this package handles the conversion.

1. The Japanese writing system

The modern Japanese uses four distinct character sets: Latin (also known as 'romaji' in Japanese), a pair of syllabic sets known as kana and ultimately kanji, the complex ideographic set borrowed from the Chinese. They combine these four sets regularly, a practice usually very confusing for newcomers to the language.

Kanji is based on a subset of Chinese ideograms known as 'hanzi' in China and cannot be transliterated to Latin by a simple state automaton, requiring solid context awareness.

Kana, however, is a phonetic system based on syllables which can be directly transliterated to Latin. The two kana sets are known as hiragana and katakana. Kana represents a set of roughly 46 syllables (48 including two obsolete ones), each syllable has a hiragana and a katakana character assigned to it. There are five vowel characters (a, i, u, e, o), an 'n' character and the rest are syllabic compounds of vowels and consonants, such as 'wo'.

Hiragana is used for native syntactic and grammatic constructs as well as common words and phrases. It's also used in material intended to be read by juveniles and children who do not yet understand complex kanji.

Katakana is used for loanwords, foreign words and usually onomatopoeia among other uses. The two kanas cover the same set of syllables and as such can be freely converted between each other.

2. Differences between the kanas

Despite covering the same syllable set, there are certain differences between the systems. The most striking difference is in how the sets prolong their syllables. Prolongation here means extending a vowel-terminated syllable by a pure vowel, getting a syllable of double length. An example of this is ma => maa.

Hiragana prolongs syllables by explicitly putting a vowel character after a syllable: ま => まあ. Here you can see how an あ gets appended to ま to prolong it. Syllables ending in o and e are instead prolonged by u and i, respectively: mo => mou (も => もう)

Katakana uses a single prolonging character, ー, to prolong any vowel-terminated syllable. This package ensures this character is always correctly transliterated to its respective hiragana vowel or Latin vowel. モ => モー in katakana translates correctly to も => もう in hiragana and mo => mou in Latin.

Another difference is in katakana's added support for various foreign syllables. These syllables don't exist in native Japanese, such as vu (ヴゥ). These syllables help in better representing foreign words and as such don't commonly have hiragana counterparts. However, thanks to the inter-compatibility of kana character set, even these syllables can be written in hiragana, although such use is very unusual: vu (ゔぅ). This package supports such conversions to promote learning of the character sets.

3. Consonant gemination

Japanese language supports doubling (or gemination) of certain unvoiced consonants (s, t, k, p, ch) when they appear at the beginning of a syllable. An example of this is the syllable 'ka' (か) which turns into 'kka' (っか) when geminated. As seen in the example, the kana sets have a special character, っ, called sokuon (little tsu), a small version of the 'tsu' (つ) character, which is placed in front of the syllable which is to be geminated. This package detects correct usage of sokuon and represents it in Latin by doubling the respective consonant. In several romanization systems, gemination is represented by using 't' instead in all cases but I find the doubling of the affected consonant a better way to show the true nature of sokuon.

4. Ambiguity of ‘n’

N is the only consonant in Japanese with its own kana character, ん. As such, there is some ambiguity in following it by other characters. There are several syllables beginning in ‘n’, such as nya (にゃ) or nyo (にょ), which could be ambiguously split into ‘n-ya’ (んや) and ‘n-yo’ (んよ) respectively. To make sure there is no ambiguity in romanization of these characters, an isoLating delimiter is used: ’. To demonstrate its usage, ‘nyaa’ becomes ニャー in katakana but ‘n’yaa’ becomes ンヤー — ambiguity resolved. This works backwards too, where れんようけい which contains the ‘nyo’ syllable split to ‘n-yo’ transliterates to ren'youkei.

5. Transliteration alternatives

As expected with completely different writing systems, the conversion between them is not really isomorphic. Several syllables have multiple kana representations and several kana characters have multiple romanization options. To tackle this problem, this package tries to be as permissive as possible by letting the user configure alternatives on the go. The most frequent alternatives are selected by default and can be viewed in the kanaparser.tex file. There is a switch macro in the package that lets the user choose which kana character(s) will be used in place of the selected syllable if that syllable supports alternatives. There is always at most one alternative to a syllable representation. For example, if you wish that ‘we’ is not written as ウェ in katakana but instead as the obsolete ヱ, the package lets you do it. On the other hand, ‘sisi’ and ‘shishi’ will both transliterate to しし although backwards transliteration will always be the closer-sounding shishi. Romanization of all the alternative kana characters is enabled by default.

6. Transliterating mixed character sets and special characters

This package has limited support for this feature. Its three macros always attempt to transliterate as much as they can into the target character set. There is no option to only transliterate hiragana, for example. When targetting Latin, both kana sets will be converted. Same goes for transliterating to the kanas, both Latin and the other kana set will be converted. Characters not understood by the used macro (including ”) will be left unchanged except for apostrophes (’), which will be consumed (and treated as isolation delimiters) when transLating to kana.

7. Introducing romanization systems

There are several systems for romanization of Japanese and this package loosely follows the Hepburn system (ヘボン式ローマ字). The first difference is that the package ignores the characters with macron in long syllables (such as ō). This is to stay within the ASCII character set (which simplifies typing on a common keyboard) and lets newcomers to the language get used to the prolongation rules. As such, こうこう transliterates to koukou instead of kōkō. Contextual variations are also ignored in this package, such as writing は as wa when used as a topic particle. Another notable deviation from Hepburn is not using ‘t’ for consonant gemination except for syllables beginning in ‘t’. As such, まっちゃ becomes maccha and not matcha.

8. Fonts, unicode and implementation

Kana are multibyte unicode characters, a compatible font is needed to display any of them, hence the bundled macros won’t print anything readable without a font with japanese support. An example of such font is the ipafont family.

Both Lua and LuaTEX support unicode characters although Lua only considers them multibyte strings. As such an UTF-8 tokenizer is needed to properly recognize individual characters. Once tokenized, conversion both to and from kana sets is possible using a state automaton with a processing buffer.

When converting Latin to kana, a three-character buffer is needed to process characters such as ‘nya’ (にゃ); the other way around only two-character is required to process multi-character compounds. Based on the contents of this buffer the automaton decides what to transliterate, prolong, geminate or print as-is. Conversion between kana sets is implemented as a simple translation table.