Khmer Encoding Structure

Martin Hosken (SIL International, NPIC LSDU)
Contributors: Norbert Lindenberg, Makara Sok

Date: September 2024

Table of Contents

Introduction....3
Rationale....4
Document Conventions....5
Scope....6
The Starting Point....7
Orthographic Syllable Structure....9
Coengs....9
Triisap and Muusikatoan....10
Vowels....11
Signs....12
Miscellaneous....14
Conclusion....18
Consonant Shifters....22
Downshifting....23
The Main Regular Expression....26
Ordering....28
Confusables and Undesirables....30
Deprecated or discouraged....30
Canonical equivalence....30
Do not use....31
Confusables....32
Coeng ta and *Coeng da*....35
Yuukaleapintu and Colon....35
Transition....36
Compatibility....37
Data Transition....38
Normalization....38
Shaping and Font Development....43
Shaping....44
Tests....46
Design Issues....46
Acknowledgements....48
Appendix 1 - Current Confusion....49

- Example Words.....49
- Coeng Ta vs Coeng Da.....52

Appendix 2 - Middle Khmer.....53

- Final Coengs.....53
- Multiple Vowels.....54
- Diacritics.....55
- More Examples.....56
- Khom Thai.....57
- Conclusion.....57

Bibliography.....58

Introduction

The Khmer script has been encoded in Unicode since September 1999. But it is a very complex script used to write several languages. The primary driving orthography is for the modern Khmer language and there is some confusion in understanding how to encode it. This document is a step in a long history of the discussion of how to encode the Khmer script. It proposes no changes to the character repertoire of the Unicode Standard, but concentrates on the description of what a Khmer sequence consists of and what rules are to be applied to it.

One of the primary concerns with this revision of the encoding structure is to ensure that **one visual form has only one encoding**. That is, there are not two ways to encode the same visual representation. This is important for three reasons:

1. Confusability. If you can encode the same visual representation in two ways, you can come up with, for example, two different website addresses that look identical, thus allowing someone to spoof another site.
2. Ease of typing. If there are two ways to represent a single visual form, then it is up to the typist to ensure that they enter text in an order acceptable to fonts, spelling checkers, and other text processors, for any given visual form. If, on the other hand, there is only one encoding possible, then the computer can generate that encoding based on a visual input from the typists, regardless of how badly they type that visual form. This means that those who are not experts in Khmer Unicode can successfully enter good data based exclusively on the visual form.
3. Sorting and searching. If there are two ways to encode the same visual representation, people will be confused as to why two identical looking words do not match each other, or sort to different places or why one might be marked as misspelled for no visual reason or why some words cannot be found in a search.

Therefore the current encoding results in significant user experience difficulties (see Appendix 1). Thus while, if one is careful, one can enter Khmer text in a manually enforced consistent fashion with correct rendering, most users are not proficient enough to ensure such consistency of data entry.

When considering the encoded syllable structure for a script there are two extremes that can be created:

- The loose specification aims to allow any possible sequence so long as it is visually contrastive to all other sequences, even if such a sequence could never occur in any orthography using the script. For example, in Khmer, this would allow multiple vowels or multiple non spacing diacritics. The cost with this approach is that implementers then need to support these extra sequences. For example, font developers would need to support vertical stacking diacritics for vowels and other diacritics for Khmer.
- The tight specification aims to only describe sequences that could occur in an orthography. The danger here is that it may limit someone who needs to use the script in unexpected ways for a new orthography or the transliteration of foreign words.

In this discussion, we will start from a relatively tight position of building the syllabic structure from orthographic analysis and then loosen it according to recognised needed extensions. Such an analysis cannot be completed without consideration of all orthographies that use Khmer script, including Old and Middle Khmer. In this discussion, ‘Khmer language’ refers to Modern Khmer. Other Khmer language orthographies (like Middle or Old Khmer) are explicitly identified. It is worth noting that modern minority language orthographies have been designed as much as possible, to fit within the structural norms of Modern Khmer. For example, consonant shifters exhibit the same downshifting behaviour found in Modern Khmer.

In handling what this document describes as illegal sequences, font and shaping engine developers need to ensure that their rendering differs from that of correct sequences. This may be done by inserting dotted circles into incorrect sequences of signs, or by stacking non-spacing marks such that the differences become visible to the user. In OpenType, where shaping engine developers and font developers share the responsibility for this, shaping engine documentation must specify which cases are handled by the shaping engine and which ones remain as font developer responsibilities.

Rationale

Encoding issues are something that end users of applications, working with Khmer script, should never have to consider. Systems should do the right thing for them, and they should not even need to know what Unicode is. That users need to know about Unicode in order to type is a failing on the part of the system implementers. This is not a criticism of current system implementers because implementation is hard and has lacked the necessary supporting standards. Users should be able to type in a way that is natural to them. They should be able to search for a word and find all the instances of that word across multiple documents, typed by different people with different applications (assuming such a search can deal with the different file formats appropriately).

The current situation, for Khmer, is that users are expected to have a relatively deep understanding of how Khmer is encoded in Unicode in order to type correctly. They are expected to type their coengs, vowels and other diacritics and signs in the right order with nothing to help them visually. It is not a surprise therefore that there can be a plethora of ways in which a single word is spelled (see Appendix 1).

It is assumed, in this document, that the implementation of support for Khmer will include everything needed to hide the encoding complexities from users. There is no need to consider the impact that encoding will have on how end users type. Instead the question is of the impact the encoding has on the implementation of input methods that enable users to type in a natural way without consideration of encoding issues.

> *One of the purposes of this document, therefore, is to help implementers create solutions that allow users to not have to know about Unicode to be able to type their language correctly.*

We hope that this document will lead to others that will enable font designers and keyboard implementers to produce fonts and tools that work consistently and enable consistent data entry and rendering. We do this by concentrating on describing an unambiguous encoding structure for Khmer.

One important presupposition of this document is that if there is only one way to encode something, then it makes it easier to produce a system that works with that one way. Thus if there is only one way to store a word, an input method can take a variety of ways that a user might type a word and normalise them into the single correct way. On the other hand, if there are multiple ways to encode a word, then the input method cannot do this and the user is expected to resolve the ambiguity and pick the right ordering. Which, in turn, requires them to know what is the correct ordering and, as will become evident, this involves every typist in the country having a relatively deep knowledge of Unicode encoding.

In the technical context where a keyboard only allows a user to type single codepoints (or short sequences), there has been no other option. Users want to be able to type in a variety of orders and therefore have made use of the existing ambiguities in the encoding. But modern keyboard applications are far more sophisticated and are capable of allowing different typing orders and outputting a single correct order. But they can only do this if there is an agreed single correct order to output. If there is no expected order, then the keyboard cannot resolve its input and the decision of what output to generate has to be pushed back onto the users. Users therefore have to be aware of encoding issues. This is the problem this document attempts to resolve: **What is the agreed single correct order to be used?**

The descriptive approach taken in this document is that of a regular expression. Some may wonder why the current canonical combining order mechanism in Unicode is insufficient to meet this need. Leaving aside the issues of Brahmic scripts and Unicode normalization, one unchangeable rule is that once the combining order for a character is set, when it is added to the standard, it can *never* be changed. Even if the canonical combining order mechanism were sufficient to resolve issues of this complexity (which it is not and will be evidently so in reading this document), the combining orders for the Khmer characters are set and can never change. Given text is often normalized according to canonical combining order, we have to ensure that the structure here does not disagree with that normalization process.

