

KKsymbols Package Documentation

Kosei Kawaguchi a.k.a. KKT_{EX}

Version 2.2.2 (2026/06/26)

目次

1	Outline	3
2	Comparison with <code>luatexja-otf</code>	4
2.1	Command-by-command examples	5
2.2	Practical selection guide	5
3	Acknowledgements / Credit	5
4	Installation	6
4.1	Dependencies	6
4.2	Loading and Options	6
5	Caution	6
6	Commands	7
6.1	The maru series	7
7	Roman numerals	8
8	Automatic alignment (reference box)	9
8.1	Overriding the reference	9
9	Rotation	10
9.1	Commands	10
9.2	Vertical mode	11
10	The seihou series	12
11	The kakko series	13
11.1	Additional Description: <code>\ichimoji</code> and <code>\zenkakuhabafixer</code>	14
12	License	14
13	Version History	14

1 Outline

Japanese このドキュメントでは、主要な仕様説明を日本語と英語の両方で記述します。日本語の説明で実用上の注意点を先に述べ、その直後に対応する英語説明を置きます。

このパッケージは、既存の OTF グリフだけに頼るのではなく、「現在のフォント」「任意の引数」「横書き・縦書き」に合わせて、丸数字や括弧付き文字などの囲み記号を構成するために作られています。

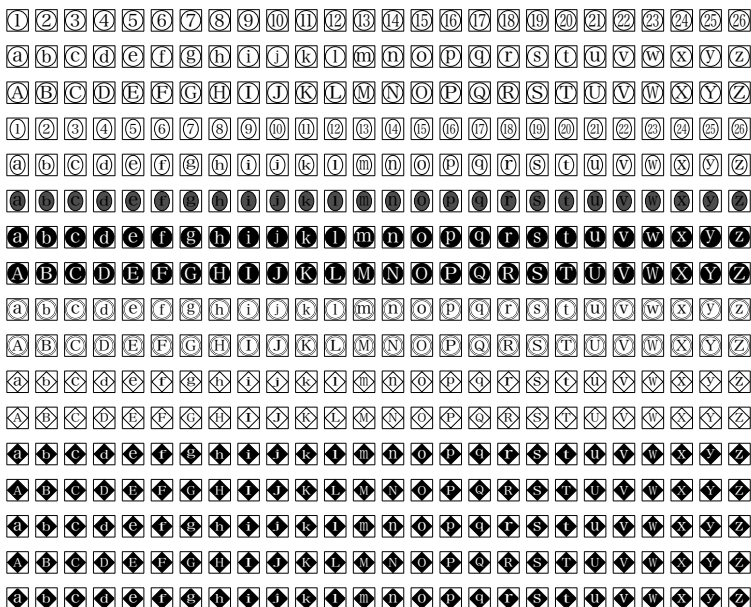
`luatexja-otf` の `\ajMaru` などは、Adobe-Japan1 の既存グリフを呼び出すため、品質が高く高速である一方、どの番号がどのグリフになるかを知るための早見表や CID 知識が必要になりがちです。本パッケージは図形と中身を組み合わせる再構成する設計なので、任意の文字列を同じインターフェースで囲めます。

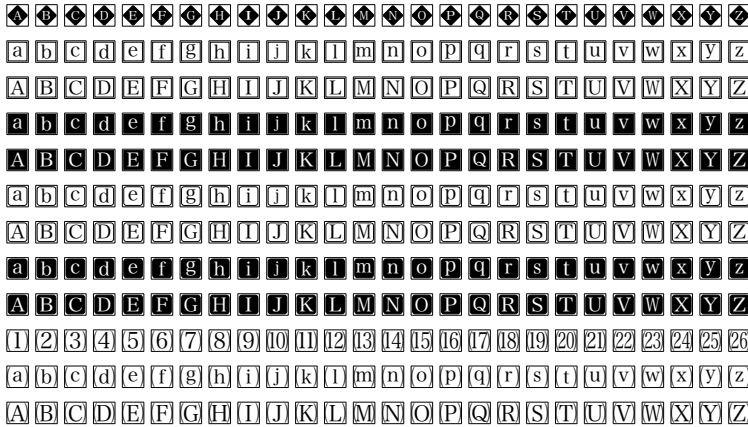
English This documentation now gives the important usage notes in both Japanese and English. The Japanese explanation states practical details first, followed by the corresponding English explanation.

This package is designed to build enclosed symbols such as circled numbers and parenthesized characters from the current font and arbitrary arguments, in both horizontal and vertical writing modes, rather than relying only on pre-existing OTF glyphs.

