Implementing Kawi

Norbert Lindenberg
Version 1, 2022-09-13

This document assists in implementing the Kawi script in fonts, font rendering systems, keyboards, and other software by providing information that complements information in *The Unicode Standard*.

This document was completed on the day Unicode 15 was released, and the OpenType Universal Shaping Engine, which is referenced several times in this document, needs final Unicode data to implement a newly encoded script. Statements about this shaping engine are therefore based on the documentation describing it and on Unicode 15 data; not on actual implementations.

Contents

1 Reference materials

The materials listed here should be consulted together with this document.

- The Unicode Consortium: *The Unicode Standard, Version 15.0.0*. The Unicode Consortium, 2022. Provides a 4-page summary of the *Proposal to encode Kawi* in section *17.9 Kawi*, the code chart for the Kawi block, and comprehensive character data.
- Aditya Bayu Perdana, Ilham Nurwansah: *Proposal to encode Kawi*. The Unicode Consortium, 2020. Provides comprehensive information about the script with numerous illustrations and references to additional documents.
- Microsoft Corporation: *Creating and supporting OpenType fonts for the Universal Shaping Engine*. Microsoft Corporation, 2020. Documents the OpenType shaping engine that should support Kawi.
- Lindenberg Software LLC: *Aksara Kawi*. Lindenberg Software LLC, 2022. Prototype application providing a font and a keyboard for Kawi for iOS and iPadOS.

2 Script identification

The ISO 15924 script code for Kawi is “Kawi”. The OpenType script tag is “kawi”.

3 Special characters

The following characters are mentioned multiple times in this document:

- U+11F42 𑼂 KAWI CONJOINER is used as the first part of the character sequences used to encode the conjunct forms of consonants and vocalic liquids. It is not intended to be used or visible by itself. The visible virama sign is separately encoded as U+11F41 𑽁 KAWI SIGN KILLER.
- U+11F02 𑼂 KAWI SIGN REPHA represents the *repha*, a vowelless consonant *r-* that starts an orthographic syllable. This document uses REPHA to denote the code point, and *repha* for the glyph.

4 Encoding order of orthographic syllable components

Kawi, like other Brahmic scripts, has features where phonetic and visual order of characters within an orthographic syllable may differ: Dependent vowels and a conjunct form to the left of the base, as well as an above-base *repha*. The script's encoding uses a primarily phonetic ordering. In addition, the conjunct forms of consonants and vocalic liquids are encoded as sequences of CONJOINER and the respective letter; such sequences should never be broken up. Characters and conjunct forms within an orthographic syllable should be encoded in the relative order shown in the following table. The *encoding* column uses the syntax defined in section A.2 Extended BNF of The Unicode Standard. The *count* column states how many characters of each class occur in real-life orthographic syllables.

Class	Characters	Encoding	Count
repha	◌	U+11F02	0 to 1
consonant, independent vowel, number, generic base	𑼰 𑼲 𑼴 𑼵 𑼶 𑼷 𑼸 𑼹 𑼺 𑼻 𑼼 𑼽 𑼾 𑼿 𑽀 𑽁 𑽂 𑽃 𑽄 𑽅 𑽆 𑽇 𑽈 𑽉 𑽊 𑽋 𑽌 𑽍 𑽎 𑽏 𑽐 𑽑 𑽒 𑽓 𑽔 𑽕 𑽖 𑽗 𑽘 𑽙 𑽚 𑽛 𑽜 𑽝 𑽞 𑽟 𑽠 𑽡 𑽢 𑽣 𑽤 𑽥 𑽦 𑽧 𑽨 𑽩 𑽪 𑽫 𑽬 𑽭 𑽮 𑽯 𑽰 𑽱 𑽲 𑽳 𑽴 𑽵 𑽶 𑽷 𑽸 𑽹 𑽺 𑽻 𑽼 𑽽 𑽾 𑽿	[U+11F12..U+11F33, U+11F04..U+11F10, U+11F50..U+11F59, U+25CC]	1
conjunct form	◌𑼰 ◌𑼲 ◌𑼴 ◌𑼵 ◌𑼶 ◌𑼷 ◌𑼸 ◌𑼹 ◌𑼺 ◌𑼻 ◌𑼼 ◌𑼽 ◌𑼾 ◌𑼿 ◌𑽀 ◌𑽁 ◌𑽂 ◌𑽃 ◌𑽄 ◌𑽅 ◌𑽆 ◌𑽇 ◌𑽈 ◌𑽉 ◌𑽊 ◌𑽋 ◌𑽌 ◌𑽍 ◌𑽎 ◌𑽏 ◌𑽐 ◌𑽑 ◌𑽒 ◌𑽓 ◌𑽔 ◌𑽕 ◌𑽖 ◌𑽗 ◌𑽘 ◌𑽙 ◌𑽚 ◌𑽛 ◌𑽜 ◌𑽝 ◌𑽞 ◌𑽟 ◌𑽠 ◌𑽡 ◌𑽢 ◌𑽣 ◌𑽤 ◌𑽥 ◌𑽦 ◌𑽧 ◌𑽨 ◌𑽩 ◌𑽪 ◌𑽫 ◌𑽬 ◌𑽭 ◌𑽮 ◌𑽯 ◌𑽰 ◌𑽱 ◌𑽲 ◌𑽳 ◌𑽴 ◌𑽵 ◌𑽶 ◌𑽷 ◌𑽸 ◌𑽹 ◌𑽺 ◌𑽻 ◌𑽼 ◌𑽽 ◌𑽾 ◌𑽿	U+11F42 [U+11F12..U+11F32, U+11F0A, U+11F0C]	0 to 3
left-side dependent vowel	◌𑼾 ◌𑼿	[U+11F3E..U+11F3F]	0 to 2
top dependent vowel	◌𑼶 ◌𑼷 ◌𑽀	[U+11F36..U+11F37, U+11F40]	0 to 1
bottom dependent vowel	◌𑼹 ◌𑼺 ◌𑼻	[U+11F38..U+11F3A]	0 to 1
right-side dependent vowel, vowel killer	◌𑼰 ◌𑼱 ◌𑽁	[U+11F34..U+11F35, U+11F41]	0 to 1
bindu	◌𑼵 ◌𑼶	[U+11F00..U+11F01]	0 to 1