Khmer gets off lightly with regard to non zero canonical combining classes (CCC). The only two characters with non zero CCC values are U+17D2 (Coeng) CCC=9 and U+17DD (Atthacan) CCC=230. Since, in this encoding structure U+17D2 and U+17DD are never adjacent, there is no problem. In the current Unicode standard, which allows for coengs after vowels and diacritics, this can be a problem with a diacritic being reordered to between a coeng and its following base. Canonical ordering, therefore, is more of a problem than a solution. It is insufficiently nuanced to handle the complexity of structure needed and too often the setting of values by those unaware of the complexities can break a script. Thankfully, this is not the case for Khmer, but it is a potential minefield that has to be avoided during the development of the structure.

Document Conventions

The approach taken in describing the encoding, will be strongly regular expression based both in terms of describing the structure but also in describing transformations for rendering and keyboarding. The regular expression language used is based on PCRE[1] and EBNF[2] . In particular the following more advanced aspects of regular expressions and EBNFs are used:

- Alternation. Sequences are kept together between `|` alternation marks within a group.
- A separately defined subexpression, which is presented as a non-terminal declaration and use, is considered its own group.
- `(?=`*regex*`)` is a zero width look ahead assertion and `(?<=`*regex*`)` is a zero width look behind assertion. A sub expression is tested at the point in the sequence that the matcher as got to, either looking forwards or backwards.This allows the identification of a specific subexpression in the context of text before and after. This is used in rewrite rules where the matched expression is replaced, leaving the context unchanged.
- `(?!`*regex*`)` is a negative look ahead assertion (fail if the regex matches) and `(?<!`*regex*`)` is a negative look behind assertion.
- From standard regular expressions we use `?` for optional and `{0,2}` for ‘up to two’. Parentheses are used to group sequences and alternations. `[]` match one of the characters in the list.

Look behind assertions are problematic in that many regular expression engines that support them, require them to be of fixed match length. A positive look behind assertion can often be refactored into a match and replace, but negative look behind assertions can be more difficult to refactor.

There are various technical terms that are often used very ambiguously. This document is not immune to such ambiguities. But it is hoped that:

[1] Perl Compatible Regular Expressions (https://www.pcre.org)
[2] Extended Backus Naur Form. Used here for its higher level structure of named non terminals.

- A **diacritic** is a nonspacing chracter, above or below a letter. This is in contrast to general Khmer linguistics. *Thus for this document the following are also diacritics: nonspacing coengs and nonspacing vowels.*
- A **symbol** is a character that is not part of a phonetic syllable. For example, digits, punctuation.
- A **sign** is a character that is not a symbol, consonant, coeng, independent vowel or vowel. Most of the signs are diacritics, but some are spacing. Signs may not occur on their own and are part of the syllable structure. This is more commonly called diacritics in Khmer linguistics.
- The term **spacing**, for example when talking about spacing **signs**, vowels and coengs, is restricted in this document from here on to mean only right spacing, unless otherwise specified. Thus coeng ro (`17D2 179A`) and pre-base vowels [`17BE 17C1..17C3`] are not considered spacing. Split vowels with both pre-base and post-base components [`17BF 17C0 17C4 17C5`] are considered spacing.
- The **syllables** discussed in this document are orthographic syllables, unless specifically called phonetic syllables.

Scope

The Khmer orthography is used for more than just modern Khmer. It is used for a number of languages. The current list of languages that use the Khmer script as their primary script is: Brao (brb), (Modern Khmer) (km), Krung (krr), Kavet (krv), Northern Khmer (kxm), Old Khmer (okz), Pear (pcb), Kraol (rka), Somray (smu), Tampuan (tpu), Middle Khmer (xhm). Other languages that have Khmer script based orthographies include: Bunong (cmo), Jarai (jra), Pali (pi), and Sanskrit (sa). Makara Sok (2021) provides information on the specific use of the Khmer script for several of these languages. Included in this list are Modern Khmer (km) and Middle Khmer (xhm). In addition to the need of supporting dictionary words in a language, people can sometimes spell their names in novel ways and systems supporting the script need to support those novelties. Here is a list of names that will be examined again as a final check of the efficacy of the encoding.

ឃុី ណ៊ិប រ៉មាីអះ កន្តិ

The names listed show various ways in which the encoding must support more sequences than are merely required for the dictionary. Again, any such extended support needs to ensure that for any given visual representation there is only one valid encoding.

Those readers interested in only the core Khmer encoding structure and its descriptions need only read this introduction and the following “Orthographic Syllable” section. The later sections are mainly informative or support for the arguments made in the Orthographic Syllable section.

The later section on “Consonant Shifters” explains the derivation of the consonant shifter rules. The appendix on “Middle Khmer” analyses samples of Middle Khmer words to consider how well the encoding structure supports Middle Khmer and whether there are extensions that might be needed for Middle Khmer. The section on “Confusability” looks at what tools like Keyboards need to do to address character sequences that, while different and having a different visual representation, result in visual forms that are so close to each other as to be confusible. The section on “Transition” examines what needs to be done to transition data from existing Unicode encoded representations into this new encoded representation. This section includes a sample reference implementation of both a normalizer that converts text and a tester that checks that strings conform to the specification here. The section on “Shaping and Font Development” considers font design issues arising from the encoding. A large proportion of the burden of ensuring that different strings look different falls on the font developer. There is also an appendix that lists sample problem words and how they are often misspelled.

The Starting Point

Over the years, there have been a number of attempts at resolving the issues in the Khmer encoding and various orthographic syllable structures have been proposed. The result has been a number of different encoding structures, all vying to be the one true structure:

```
SylU =  Base (Robat | Shifter)? (Coeng Robat?)* (ZW? Vowel)? (Modifier | Final)?
Coeng?

SylMS = Base Coeng{0,2} (PreVowel | BelowVowel)? Shifter? AboveVowel? Modifier{0,2}
       FollowingVowel? Final?

SylH = Base ZW? (Shifter | Robat)? (Coeng (ZW? (Shifter | Robat))?)* ((ZW Modifier)*
       PreVowel? (ZW Modifier)* BelowVowel? (ZW Modifier)* (ZW? AboveVowel)?
                 (ZW Modifier)* FollowingVowel? (ZW Modifier)* Coeng? Final* | 17D2)

SylE =  Base (Robat | (ZWNJ? Shifter)) Coeng* (ZWNJ? Shifter)?
       (ZWNJ? Vowel)? (Modifier | Final)? (ZWJ Coeng)?
```

```
Base                               # Base consonant or independent vowel
Coeng                              # Subscript consonant or independent vowel sign
Final                              # Modifier Final
Modifier                           # Modifier Sign
Vowel                              # Dependent vowel sign
PreVowel                           # Pre-base vowel sign
BelowVowel                         # Below-base vowel sign
AboveVowel                         # Above-base vowel sign
FollowingVowel                     # Post-base vowel sign
Other                              # Other signs
ZWNJ = 200C                        # zero width non joiner
ZW = [200C 200D]                   # ZW(N)J
Robat = 17CC                       # Robat
Shifter = [17C9 17CA]              # Shifter Character
```

SylU is the current Unicode standard (v14). SylMS is the Microsoft Khmer shaper specification (as of 2022). SylH is the Harfbuzz Khmer shaper implementation for the “Khmr” script tag. SylE is an adapted structure from an unpublished proposal from Javier Solá [2004] to refine the Khmer syllable structure. Complete definitions of some key terminals (classes) are not given since they vary between encoding structure definitions. But their informal description is sufficient for this analysis.