Commands such as `\ajMaru` in `luatexja-otf` call Adobe-Japan1 glyphs directly. This gives high-quality and fast output, but it often requires a reference chart or CID knowledge to know which number maps to which glyph. Kksymbols instead composes the enclosure and the content, so arbitrary strings can be enclosed through the same interface.

あ あ (あ) あ あ





2 Comparison with luatexja-otf

Japanese `luatexja-otf` は Adobe-Japan1 のグリフを直接呼び出すための強力なパッケージです。代表的なコマンドには、`\ajMaru`、`\ajKuroMaru`、`\ajKaku`、`\ajKakko`、`\ajRoman`、`\ajroman`、`\ajKakkoroman`、`\ajKakkoRoman` などがあります。これらは既存グリフをそのまま使うため、印刷品質は非常に高く、処理も軽量です。

一方で、`luatexja-otf` の囲み文字は「存在するグリフを選ぶ」方式です。たとえば `\ajMaru{1}` は丸数字のグリフを出しますが、`\ajMaru{ABC}` のような任意文字列を丸で囲む用途ではありません。対応範囲は Adobe-Japan1 側に存在するグリフと、`luatexja-ajmacros` が定義している番号範囲・合字名に依存します。

`KKsymbols` は反対に、「囲みの形」と「中身」をその場で組み合わせる方式です。そのため `\maru{ABC}`、`\kakko{QED.}`、`\seihou{長い文字列}` のように、数字以外の文字列や現在フォントの文字も同じ命令で処理できます。代償として、既存の完成グリフを呼び出す `luatexja-otf` よりも内部処理は重く、極端な引数では縮小や中央寄せの調整が見た目に出ることがあります。

English `luatexja-otf` is a powerful package for accessing Adobe-Japan1 glyphs directly. Typical commands include `\ajMaru`, `\ajKuroMaru`, `\ajKaku`, `\ajKakko`, `\ajRoman`, `\ajroman`, `\ajKakkoroman`, and `\ajKakkoRoman`. Since these commands use pre-designed glyphs, the output quality is excellent and the processing cost is low.

The important limitation is that `luatexja-otf` selects existing glyphs. For example, `\ajMaru{1}` outputs a circled-number glyph, but `\ajMaru{ABC}` is not a general command for enclosing an arbitrary string. Its coverage depends on the glyphs available in Adobe-Japan1 and on the numeric ranges and ligature names defined by `luatexja-ajmacros`.

`KKsymbols` takes the opposite approach: it composes the enclosure and the content at typesetting time. Therefore commands such as `\maru{ABC}`, `\kakko{QED.}`, and `\seihou{long text}` work with non-numeric strings and with the current font. The tradeoff is that `KKsymbols` does more internal work than `luatexja-otf`, and very long or unusual arguments may visibly depend on the package's

scaling and centering rules.

2.1 Command-by-command examples

Japanese 次の表は、同じような用途に見えるコマンドの比較です。左側の `luatexja-otf` は既存グリフ、右側の `KKsymbols` は現在フォントから組み立てた結果です。

English The following table compares commands that look similar in ordinary use. The `luatexja-otf` examples on the left are existing glyphs, while the `KKsymbols` examples on the right are composed from the current font.

表 1: Comparison with `luatexja-otf`

Purpose	<code>luatexja-otf</code>	<code>KKsymbols</code>
circled number	<code>\ajMaru{8}</code> ⑧	<code>\maru{8}</code> ⑧
black circled number	<code>\ajKuroMaru{8}</code> ⑧	<code>\kuromaru{8}</code> ⑧
square number	<code>\ajKaku{8}</code> ⑧	<code>\seihou{8}</code> ⑧
parenthesized number	<code>\ajKakko{8}</code> (8)	<code>\kakko{8}</code> (8)
uppercase roman	<code>\ajRoman{8}</code> VIII	<code>\kakko{\Rrnum*{8}}</code> (VIII)
lowercase roman	<code>\ajroman{8}</code> viii	<code>\kakko{\Rrnum{8}}</code> (viii)
parenthesized roman	<code>\ajKakkoroman{8}</code> (viii)	<code>\kakko{\Rrnum{8}}</code> (viii)

2.2 Practical selection guide

Japanese 完成済みの標準グリフが必要な場合、たとえば丸数字・括弧付き数字・ローマ数字が Adobe-Japan1 の範囲に収まる場合は、`luatexja-otf` が最も安定します。フォントベンダが設計した字形なので、紙面上の完成度も高くなります。

任意の文字列、現在の欧文・和文フォント、太字、色付き文字、長いラベル、独自の囲み形状が必要な場合は `KKsymbols` が向いています。`KKsymbols` は「グリフを探す」必要がなく、`\maru{任意}` や `\kakko{ABC}` のように直接書けます。

English Use `luatexja-otf` when you need standard pre-designed glyphs, such as circled numbers, parenthesized numbers, or roman numerals that are covered by Adobe-Japan1. Since these glyphs are designed by the font vendor, they usually give the most polished printed result.