Class	Characters	Encoding	Count
visarga	◌ꦃ	U+11F03	0 to 1

In some late Kawi varieties, the *repha* glyph may be used for a final *-r* consonant. This does not affect the encoded representation, which keeps treating REPHA as the start of the orthographic syllable.

The encoding order shown above is the one resulting from the OpenType Universal Shaping Engine's default interpretation of the Unicode character data for Kawi characters, so no overrides are needed in that shaping engine. The engine inserts dotted circles into character sequences with out-of-order marks, so that OpenType fonts only need to deal with correctly ordered characters. Fonts based on technologies other than OpenType, such as Apple Advanced Typography or Graphite, should themselves insert dotted circles into character sequences with out-of-order marks.

Keyboards and other text-generating software should ensure that typed or generated text conforms to this encoding order. Spelling checkers and other text-validating software should use this encoding order in their reference data.

5 Rendering

The following steps must be taken, in the order given, for each orthographic syllable to achieve orthographically correct rendering:

1. Combine each CONJOINER with the following consonant or vocalic liquid to a conjunct form.

   ◌꧀ + ꦩ → ◌ ꦩ꧀ *-ka*

   ◌꧀ + ꦉ → ◌ꦉ꧀ *-r*

2. Move the *repha*, if present, after the base, and treat it as a nonspacing combining mark.

   ꦱ + ꦩ → ꦩ + ◌ꦿ → ꦩꦿ *rka*

3. Move all pre-base vowels (◌ꦇ and ◌ꦈ) before the base.

𑼒 + ◌𑼾 → 𑼒𑼾 *ke*

4. Convert any *ra* conjunct form to its below-base variant if necessary to prevent collisions with other glyphs.

   𑼪 + ◌𑽂𑼮 + ◌𑽂𑼬 → 𑼪 + ◌𑽂𑼮 + ◌𑽂𑼬 *mwra*

5. Move any *ra* conjunct form that is not converted to its below-base variant before the base (but after pre-base vowels).

   𑼒 + ◌𑽂𑼬 → 𑼒𑽂𑼬 *kra*

   𑼒 + ◌𑽂𑼬 + ◌𑼾 → 𑼒𑼾 + ◌𑽂𑼬 → 𑼒𑽂𑼬𑼾 *kre*

For OpenType fonts using the Universal Shaping Engine, step 1 needs to be implemented in the font; steps 2 and 3 should be implemented by the shaping engine based on Unicode character data; step 4 needs to be implemented in the font; and step 5 can be triggered by applying the “pref” feature to the *ra* conjunct form, which should cause the shaping engine to reorder the glyph.

Several additional steps may be taken to improve typography and to support stylistic preferences:

- Glyphs that sit below below-base glyphs may be reduced in height, and below-base vowels may be attached at a higher-than-usual position, to reduce overall line height.

  𑼥 + ◌𑽂𑼡 + ◌𑽂𑼭 + ◌𑼹 → 𑼥𑽂𑼡𑽂𑼭𑼹 *ntlū*

  𑼥 + ◌𑽂𑼡 + ◌𑽂𑼬 + ◌𑽂𑼫 → 𑼥𑽂𑼡𑽂𑼬𑽂𑼫 *ntrya*

- Above-base marks are commonly not stacked vertically but ligated or displayed slightly offset. When determining the set of supported combinations, it is reasonable to assume that at most one of the vowels U+11F36 ◌𑼶 KAWI VOWEL SIGN I, U+11F37 ◌𑼷 KAWI VOWEL SIGN II, or U+11F40 ◌𑽀 KAWI VOWEL SIGN EU occurs.