While there is considerable variation, there is some commonality and it is important that any encoding structure derived should embrace that commonality. For example, Robat occurring early in the sequence, while not universal, is popular. The main difficulty of these encoding structures is that they allow multiple ways to encode the same visual form. In addition, proposed syllable structures, actual font implementations, and shaping engines diverge from each other in various ways (e.g. Lindenberg 2019). We will, therefore, build up a new regular expression, but in the same direction. Due to the complexities of Middle Khmer, these will not be considered initially, although they may be mentioned, for this initial development of an encoded orthographic syllable structure. Current minority language orthographies will be considered where appropriate.

All analysis of Khmer encoding so far has concentrated on the rendering of correctly input text with a strong linguistic motivation. Unfortunately, many Khmer writers have insufficient expertise in the linguistics of the language, let alone Unicode, to be sure to enter correctly encoded text. Ambiguities in the encoded syllable mean that it is impossible for tooling to help users to type well. If we can remove the visual ambiguities from the structure, it is then possible for tooling to enable users to type in any order or resolve common typing errors. It also removes confusability (where two different sequences render identically) which is rife in Khmer encoded data.

Orthographic Syllable Structure

At its simplest, a Khmer orthographic syllable may consist of a single consonant or independent vowel. This is the only required element of an orthographic syllable.

```
Syllable = Base | IndependentVowel
Base = [1780-17A2]
IndependentVowel = [17A5-17B3]
```

The characters 17A3 and 17A4 are excluded since they are deprecated.
We can treat independent vowels just like base consonants. While they have different linguistics associated with them, visually they behave as bases. For example, it makes no linguistic sense for an independent vowel to have another vowel diacritic. But this is purely a spelling issue, there is no visual contrast that needs to be resolved. This also holds for all other diacritics and signs. Thus:

```
Syllable = Base
Base = [1780-17A2 17A5-17B3]
```

For the purposes of the syllable structure, there are also other characters that we treat as being a syllable in their own right. They take no diacritics and occur, simply, on their own. In addition we add all the deprecated characters because they should not be used and therefore we do not want to see them in the main syllable structure.

```
Other = [17A3 17A4 17B4 17B5 17D3-17DC 17E0-17E9 17F0-17F9 19E0-19FF]
```

A Khmer syllable may also include a single dependent vowel, unlike other scripts of Southeast Asia, which allow longer vowel sequences:

```
Syllable = Base Vowel?
Vowel = [17B6-17C5]
```

Coengs

Khmer has subjoined consonants, called “coengs” in Khmer, that are used for initial consonant clusters (sequence of consonants) and also for syllable chaining (whereby the base consonant is read as the final consonant of the previous phonetic syllable and the coeng as the initial consonants of the next phonetic syllable). In Middle Khmer they can even represent the final consonants of phonetic syllables. These uses are not contrasted. For example, Khmer allows for syllable chaining where final consonant from a previous phonetic syllable that is chained visually with the initial consonant of the next phonetic syllable as the initial of the next orthographic syllable rendered underneath. There is a linguistic syllable boundary in the middle of a grapheme cluster. In orthographic syllable terms, we treat the final of the previous linguistic syllable as the initial of the following orthographic syllable and the initial of the following linguistic syllable as an orthographic medial consonant. For example, សង្គ្រាម (179F 1784 17D2 1782 17D2 179A 17B6 1798) /sang.kriam/ is made up of two phonetic syllables but three orthographic syllables, each starting with a base character (ស ង ម). The second base character (ង) in this example is acting as the final consonant of the first phonetic syllable.
There may be up to two coengs in an orthographic syllable, (e.g. involving coeng ro or as result of syllable chaining). While theoretically more could occur, two pre-vowel coengs is the limit used in any known orthography. The cost, therefore, of requiring support for more, given they are never used, outweighs the value of increased generality. In Unicode a coeng is made up of a Khmer sign coeng (17D2) followed by a base consonant or independent vowel.

```
Syllable = Base Coeng{0,2} Vowel?
Coeng = 17D2 Base
```

With regard to the ordering of coengs, all coengs have a component that renders under the base or a previous coeng. *Therefore coengs are considered as being ordered downwards from the base character regardless of whether they are spacing.* Fonts need to ensure that contrastive order results in contrastive rendering.

Coeng Ro ◌្រ

Coeng ro (17D2 179A) is special because it is the only coeng that is rendered before the base consonant. It also has a hook component that goes under the following consonant. Often this hook is subtle and does not visually interact with any other coeng under the consonant, particularly if that coeng is spacing. This means that there can be two orders of storing the coeng but the same visual result. This is a common source of typing errors, since few Khmer users use a smart keyboard that could reorder as needed. To resolve this, it is best to choose a fixed order for coeng ro in relation to other coengs. In Modern Khmer, there are no occurrences of multiple coengs in which the coeng ro comes first. We can establish a rule that in a sequence of two initial coengs, the coeng ro will always be stored second.

In Tampuan, coeng ro (17D2 179A) always comes, phonetically and so visually[3], before coeng vo (17D2 179C), although it comes after any other coeng. Thus we say that a font may use language specific styling for Tampuan to do the visual reordering, if coeng ro and coeng vo interact visually.

```
Syllable = Base Coengs? V?
Coengs =  ((Coeng NonRo)? Coeng Base)

Coeng = 17D2
NonRo = [1780-1799 179B-17A2 17A5-7B3]          # All bases except Ro
```

Triisap and Muusikatoan

The consonant shifters, triisap (17CA) and muusikatoan (17C9), are such important signs in Khmer that they require their own very careful analysis. There are contextual shaping issues, keyboarding issues and encoding issues with these signs. The full derivative details of consonant shifters are described in the section on Consonant Shifters.

In the Khmer orthography, the series of a consonant cluster affects the phonetics of the vowel following it. A consonant shifter is used to change the series of a consonant clauster. Therefore, it is the orthographic consonant cluster that is interpreted as having a series, even if its constituent consonants are from different series. Where in a sequence should the consonant shifter be stored? There are two schools of thought:

- The consonant shifter immediately follows the consonant it affects. Thus if the shifter changes the series of the base consonant, it is stored after the base and before a possible coeng. Likewise if it affects the coeng, it follows that coeng. The problem is that if the coeng is not spacing, it requires considerable linguistic awareness on the part of the typist, to decide whether the shifter affects the base or coeng and in some cases not even this is sufficient to resolve the ambiguity.
- The consonant shifter is stored at the end of the consonant cluster, since it applies to the whole cluster.

Since in all orthographies we know about, there is no semantic difference between a shifter before or after a spacing coeng, it is preferable to give the shifter a fixed position in the sequence. Thus we propose the shifter is stored at the end of the consonant cluster. (ស្ម៊ី)

[3] In fonts where the two coengs visually interact, the presentation is such that it is evident that coeng ro comes before coeng vo.