Use `KKsymbols` when you need arbitrary strings, the current Latin or Japanese font, bold or colored content, long labels, or custom enclosure styles. `KKsymbols` does not require you to look up a glyph; you can write commands such as `\maru{任意}` or `\kakko{ABC}` directly.

3 Acknowledgements / Credit

In developing this package, I made extensive use of the advice I received from Mr. Yusuke Terada.

I recommend you to refer to his article when you develop new-type symbols on L^AT_EX.

<https://doratex.hatenablog.jp/entry/20211205/1638697391>

4 Installation

Place `KKsymbols.sty` in a directory where LaTeX can find it, e.g., your local `texmf` tree or alongside your document.

4.1 Dependencies

This package depends on the following packages.

- `LuaLaTeX-jā`
- `tikz`
- `clac`
- `luacode`
- `kvoptions`

4.2 Loading and Options

Load the package: `\usepackage[<options>]{KKsymbols}`

Currently, the following package option is provided.

tsumesuji To specify whether or not to “shrink” the box width occupied by “1”. By default, `tsumesuji=1` is specified, and the effect is activated. If you deactivate it, `tsumesuji=0` will do.

To clarify, this option is applied when the argument of a command subject to the `tsumesuji` option—after expansion—is a numeric string of two or more digits and contains the digit “1”. The difference is present in the following example:

Input	
1	<code>% NOT APPLIED</code>
2	<code>\kakko{\vphantom{X}12}</code>
3	
4	<code>% APLIED</code>
5	<code>\kakko{12}</code>

Output	
	(12)
	(12)

5 Caution

Since this package internally calls `\ltjghostbeforejchar` and `\ltjghostafterjchar`, it can be used only in a LuaLaTeX environment.

6 Commands

6.1 The maru series

This package provides `\maru`, `\kuromaru`, and `\nmaru`. Each of them takes one mandatory argument and no optional arguments. You can pass strings of any length and in any font as arguments.

In most cases, `\maru{argument}` will meet your demands. However, only when you take lowercase alphabet in these commands, you must use star-command just like `\maru*{m}`¹⁾.

Input

Mind the star option!

```
1 % Normal Characters
2 \maru{A}\maru{あ}\maru{QED.}
3
4 % Lowercase Alphabetic Characters
5 \maru*{a}\maru*{j}\maru*{z}
```

Output



They are used as follows.

They behave as if they were single kanji or hiragana characters:

あいう(あ)あいう①②③あいうえお

The spacing between `\maru` and other characters is adjusted using `\ltjghostbeforejchar` and `\ltjghostafterjchar` so that it behaves like hiragana or kanji.

When changing the font size using commands such as `\Large`, each command is scaled proportionally according to the font size change:

●●●
①②③

1) `\jegg` is an exception of this rule. When you use `\jegg`, you don't have to put the star option no matter the argument is lowercase or not. If you do it, the background color of the `\jegg` changes into gray. Only in this case, the effect of the option is different.

表 2: maru series

argument	\maru	\kuromaru	\nmaru	\jegg	\jegg*
1	①	❶	①	①	❶
97	⑨⑦	❹⑦	⑨⑦	⑨⑦	❹⑦
だ	①	❶	①	①	❶
ばばば	③③③	❸③③	③③③	③③③	❸③③
m	③③③	❸③③	③③③	③③③	❸③③
Qjg	③③③	❸③③	③③③	③③③	❸③③



You can also change the current font:



7 Roman numerals

Japanese KKsymbols 自体はローマ数字を生成しません。ローマ数字の生成には、通常 KKran パッケージが提供する `\Rrnum` を使います。`\Rrnum{8}` は小文字ローマ数字、`\Rrnum*{8}` は大文字ローマ数字を出します。

大文字ローマ数字は大文字アルファベットと同じ扱いで、通常は非スター版の囲みコマンドに入れます。たとえば `\kakko{\Rrnum*{8}}` や `\maru{\Rrnum*{8}}` のように書きます。