  ◌𑼂 + ◌𑼶 → ◌𑼂𑼶 *r-i*

  ◌𑼶 + ◌𑼁 → ◌𑼶𑼁 *-iŋ*

- Above-base marks are positioned above base glyphs or right-side conjunct forms, not above right-side vowels. In OpenType, this means that right-side vowels must be treated as marks, so that they can be ignored when positioning the above-base marks. Because of a compatibility feature in the Universal Shaping Engine, this causes their width to be set to 0, so that it must be added back later using the “dist” feature.

  𑼐 + ◌𑼴 + ◌𑼀 → 𑼐𑼴𑼀 *om*

- *Repha* and U+11F35 ◌𑼵 KAWI VOWEL SIGN ALTERNATE AA may be ligated.

  ◌𑼂 + ◌𑼵 → ◌𑼂𑼵 *r-ā*

- Fonts may enable the selection of stylistic glyph variants via features such as stylistic sets.

  ◌𑽂𑼲 → ◌𑽂𑼲 *-ha*

  ◌𑽂𑼰 → ◌𑽂𑼰 *-ṣa*

  ◌𑼵 → ◌𑼵 *-ā*

- Orthographic syllables often need to be spaced apart to avoid collisions between below-base glyphs.

  𑼒𑼺𑼰𑽂𑼠𑼦𑼒𑽂𑼰 → 𑼒𑼺𑼰𑽂𑼠𑼦𑼒𑽂𑼰 *kṛṣṇapakṣa*

Ligatures versus glyph positioning. The Kawi script has some glyphs that may, depending on the design of a typeface, connect to each other. For example, the dependent vowel ◌𑼸 *u* may connect to the right-most stems of many base consonants, as in 𑼒𑼸 *ku*. For best rendering results, all such combinations of connecting glyphs would be implemented as ligatures. However, this might substantially increase the size of a font. The alternative is to not include such ligatures, or only the most commonly used ones, and instead rely on positioning the connecting glyphs appropriately. This may however result in slight offsets in connecting lines, as renderers usually first rasterize the glyphs separately, then position them relative to each other, and both steps involve rounding of coordinates. Font developers should consider the trade-off carefully.

6 Keyboards

Key issues in the development of keyboards include which characters to support, how to arrange them on a physical keyboard or on screen, how to let the user interact with the keyboard, whether and how to support predictive input, and how to ensure the correct encoding order of orthographic syllable components. This section discusses some of these issues, using the on-screen keyboard in the Aksara Kawi prototype app of Lindenberg Software as a reference.

Supported characters. When selecting characters to support, a common practice is to select only the characters needed for a given language, not all characters in a script. However, a language-based approach requires identifying the character set used for each language and then creating separate keyboards for each. As the currently encoded Kawi character set is not that large, the app's keyboard provides the complete character set.

In addition to encoded characters, developers need to consider how to support conjunct forms. In Kawi, conjunct forms are encoded as sequences of CONJOINER, which is not intended to be displayed by itself, and a consonant or vocalic liquid. If possible, keyboards should have dedicated keys for conjunct forms so that users do not have to enter CONJOINER separately. The Aksara Kawi app provides a separate layer with conjunct forms, which is shown below.

Keyboard layout. There is no standard keyboard layout for Kawi yet. The Aksara Kawi prototype app takes advantage of the fact that on-screen keyboards can show the keys of all layers, and uses two layers for base characters and one for conjunct forms, each with five rows.

Encoding order of orthographic syllable components. The keyboard of the Aksara Kawi app relieves users of having to understand the encoding order of orthographic syllable components by automatically reordering characters within an orthographic syllable. It assumes that a consonant without preceding CONJOINER, or a repha followed by such a consonant start a new orthographic syllable. Other syllable components can be entered in arbitrary order and will be reordered. This is possible because the keyboard API on iOS lets keyboards read and edit the text surrounding the current insertion point. The input method APIs on Android, macOS, and Windows provide similar capabilities. Smart keyboards should take advantage of them to help users avoid incorrect text and dotted circles.

7 Line breaking

Lines of Kawi text can be broken at any orthographic syllable boundary. The Unicode Line Breaking Algorithm and implementations based on it, such as the Internationalization Classes for Unicode (ICU) library, do not yet provide this style of line breaking. Instead, two errors commonly occur: Either lines are broken only at punctuation, resulting in text overflowing the space available to it, or lines are broken at Unicode grapheme cluster boundaries, resulting in broken conjunct forms, as CONJOINER is treated as the end of a grapheme cluster.

A proposal to correct the Unicode Line Breaking Algorithm has been submitted to the Unicode Technical Committee, which has requested that this proposal be implemented in ICU. Owners of other implementations should update them based on the proposal.

8 Acknowledgments

I would like to thank Aditya Bayu Perdana and Ilham Nurwansah, who provided a detailed description of Kawi in their encoding proposal. Bayu also designed the font that is used in the proposal, in the Unicode Standard, in this document, and in the Aksara Kawi app. Finally, I would like to thank the reviewers of the peer document on Javanese, whose comments helped shape this document as well.