The later section on Consonant Shifters examines the whole issue of how in some contexts a consonant shifter changes to the glyph of a -u vowel (17BB). The only structural implications of that section is that it is sometimes necessary to override the default shaping behaviour and to not allow a consonant shifter to ‘down-shift’. To do this we use the zero width non-joiner ZWNJ (200C) to stop a shifter from down-shifting. This is already common practise in Khmer text. This affects the syllable description:

```
Syllable = B Coengs Shifter?
Shifter = ([17C9 17CA] ZWNJ?)
ZWNJ = 200C
```

Simply adding a ZWNJ into the syllable description also introduces an ambiguity. Where a consonant shifter cannot be downshifted (because there is no upper diacritic following), then the ZWNJ is redundant and a string with or without the ZWNJ, by definition, will render the same. To complicate things further, a simplistic algorithm that says any shifter followed by an appropriate above diacritic shifts down, results in ambiguity, since either consonant shifter will result in the same visual representation. Thus, while we can allow any consonant shifter after a consonant cluster, only those that might result in down shifting may take a ZWNJ. The constraints are complicated and we copy them here from the discussion and analysis in the section on Consonant Shifters.

```
Shifter = (  (?<= StrongContext AnyFinalCoeng?) 17CA ZWNJ (?=AboveVowel)
           | (?<! StrongContext AnyFinalCoeng?) 17C9 ZWNJ (?=AboveVowelSamyok)
           | [17C9 17CA])

StrongContext = Series1Base Robat? CoengNoBA{0,2}
      | NoBA Robat? (Series1Coeng CoengNoBA? | CoengNoBA Series1Coeng)
      # Contains a strong consonant and no BA

AnyFinalCoeng = Coeng ZWJ NonRo
Robat = 17CC                              # Robat [ ៌]
Series1Base = [1780-1783 1785-1788 178A-178D 178F-1792 1795-1797 179E-17A0 17A2] #
Strong consonants
Series1Coeng = (17D2 Series1Base)         # Strong coengs
CoengNoBA = (17D2 NoBA)                   # Coeng no BA
NoBA = [1780-1793 1795-17A2]              # Series consonant with no BA
AboveVowel = [17C1-17C5]?[17B7-17BA 17BE 17DD] | 17B6 17C6     # Above vowel context
AboveVowelSamyok = AboveVowel | [17C1-17C3]? 17D0       # Above vowel or samyok sannya
ZWNJ = 200C                               # Zero width non-joiner
```

Notice that the definition for VA is looser than the vowel specification. For example, it allows 17C1 17B8, which the vowel specification does not. This is because if vowel specification does not allow a sequence, then it will not occur and whether or not VA would match it makes no difference.

Vowels

Single vowels have already been discussed and these are sufficient for Modern Khmer. But Middle Khmer and some names require the ability to encode vowel sequences.

Multiple Vowels

A natural approach to supporting multiple vowels is to consider there to be 4 positions (before, above, below, after) in which vowel components may occur in relation to the consonant cluster. Since Khmer typically uses only a single vowel, any such sequences are rare and the need for more than 2 is unknown. Therefore there is no need to burden font developers with needing to support diacritic vowel stacking. To keep rendering simple and ensure visual contrast, we can allow vowel components in any combination of positions. The only encoding requirement is that vowels are stored in an appropriate order. Unfortunately there are also certain vowel sequences that may not occur because they either are confusable with an existing combined vowel, or as in the case of 17BB are the representation of a downshifted consonant shifter. Below vowels do not occur with the special 17BF and 17C0 compound vowels.

One way of describing the vowel sequence would be:

```
Vowels = VPre? VBelow? VAbove? VPost? | VDeprecated
VPre = [17BE-17C5]
VBelow = [17BB-17BD]
VAbove = [17B7-17BA]
VPost = 17B6
VDeprecated = [17B4-17B5]
```

But there are exclusions:

```
NonVowelSequences = 17BB VAbove | 17BB 17B6 17C6 | 17BB [17D0 17DD]
    | 17C1 VBelow? 17B8 | 17C1 VBelow? VAbove? 17B6
    | 17BE 17BB
    | [17BE-17C0] VBelow? VAbove | [17BF 17C0] VBelow | [17BF 17C0] 17B6
    | [17C4 17C5] VBelow? VAbove? 17B6 | 17C4 VBelow? 17B8
```

Alternatively, if this is then converted into a single positive regular expression we get:

```
Vowels = 17C1 [17BC 17BD]? [17B7 17B9-17BA]?      # VPre VBelow? VAbove?
  | [17C2-17C3]? [17BC-17BD]? [17B7-17BA]? 17B6  # VPre? VBelow? VAbove? VPost
  | [17C2-17C3]? 17BB 17B6? (?! [17D0 17DD] | 17C6)      # VPre? VBelow? VPost
  | 17BE [17BC-17BD]? 17B6?                      # 17BE VBelow? 17B6?
  | [17C1 17C4 17C5]? 17BB (?! [17D0 17DD])      # VPre? 17BB Modifier?
  | [17BF 17C0]                                  # Standalone VPre
  | [17C2 17C3 17C5]? [17BC-17BD]? [17B7-17BA]?  # VPre? VBelow? VAbove?
  | 17C4 [17BC-17BD]? [17B7 17B9 17BA]?[4]        # VPre but don't clash with 17BE 17B6
```

Note that the last line of the definition must be the last line, since it may be zero length, not matching anything. This makes vowels inherently optional.

Signs

In Modern Khmer script, there can be up to two non-spacing signs and / or one spacing sign. We categorise the non-spacing sign as a modifying sign and the final spacing sign as a modifying final. This gives flexibility to categorise non-spacing signs as modifying finals if that is needed (see Middle Khmer). Robat, while technically a sign, is in its own category and position, as discussed below.

```
Syllable = Base Robat? Coengs? Shifter? Vowels Modifier{0,2} Final?

Modifier = [17C6 17CB 17CD-17D1 17DD]      # Modifying Signs [ំ ់ ៍៎៏័៑៝ ]
Final = [17C7 17C8]                        # Modifying Finals [ ◌ះ ◌ៈ]
Robat = 17CC                               # Robat [៌ ]
Other = [17D4-17DC 17E0-17F9 19E0-19FF]
              # Other: Symbols and digits [ ។ ៕ ៖ៗ៘៙៚៛ៜ ៰៱៲៳៴៵៶៷៸៹ ᧠-᧿]
```

Since some diacritic signs may occur with other diacritic signs, we generalise the syllable structure to allow two diacritic signs. The specifics are given for each sign.

U+17CC Robat ◌៌

Robat represents a final r in the previous phonetic syllable, like repha in Devanagari. A consonant followed by a robat is equivalent to the sequence of a ro followed by the consonant as a coeng. For

[4] 17C4 17B8 is visually identical to 17BE 17B6. We choose the latter for legacy shaping support.

example ko (1780) robat (17CC) is equivalent to ro (179A) coeng ko (17D2 1780). In Modern Khmer, robat does not co-occur with coengs or with vowels other than 17B6 (◌ា) and 17BB (◌ុ), and is rarely sounded.

Because of the equivalence described above, we propose that robat is stored immediately after the base consonant, and that there can be only one per orthographic syllable. This differs from the syllable structure currently specified in the Unicode Standard, which allows robat to both occur immediately after the base and to follow coengs, and allows multiple robats. For Modern Khmer, this change makes no difference, as robat never co-occurs with coengs. For Middle Khmer, it may result in an incompatibility with text written with HarfBuzz, the only shaping system that has so far allowed robat to follow coengs.