小文字ローマ数字は注意が必要です。小文字の `i`、`v`、`x` は字ごとに高さが違うため、単に `\kakko{\Rrnum{1}}` から `\kakko{\Rrnum{10}}` までを並べると、囲み内部での縦方向の見え方が完全には揃わないことがあります。これはバグではなく、引数の実際の高さが異なるためです。

小文字ローマ数字をリストラベルなどで厳密に揃えたい場合は、`\kksref{\Rrnum{6}}` をグループ内で指定してください。これは各囲みコマンドの内部に `\vphantom{\Rrnum{6}}` 相当の基準を加えるためのフックです。

English KKsymbols itself does not generate roman numerals. In ordinary use, roman numerals are supplied by the KKran package through `\Rrnum`. `\Rrnum{8}` gives a lowercase roman numeral, while `\Rrnum*{8}` gives an uppercase roman numeral.

Uppercase roman numerals behave like uppercase Latin letters. They should normally be used with the non-starred enclosure commands, for example `\kakko{\Rrnum*{8}}` or `\maru{\Rrnum*{8}}`.

Lowercase roman numerals require more care. The letters `i`, `v`, and `x` have different heights, so a sequence from `\kakko{\Rrnum{1}}` to `\kakko{\Rrnum{10}}` may not look vertically identical inside the enclosures. This is not a bug; it follows from the actual glyph metrics of the arguments.

If lowercase roman numerals must be strictly aligned, for example in list labels, set `\kksref{\Rrnum{6}}` inside a group. It works as a hook that adds a reference equivalent to

`\vphantom{\Rrnum{6}}` inside each enclosure command.

Input	Roman numerals
<pre>1 % Uppercase roman numerals: use the non-starred enclosure command. 2 \kakko{\Rrnum*{1}}\kakko{\Rrnum*{2}}\kakko{\Rrnum*{3}} 3 4 % Lowercase roman numerals: automatic reference only. 5 \kakko{\Rrnum{1}}\kakko{\Rrnum{2}}\kakko{\Rrnum{3}}\kakko{\Rrnum{6}} 6 7 % Lowercase roman numerals: force a common reference. 8 {\kksref{\Rrnum{6}}}% 9 \kakko{\Rrnum{1}}\kakko{\Rrnum{2}}\kakko{\Rrnum{3}}\kakko{\Rrnum{6}}}</pre>	
Output	
	(I)(II)(III) (i)(ii)(iii)(vi) (i)(ii)(iii)(vi)

8 Automatic alignment (reference box)

Japanese v2.2.0 以降、各囲みコマンドは既定で「現在フォントの参照高さ」、具体的には x-height 相当を基準として中身のサイズと中央位置を決めます。これは短い文字や深さのない文字を、引数自身のバウンディングボックスだけで処理したときのばらつきを抑えるためです。

ただし、既定の自動基準は `\vphantom{\Rrnum{6}}` と同一ではありません。したがって、小文字ローマ数字を `\Rrnum{6}` の高さ・深さで揃えたい場合は、前節のように `\kksref{\Rrnum{6}}` を明示してください。

漢字・かな・大文字・数字など、既定基準より背の高い内容は、引数自身の寸法が優先されます。このため、通常の文書では既存の出力が大きく変わらないようになっています。

English Since v2.2.0, each enclosure command uses the current font's reference height, roughly its x-height, as the default reference for sizing and vertical centering. This reduces variation when short glyphs or glyphs with no depth would otherwise be measured only by their own bounding boxes.

However, the automatic reference is not identical to `\vphantom{\Rrnum{6}}`. If lowercase roman numerals should be aligned against the height and depth of `\Rrnum{6}`, explicitly use `\kksref{\Rrnum{6}}` as shown in the previous section.

For content taller than the default reference, such as kanji, kana, uppercase letters, or digits, the argument's own metrics still dominate. This keeps ordinary documents close to their existing appearance.

8.1 Overriding the reference

\kksref{<material>} **Japanese:** <material> の高さ・深さを、既定の自動基準の代わりに使います。効果は `\RotYoko` と同じくスコープ依存なので、通常はグループで囲みます。

English: Use the height and depth of <material> as the reference instead of the automatic one. The effect is scoped like `\RotYoko`, so enclose it in a group.

```
Input
1  {\kksref{\Rrnum{6}}}%
2  \maru{\Rrnum{1}}\maru{\Rrnum{4}}\maru{\Rrnum{8}}
```

\kksreffoff **Japanese:** 参照ボックスを無効化し、v2.2.0 以前のように引数自身の寸法を使います。

English: Disable the reference box and fall back to the argument's own metrics, which is the behavior before v2.2.0.