U+17C6 Nikahit ◌ំ

Nikahit marks final nasalisation and, in the Khmer language, occurs in three contexts: on its own, after -u (17BB) and after -aa (17B6). In conjunction with -aa, it pushes a consonant shifter down. May be followed by kakabat or toandakhiat and also, in Tampuan, samyok sannya.

U+17CB Bantoc ◌់

Bantoc is placed over a final consonant to shorten or change the nature of the vowel in the phonetic syllable (including the inherent vowel) that ends with that consonant. In Modern Khmer, it only occurs after a simple base consonant. In Middle Khmer, it may also follow -aa 17B6 where it renders over the consonant (ព្យីភា់ល).

U+17CD Toandakhiat ◌៍

This is a silencer and may occur after any vowel (except 17B6) or final consonant cluster. It is a rare character that, in the Khmer language, most often occurs alone or after 17B7 (◌ិ), with which it visually ligates.

U+17CE Kakabat ◌៎

Kakabat marks exclamation in Khmer. It renders above any vowel. The presence of kakabat does not cause the shifter to go down. (See the later section on Consonant Shifters). In an extreme case there may be a consonant shifter, vowel and kakabat all stacked up: អ៉ឺ៎ះ. (It can also occur after samyok sannya when transcribing Thai tones.)

U+17CF Ahsda ◌៏

This mark highlights certain consonants as being a discrete word. It may occur after 17C5 (◌ៅ) in which case it renders centred over the base consonant. Such occurrences are rare but do turn up in the Chuon Nath dictionary. It may also occur before 17C7 (◌ះ).

U+17D0 Samyok Sannya ◌័

This modifies the vowel and final sound on a syllable. In many respects, samyok sannya behaves like a vowel. It pushes muusikatoan and triisap down. Since there is a possible sequence that renders identically with 17BB, samyok sannya may not co-occur with -u 17BB. It has similar positioning behaviour to Kakabat and Ahsda. In Tampuan there are words like: ក្រោ័ះ 179A 1780

`17D2 179A 17B6 17D0 17C7` where the `17D0` (◌័) is rendered over the reahmuk. This sequence does not occur in Modern Khmer.

Samyok sannya may also be used for transliteration purposes. For example in the Chuon Nath dictionary it may be followed by kakabat to represent Thai tones.

U+17D1 Viriam ◌៑

This kills the inherent vowel in a consonant and as such marks it as a final. This currently does not co-occur with Bantoc, which has a similar effect as well. This character is primarily used in transliteration of Old Khmer (Chuon Nath Dictionary), where it is common. It does not co-occur with vowels or shifters.

U+17DD Atthacan ◌៝

This was originally used to indicate the presence of the inherent vowel. In minority languages, its meaning is often reappropriated to represent a vowel, e.g. in Bunong: នហ៝រ៝ /nho ro/ 'unending'. May be followed by kakabat or toandakhiat. There are no known occurrences of atthacan with a consonant shifter. But since atthacan behaves like a vowel, it is given the same downshifting behaviour as an above vowel.

U+17C7 Reahmuk ◌ះ

Acts like a final -h as is found in other scripts in the region (Devanagari, Myanmar). It is spacing but is not considered its own cluster. It can also occur following an independent vowel. There is no expectation to be able to insert a cursor before it. In Tampuan samyok sannya (`17D0`) renders over the `17C7` even though it is stored before. This is language specific styling. See the example in the discussion of samyok sannya above.

U+17C8 Yuukaleapintu ◌ៈ

This indicates a stopped short vowel which may occur on its own or after another vowel.

Miscellaneous

Other Characters

There is a set of characters that stand alone in terms of the syllable structure. This includes digits, punctuation, etc. It also includes all deprecated characters because they should not occur in real data. These characters do not participate in orthographic syllables.

```
Syllable = Base Robat? Coengs? Shifter? Vowels Modifier{0,2} Final?
```

```
Other = [17A3 17A4 17B4 17B5 17D3-17DC]
              # Other: Symbols and deprecated characters within range 1780-17DD
```

Collation

There is a de facto standard for sorting Khmer complete with a tailoring[5] specific to Khmer. But its full analysis and review is beyond the scope of this document. However, we can learn some expectations of what sequences might be expected to exist for collation to work well. In general, collation follows the Chuon Nath dictionary and the Unicode tailoring for Khmer works well pretty much regardless of the order of characters in the cluster. The primary orders are: Consonant, Vowel, Coengs. With Shifter and the Modifiers taking secondary orders. With Coengs

[5] Tailoring is a technical term for language specific collation variation.

being stored before Vowels, the coeng sign character (17D2) is given a sort key greater than any vowel, that way Cons + Vowel sorts before Cons + Coeng.

The CLDR tailoring, for Modern Khmer, currently assumes that both robat and coengs follow the consonant immediately. The mapping of robat to final r in the previous consonant is handled in sort tailoring by making the consonant robat sequence sort almost equivalently to the letter ro followed by a coeng of the consonant. This replacement means that a consonant shifter cannot be stored before the coeng otherwise, given that ro (179A) can take a shifter, the shifter would interfere with the contraction (a code sequence with a single collation key). This establishes the initial cluster order as:

```
Base Robat? Coeng{0,2} Shifter?
```

A different order would cause a significant growth in tailoring rules. The cost of changing the order to move the S to before the coeng is significant, if not huge. Notice that this ordering also means that the robat cannot become just another MS.

Extensions

In the introductory scope section we listed 5 names with novel spellings. Such spellings are rare and typically arise in multilingual contexts where a minority language is the foundation for someone's name. These novel spellings were found on voting registration forms and had been typed by officials. How would these be encoded in the proposed encoding structure?

Table 1 - Unconventional words and spellings.

Issues	Examples	Spelling	Proposed Unicode
Two shifters, one stays up and another downshifted	ស្រ៊៊ីម	ស ◌្ រ ◌៊ ◌៊ ◌ី ម	179F 17D2 179A 17CA *17CA* 17B8 1798
Consonant shifter used despite their being an alternative consonant in the other series.	ឃ៉ី	ឃ ◌៉ ◌ី (alternative: ខី - ខ ◌ី)	1783 17C9 17B8
	ណ៊ប	ណ ◌៊ ប (alternative: នប)	178E 17CA 1794
Two vowels in a row	រ៉មីាអះ	រ ◌៉ ម ◌ី ◌ា អ ◌ះ	179A 17C9 1798 17B8 17B6 17A2 17C7
Final coeng (កនិ្ថា)	[6] កន្ថិា	ក ន ◌្ ថ ◌ិ ◌ា ក ន ◌ិ ◌ា ◌្ ថ ក ន ◌ិ ◌្ ថ ◌ា	1780 1793 17D2 1790 17B7 17B6

The extensions provided to the encoding structure beyond what is strictly needed for Modern Khmer are in keeping with there being only one way to encode one visual form. An encoding structure is not a lightweight spell checker. Its aim is to allow the maximal representation of textual expression while balancing uniqueness and implementation costs. There is no point in allowing a sequence that is difficult to render, that nobody is going to use. It is hoped this document has the balance correct, but only time will tell. As such the first example above is disallowed. The cost of supporting more than one consonant shifter outweighs the benefit for a single name, even assuming it is correctly spelled.

Final Coengs

Middle Khmer allowed for post-vowel consonants represented using final coengs. Modern Khmer dictionary words don't use them orthographically. That is, what may be interpreted as a final coeng in say: ឲ្យ (17B2 17D2 1799) is treated as an initial coeng orthographically. Some personal

[6] The rendering in this image is erroneous. This is more accurate: កន្ថិា. Since the coeng is between the first and second vowel, it renders below the base not between the base and the final vowel, which would show it follows that vowel.

names, for example, do use final coengs as a carry over from Middle Khmer. This causes some confusion. Consider the situation regarding a spacing coeng and a non spacing vowel: ផ្សិ (1795 **17D2 179F** 17B7) in which the coeng is before the vowel and ផិ្ស (1795 17B7 **17D2 179F**) in which the coeng is after the vowel. In Modern Khmer there is no contrast between these two character sequences because the coeng never occurs after the vowel. Modern Khmer users therefore, typically do not see a contrast between these two and so will happily type either. For Modern Khmer, therefore, we never want to allow people to type a coeng after the vowel and have it stored as a final coeng. There are two ways to ensure this:

- Require the use of a smart keyboard that can reorder the coeng typed after the vowel to before, or
- Disallow the storage of coengs after vowels and mark them as erroneous.

The first solution is not available to most users because systems typically do not ship with keyboards smart enough to address this issue. This means the only option is the latter.

For Middle Khmer, there are three positions in which a final coeng may be stored in the structure: immediately after the medial signs, immediately after the vowels or immediately after the coengs.

- *After medial signs*: This position can result in the sequence atthacan (17DD) coeng (17D2). The combining order for 17DD is 230 and for 17D2 it is 9. Thus under normalization, the characters would be reversed causing all kinds of problems. This occurrence should be avoided and this is best done by not storing final coengs after the medial signs. Thankfully atthacan is a relatively recent invention from around 1900 and does not occur in Middle Khmer and so not in conjunction with a final coeng.
- *After the vowels*: This is a possibility, but the Universal Shaping Engine (USE) does not support such sequences. It might be argued that the USE can be structurally changed or a new Khmer specific shaper be developed. In the current technical climate both of these are unlikely. Without final coengs after vowels, the USE can support the syllable structure (if not with full checking) merely by changing the default categories of a number of characters, using established techniques. As has often been stated, there is a case for adding extra checking to the front of the USE and this may be possible, but it is unlikely that the core USE structure itself will be changed, just for Middle Khmer. This problem also applies to positioning final coengs after medial signs.
- *Last Coeng*: This leaves placing final coengs after any other coengs. This position adds complexity to font implementation with final coengs having to reorder after spacing vowels. But apart from this added cost, there are no technical limitations on this position. Another value of this position is that it harmonizes better with the needs of a related script, Tai Tham[7].

Having chosen a position in the order, we need to mark the final coeng in some way. We do this using a ZWJ. The question is where that ZWJ should go in relation to the coeng. There are three options: before the coeng, between the coeng and the base and after the coeng (and its internal base code). All three approaches are implementable[8] with the existing shaping engines, even if some sequences are currently considered illegal.

- Between coeng and base. This has the advantage of being very different from any other encoding of coengs. It fits the Devanagari model. The problem with this position is that it breaks the tight coupling of a coeng character followed by a base character. This breaks

[7] Tai Tham is a closely related script, particularly to Middle Khmer. Tai Tham makes regular use of final consonants. But in its modern form, the visual variation is rarely used. The confusion is multiplied by some coeng forms representing any of a medial, vowel or final. Thus the marking of final consonants can be considered to be in near free variation with unmarked forms (medials). Requiring a different position emphasizes a difference that is often not perceived by typists. Instead treating final and non final medials the same but allowing marking makes for a better user experience. For example words can be compared simply by ignoring ZWJ. The definition of Tai Tham is waiting on the definition of Khmer, that the two may be harmonized as much as possible.

[8] At the time of writing, in Harfbuzz and DirectWrite, but not CoreText engine.

most Khmer fonts. It also makes tooling around Khmer more difficult in having to support ZWJ in between the coeng character and the following base.

- Before coeng. This has the advantage of giving a clear boundary between medial coengs and final coengs. In Khmer there is only one final coeng, but Tai Tham allows for multiple final coengs. The precise encoding for Tai Tham is out of scope for this document.
- After base. Along with the before coeng position, this position more naturally reflects the implementation in marking the final coeng glyph during processing. This is a less preferable position since it places the ZWJ further from the coeng character itself.

The conclusion is to use the before coeng position for the ZWJ to mark a final coeng. While any position is technically feasible, the before coeng position has the most benefits, feels most natural to those looking at the sequence and has the least implementation difficulties.

In order to ensure that a coeng does indeed cause a change in visual form, a final coeng is constrained in relation to the vowel(s) in the syllable. For non-spacing diacritic vowels, the coeng must be right spacing. This is because there is a visual contrast in the position of the vowel in relation to a non-final versus final coeng. For spacing vowels, any coeng (apart from coeng ro) may be used. An added complexity can arise where, while there is no visual contrast (both the coeng and above vowel are non-spacing), a final coeng, if stored as an initial coeng, would change the consonant shifter downshifting rules. This can happen if a syllable cluster only contains series 2 consonants, but the final is series 1. In this case, when there is a downshifting consonant shifter, we require the non spacing final coeng to not be stored initially as it would normally be. The other case where the final coeng could affect the series of the consonant cluster is if the final coeng is a BA following a series 1 consonant cluster. But since coeng BA is spacing, it would be marked anyway in the context of an above vowel.

Normally there is no contrast between a medial and final coeng if the coeng and the vowel are non spacing. But in the case of nyo, there is a contrast in that the swash underneath is removed for a medial coeng but not for a final coeng.

```
Syllable = Base Coengs FinalCoeng? Shifter? Vowels? Modifier{0,2} Final?

FinalCoeng = (FCoengID RightCoengBase
                   (?= ShifterChar? (  BelowVowel AboveVowelSamyok
                                     | BelowVowel))     # spacing coeng
              | FCoengID NonRo
                   (?= ShifterChar? [17C2-17C3]? BelowVowel? AboveVowel?
                       SpacingVowel)                    # spacing vowel
       # final coeng series 1 in a series 2 consonant cluster
             | (?<= (  Series2Base Robat? (Coeng Series2Base){0,2}
                     | BA Robat? (Coeng Base){0,2} | (Coeng BA (Coeng Base)?)))
               FCoengID Series1Base (?= 17C9 AboveVowelSamyok))
    | (?<= 1789) FCoengID AnyFinalCoeng          # special layout

Coeng = 17D2
FCoengID = (ZWJ Coeng)
BA = 1794
RightCoengBase = [1783 1788 178D 1794 1799 179F] # Right spacing coeng [ឃឈឍបយស]
Series2Base = [1784 1789 178E 1793 1798-179D 17A1]
ShifterChar = ([17C9 17CA] ZWNJ?)                # Consonant shifter
AboveVowel = [17C1-17C5]?[17B7-17BA 17BE 17DD] | 17B6 17C6     # Above vowel context
AboveVowelSamyok = AboveVowel | [17C1-17C3]? 17D0       # Above vowel or samyok sannya
BelowVowel = [17C1-17C3]?[17BB-17BD]             # Below vowel context
SpacingVowel = [17B6 17BF 17C0 17C4 17C5]        # Spacing vowel context
ZWNJ = 200C                                      # Zero width non-joiner
ZWJ = 200D                                       # Zero width joiner
```

The reason that this approach does not suffer from the dangers of mistaken data entry is that a user may not type a coeng after a vowel, and must consciously think about requiring a final coeng. Where a smart keyboard is being used, such constraints are not problematic, but where a dumb keyboard[9] is used, the contrast is significant.