\kksrefauto **Japanese:** 既定の自動基準に戻します。`\kksrefreset` は別名です。

English: Restore the automatic reference. `\kksrefreset` is an alias.

9 Rotation

9.1 Commands

This package provides `\RotTate` and `\RotYoko`. The differences are as follows:

In horizontal mode You should use `\RotYoko`. The default value is 0. Therefore, if you use `\RotYoko` with no arguments, this is equal to `\RotYoko[0]`

In vertical mode You should use `\RotTate`. The default value is 90. Therefore, if you use `\RotTate` with no arguments, this is equal to `\RotTate[90]`

From a technical perspective: Commands like `\maru` provided by this package automatically rotate their arguments based on the “current typesetting direction” Specifically, the package applies a 0-degree rotation for horizontal writing (yoko-gaki) and a 90-degree rotation for vertical writing (tate-gaki).

The commands `\RotYoko` and `\RotTate` redefine this “automatic rotation angle” to the value specified in their arguments. Consequently, the effect is persistent unless localized (similar to how font-size commands like `\small` behave). Therefore, please ensure you use appropriate scoping, such as enclosing the command within curly braces `{...}`, when applying these settings.

Input

```

1 % In horizontal mode
2 {\RotYoko[45]\kakko{あ}\kakko{い}\kakko{う}}\par
3 {\RotYoko[60]\kakko{1}\kakko{2}\kakko{3}}
4
5 % In vertical mode
6 \parbox<t>{5\zw}{% <tb option requires lljtext package.
7   {\RotTate[45]\kakko{あ}\kakko{い}\kakko{う}}\par
8   {\RotTate[60]\kakko{1}\kakko{2}\kakko{3}}
9 }

```

Output

The output shows two rows of text. The first row, corresponding to the horizontal mode code, displays three hiragana characters 'あ', 'い', and 'う' rotated 45 degrees. The second row, corresponding to the vertical mode code, displays three hiragana characters '1', '2', and '3' rotated 60 degrees.

9.2 Vertical mode

When you want to typeset ㊦, for instance, in vertical mode, you should use `\RotTate` command.

As described in the previous subsection, the effect of `\RotTate` lasts, when localized, in a certain group. So when you typeset hiragana or kanji, use it like this:

Input

```

1 % In vertical mode
2 \parbox<t>{5\zw}{%
3   {\RotTate[0]\kakko{あ}\kakko{い}\kakko{う}}\par
4   \kakko{1}\kakko{1}\kakko{3}
5 }

```

Output

The output shows three hiragana characters 'あ', 'い', and 'う' in vertical mode. The first two are rotated 0 degrees, and the third is rotated 3 degrees.

10 The seihou series

The commands introduced below are used in exactly the same way as the maru series. In most cases, `\seihou{argument}` will meet your demands. However, only when you take lowercase alphabet in these commands, you must use star-command just like `\seihou*{m}`.

表 3: seihou series

argument	<code>\seihou</code>	<code>\kuroseihou</code>	<code>\seimaru</code>	<code>\kuroseimaru</code>
1				
97				
だ				
ばばば				
m				
Qjg				

表 4: hishi series

argument	<code>\hishi</code>	<code>\kurohishi</code>	<code>\maruhishi</code>	<code>\kuromaruhishi</code>
1				
97				
だ				
ばばば				
m				
Qjg				

Input

Mind the star option!

```

1 % Normal Characters
2 \seihou{A}\seihou{あ}\seihou{QED.}
3
4 % Lowercase Alphabetic Characters
5 \seihou*{a}\seihou*{j}\seihou*{z}

```

Output

11 The kakko series

The commands introduced below are used in exactly the same way as the maru series.

In most cases, `\kakko{argument}` will meet your demands. However, only when you take lowercase alphabet in these commands, you must use star-command just like `\kakko*{m}`²⁾.