[9] A dumb keyboard has no ability to constrain input or to change anything that has been typed before.

Conclusion

After all this analysis, the full structure of a syllable in the Khmer script is still complex. But in most cases, people are only concerned with Modern Khmer and related minority languages. For this a much simpler structure is sufficient. For those implementing foundational technologies like shapers, it is essential that they provide full support for the complete description as given for Middle Khmer, otherwise if systems restrict their support to only Modern Khmer, they will make it impossible to support Middle Khmer.

Modern Khmer

The modern Khmer encoding structure is relatively straightforward with the only complexity being where consonant shifters are downshifted. To aid in reading, the top level elements are bolded. This is the only significance of bold in the description.

```
Syllable = Base Robat? Coengs? Shifter? Vowel? Modifiers? Final? | Other

Base = [1780-17A2 17A5-17B3]            # Base consonant [ក-ឳ]
Robat = 17CC                            # Robat [៌]

Coengs = ((Conjoiner NonRo)? Conjoiner Base)
Conjoiner = 17D2                        # Coeng character
NonRo = [1780-1799 179B-17A2 17A5-17B3] # Non Ro letter

Shifter = (  (?<= StrongContext) 17CA ZWNJ (?=AboveVowel)
           | (?<! StrongContext) 17C9 ZWNJ (?=AboveVowelSamyok)
           | [17C9 17CA])

StrongContext = (StrongBase Robat? CoengNonBA{0,2}
                | NonBA Robat? (StrongCoeng CoengNonBA? | CoengNonBA StrongCoeng))
                                        # Contains a strong consonant and no BA

AboveVowel = ([17B7-17BA 17BE 17DD] | 17B6 17C6)     # Above vowel context
AboveVowelSamyok = (AboveVowel | [17C1-17C3]? 17D0)  # Above vowel or samyok sannya context
StrongBase = [1780-1783 1785-1788 178A-178D 178F-1792 1795-1797 179E-17A0 17A2]
                           # Strong consonants [កខគឃ  ចឆជឈ  ដឋឌឍ  តថទធ  ផពភ  មសហ  អ]
CoengNonBA = (Conjoiner NonBA)          # Coeng no BA (ប   1794)
NonBA = [1780-1793 1795-17A2]           # Any consonant with no BA (ប   1794)
StrongCoeng = (Conjoiner StrongBase)    # Strong coengs
ZWNJ = 200C                             # Zero width non-joiner

Vowel = [17B4-17C5]

Modifiers = (([17C6 17CB 17CD-17CF 17D1] | (?<!17BB) [17D0 17DD])
            [17C6 17CB 17CD-17D1 17DD]?)

Final = [17C7 17C8]                     # Modifying Final [◌ះ  ◌ៈ]
Other = [17A3 17A4 17B4 17B5 17D3-17DC] # Other: Standalone exceptions within 1780-17DD
```

Description

Another way of describing the encoding structure is in prose. The purpose of a prose description is somewhat different to a regular expression. The regular expression is complete and precise and formally testable. This prose description describes the intent of the regular expression in high level terms, while still having a clear relationship with the regular expression.

A syllable consists of an initial Consonant or Independent Vowel followed by an optional robat. This is followed by up to 2 coengs which are constrained so that coeng ro is always second of two initial coengs.

After the coengs, if present, there may be a consonant shifter which may be followed by a ZWNJ if the vowel and consonant cluster context would cause it to downshift. A consonant shifter will downshift if it is followed by an upper vowel or appropriate diacritic or spacing vowel with nikahit. It will also only downshift if the consonant shifter is used to change the series of the initial consonant cluster. Thus:

- A triisap (17CA) will only downshift if the consonant cluster contains no BA (1794) and contains a series 1 consonant anywhere in the cluster. In such a context it may be followed by a ZWNJ to stop downshifting.
- A muusikatoan (17C9) will only downshift if the consonant cluster contains a BA (1794) or contains only series 2 consonants. In such a context it may be followed by a ZWNJ to stop downshifting.

Then comes an optional single vowel followed by any diacritic modifiers. There may be up to 2 modifier signs. Again the modifier signs are limited following -u (17BB) to those that would not cause a consonant shifter to downshift.
Finally there may be a final mark: reahmuk or yuukaleapintu.

Alternatively a syllable consists of some other Khmer character or digit not already described in the main syllable structure.

Middle Khmer

The full description includes support for Middle Khmer. This adds the concept of a final coeng and of vowel sequences. For implementers this is the full authoritative structure of the Khmer encoding.

```
Syllable = Base Robat? Coengs? FinalCoeng? Shifter? Vowels? Modifiers? Final?

Base = [1780-17A2 17A5-17B3]                      #  Base consonant [ក-ឳ]
Robat = 17CC                                      #  Robat [៌]
Coengs =  ((Conjoiner NonRo)? Coeng)
Coeng = (Conjoiner Base)
Conjoiner = 17D2                                  #  Coeng character
NonRo = [1780-1799 179B-17A2 17A5-17B3]           #  Non Ro letter

FinalCoeng = (FCoengId RightCoengBase                           #  spacing coeng
                 (?= ShifterSeq? (BelowVowel? AboveVowelSamyok | BelowVowel))
    | AnyFinalCoeng (?= ShifterSeq? [17C2-17C3]? BelowVowel? AboveVowel? SpacingVowel)
                                                                #  spacing vowel
    | (?<= WeakBase Robat? (Conjoiner WeakBase){0,2} | BA Robat? Coengs?
         | Base Robat? (Conjoiner BA Coeng? | Conjoiner NonRo Conjoiner BA))
        FCoengID StrongBase (?= 17C9 ZWNJ? AboveVowelSamyok))   #  series 1 after series 2 cluster
    | (?<= 1789) FCoengID AnyFinalCoeng           # special layout

FCoengID = (ZWJ Conjoiner)
AnyFinalCoeng = (FCoengId NonRo)                                #  Coeng marked as final
AboveVowel = ([17C1-17C5]?[17B7-17BA 17BE 17DD] | 17B6 17C6)    #  Above vowel context
AboveVowelSamyok = AboveVowel | [17C1-17C3]? 17D0        #  Above vowel or samyok sannya context
BA = 1794                                                #  BA consonant [ប]
BelowVowel = [17C1-17C3]?[17BB-17BD]                     #  Below vowel context
RightCoengBase = [1783 1788 178D 1794 1799 179E 179F 17A1]
                                          #  Right spacing coeng base [ឃឈឍបយមសឡ]
StrongBase = [1780-1783 1785-1788 178A-178D 178F-1792 1795-1797 179E-17A0 17A2]
                                          #  Strong consonants [កខគឃ ចឆជឈ ដឋឌឍ តថទធ ផពភ មសហ អ]
ShifterSeq = ([17C9 17CA] ZWNJ?)                  #  Consonant Shifter character [៉ ៊]
SpacingVowel = [17B6 17BF 17C0 17C4 17C5]         #  Spacing vowel context
WeakBase = [1784 1789 178E 1793 1798-179D 17A1]   #  Weak consonants [ងញណនមយរលវឝឡ]
ZWJ = 200D                                        #  Zero width joiner
ZWNJ = 200C                                       #  Zero width non-joiner

Shifter = (  (?<= StrongContext AnyFinalCoeng?) 17CA ZWNJ (?=AboveVowel)
           | (?<! StrongContext AnyFinalCoeng?) 17C9 ZWNJ (?=AboveVowelSamyok)
           | [17C9 17CA])

StrongContext = (  StrongBase Robat? CoengNonBA{0,2}
                 | NonBA Robat? (StrongCoeng CoengNonBA? | CoengNonBA StrongCoeng))
                                                  #  Contains a strong consonant and no BA
CoengNonBA = (Conjoiner NonBA)                    #  Coeng no BA (ប  1794)
NonBA = [1780-1793 1795-17A2]                     #  Any consonant with no BA (ប  1794)
StrongCoeng = (Conjoiner StrongBase)              #  Strong coengs

Vowels = ([17C1-17C5] [17BC 17BD]? [17B7 17B9 17BA]?
        | [17C2 17C3 17C5]? [17BC 17BD]? 17B8
        | [17C2-17C3]? [17BC 17BD]? [17B7-17BA]? 17B6 | 17BE [17BC 17BD]? 17B6?
        | [17BF 17C0] | [17C2-17C3]? 17BB 17B6 (?! 17C6)
        | [17C1-17C5]? 17BB (?! [17D0 17DD]))

Modifiers = (([17C6 17CB 17CD-17CF 17D1] | (?<!17BB [17B6 17C4 17C5]?) [17D0 17DD])
            [17C6 17CB 17CD-17D1 17DD]?)          #  Modifying Sign [ំ ់ ៍ ៎ ៏ ៑ ៝ ័]
Final = [17C7 17C8]                               #  Modifying Final [◌ះ  ◌ៈ]
Other = [17A3 17A4 17B4 17B5 17D3-17DC]           #  Other: Standalone exceptions within 1780-17DD
```