表 5: kakko series ①

argument	<code>\kakko</code>	<code>\sumikakko</code>	<code>\kakukakko</code>	<code>\kikakko</code>	<code>\ykakko</code>
1	(1)	【1】	[1]	〔1〕	⟨1⟩
97	(97)	【97】	[97]	〔97〕	⟨97⟩
だ	(だ)	【だ】	[だ]	〔だ〕	⟨だ⟩
ばばば	(ばばば)	【ばばば】	[ばばば]	〔ばばば〕	⟨ばばば⟩
m	(m)	【m】	[m]	〔m〕	⟨m⟩
Qjg	(Qjg)	【Qjg】	[Qjg]	〔Qjg〕	⟨Qjg⟩

表 6: kakko series ②

argument	<code>\nykakko</code>	<code>\namikakko</code>	<code>\kagikakko</code>	<code>\nkagikakko</code>	<code>\ichimoji</code>	<code>\zenkakuhabafixer</code>
1	⟨1⟩	{1}	「1」	『1』	1	1
97	⟨97⟩	{97}	「97」	『97』	97	97
だ	⟨だ⟩	{だ}	「だ」	『だ』	だ	だ
ばばば	⟨ばばば⟩	{ばばば}	「ばばば」	『ばばば』	ばばば	ばばば
m	⟨m⟩	{m}	「m」	『m』	m	m
Qjg	⟨Qjg⟩	{Qjg}	「Qjg」	『Qjg』	Qjg	Qjg

Input	Mind the star option!
<pre> 1 % Normal Characters 2 \kakko{A}\kakko{あ}\kakko{QED.} 3 4 % Lowercase Alphabetic Characters 5 \kakko*{a}\kakko*{j}\kakko*{z} </pre>	
Output	
<pre> (A)(あ)(QED.) (a)(j)(z) </pre>	

2) `\zenkakuhabafixer` is an exception of this rule. It does not take a star option.

11.1 Additional Description: `\ichimoji` and `\zenkakuhabafixer`

The major difference of `\ichimoji` and `\zenkakuhabafixer` is that the former changes the vertical scale to force its totalheight to `\zw`, but the latter doesn't. You can see the difference as follows:

`\ichimoji` ㊦

`\zenkakuhabafixer` ㊦

12 License

Released under the MIT License.

13 Version History

- **v1.0.0 (2025/10/03)** — Initial public release.
- **v1.0.1–1.0.4** — Added `\ichimoji`; fixed various bugs.
- **v1.1.0 (2025/10/28)** — Unified all commands to `zenkaku` (full-width).
- **v1.1.1 (2025/11/10)** — Refined `\ichimoji` scaling logic.
- **v2.0.0 (2025/12/23)** — Overhauled scaling to match OTF character quality.
- **v2.0.1 (2026/01/08)** — With the update to `luatexja` version 20260107.0, the commands provided by the `KKsymbols` package now behave identically to native Japanese characters. This improvement is due to the bug fixes in `\ltjghostbeforejachar` and `\ltjghostafterjachar`. Previously, when multiple commands from this package were used consecutively, proper glue was not inserted between them; however, this issue has been resolved in this update.
- **v2.0.2 (2026/01/20)** — This update fixes a critical bug where arguments of the `\kakko` command could overflow when using specific fonts (for example, Hiragino fonts with weights W4 and above).
- **v2.1.0 (2026/02/16)** — In this update, the following points are changed.
 - `\period` command was deleted. It have never been used since the significance of existence had been absolutely questionable. I finally decided to delete it.
 - New package option `tsumesuji` was added.
 - New command `\zenkakuhabafixer` was added.
- **v2.1.1 (2026/02/17)** — An emergency update to fix a bug where the arguments of `\kakko`, `\maru` etc. caused expansion errors.
- **v2.1.2 (2026/02/19)** — The effects are applied when the argument of a command subject to the `tsumesuji` option—after expansion—is a numeric string of two or more digits and contains the digit “1”.
- **v2.1.4 (2026/04/17)** — When the arguments provided by this package contain only one-digit-number, the internal resize algorithm is nullified.

- **v2.2.0 (2026/06/21)** — Automatic vertical alignment was introduced. By default, the content of each enclosure is sized and centered against the current font’s reference height (x-height) instead of relying only on the glyph’s own metrics. The new commands `\kksref`, `\kksrefoff`, and `\kksrefauto` (alias `\kksrefreset`) control this behavior. For strict lowercase roman-numeral alignment, use `\kksref{\Rrnum{6}}` explicitly.
- **v2.2.1 (2026/06/21)** — Bug fix: font-size commands inside arguments no longer cause an infinite loop in the internal `one-digit/tsumesuji` check. The visible output size is unaffected.
- **v2.2.2 (2026/06/26)** — Documentation update. The documentation now describes uppercase `\Rrnum*{n}` and lowercase `\Rrnum{n}` usage, clarifies when `\kksref{\Rrnum{6}}` is needed, and adds a Japanese/English comparison with `luatexja-otf`.