Description

A syllable consists of an initial Consonant or Independent Vowel followed by an optional robat. This is followed by up to 2 coengs which are constrained as follows. Coeng ro is always second of two initial coengs.

A final coeng may occur after the coengs. Final coengs are marked with a ZWJ between the coeng code and the following base. Final coengs are constrained to 3 contexts:

- A spacing coeng may take a final form before a non-spacing vowel.
- Any coeng (except ro) may take a final form before a spacing vowel.
- Where an unmarked coeng would erroneously change a potentially downshifted muusikatoan from being downshifted to not downshifted, then the coeng must be marked as final in order not to interfere. This only happens if the final coeng is from series 1 and the rest of the consonant cluster is all series 2.
- Normally a coeng causes the swash on a nyo to disappear, but in Middle Khmer it is kept if the coeng underneath is a final coeng, following a vowel.

After the coengs and final coeng, there may be a consonant shifter which may be followed by a ZWNJ if the vowel and consonant cluster context would cause it to downshift. A consonant shifter will downshift if it is followed by an upper vowel or appropriate diacritic or spacing vowel with nikahit. It will also only downshift if the consonant shifter is used to change the series of the initial consonant cluster. Thus:

- A triisap (17CA) will only downshift if the consonant cluster contains no BA (1794) and contains a series 1 consonant anywhere in the cluster. In such a context it may be followed by a ZWNJ to stop downshifting.
- A muusikatoan (17C9) will only downshift if the consonant cluster contains a BA (1794) or contains only series 2 consonants. In such a context it may be followed by a ZWNJ to stop downshifting.

The vowel, or vowels, may come after the consonant shifters. Modern Khmer only uses a single vowel, but Middle Khmer and some names use multiple vowels. The only vowel sequences allowed are those that do not result in:

- More than one visual component in any of the 4 positions: before, above, below, and after.
- A -u (17BB) occurring with an above vowel as would be the presentation form of a downshifted consonant shifter.
- A sequence for which there is already a single vowel code.

There may be up to 2 modifier signs. Again the modifier signs are limited following -u (17BB) to those that would not cause a consonant shifter to downshift.

Finally there may be a final mark: reahmuk or yuukaleapintu.

Alternatively a syllable consists of some other Khmer character or digit not already described in the main syllable structure.

Consonant Shifters

In Khmer orthography, consonant letters are considered to fall into two series, series 1 and 2. When reading Khmer, the phonetic vowel is derived from a combination of the series of the initial orthographic consonant cluster and the written or inherent vowel. Most vowel signs have a different sound associated with the two series. Most consonants have series 1 and series 2 pairs. In some cases, where there is no equivalent consonant in the other series, a consonant is identified as being able to take a consonant shifter to switch its series. To switch from series 1 to series 2, a triisap (17CA ◌៊) is used, and from series 2 to series 1, a muusikatoan (17C9 ◌៉) is used. The set of consonants in series 1 is B1 and the set in series 2 is B2. Within these series, only a smaller number can take a consonant shifter (B1S, B2S).

```
S = [17C9 17CA]
B1 = [1780 1781 1785 1786 178A 178B 178E 178F 1790 1794 1795 179E 179F 17A0 17A1 17A2]
B2 = [1782 1783 1784 1787 1788 1789 178C 178D 1791 1792 1793 1796 1797 1798 1799 179A
179B 179C 179D]
B1S = [1794 179E 179F 17A0 17A2]
B2S = [1784 1789 1793 1798 1799 179A 179B 179C 179D]
```

Several special cases exist in series assignment and consonant shifter use:

- 178E (ណ) and 179E (ឞ) have their series swapped from that implied by the Unicode name in the Unicode chart for Khmer.
- 179D (ឝ), as used for Pali/Sanskrit transliteration, gave the character series 1. But when the character was reappropriated for minority language use, it was given series 2, in keeping with its visual representation (the 'hair' on the base character does not combine visually with muusikatoan)[10]. Bear in mind that Pali and Sanskrit do not use consonant shifters.
- 1793 (ន) and 179B (ល) do have their other series counterparts, but are included in this list because their other series counterparts are Pali characters, and when showing pronunciation, Pali characters are avoided and a muusikatoan may be used.
- 1794 (ប ba) is used with triisap to create a series 2 counterpart bo, but also with muusikatoan to create a different consonant pa that is the series 1 counterpart to 1796 (ព po).

But what is the series of a consonant cluster? The series is easy to derive when there is only one consonant involved: the series of the cluster is the series of the consonant. But when a consonant cluster consists of more than one consonant (via coengs) of consonants of different series, which consonant determines the series for the cluster?

Makara Sok (2016, table 10) presents a chart of orthographic consonants based on a sonority hierarchy. His hypothesis is that the consonant within a cluster with the lowest sonority in the hierarchy (earlier in the table) determines the series for the cluster. His chart is reproduced here using Unicode codepoints. In addition, minority characters are added.

[10] The known minority languages that use consonant shifters are: Brao [brb], Jarai [jra], Kuay [kdt], Northern Khmer [kxm], Tampuan [tpu] and they all do downshifting.

Table 2 - Consonant Sonorities

Class	1st Series	2nd Series
1. Implosive	ដ(178A) ប(1794)	ឌ(178C) ប៊(1794 17CA)
2. Unaspirated Plosive	ក(1780) ច(1785) ត(178F) ប៉(1794 17C9) អ(17A2)	គ(1782) ជ(1787) ទ(1791) ព(1796) អ៊(17A2 17CA)
3. Aspirated Plosive	ខ(1781) ឆ(1786) ឋ(178B) ថ(1790) ផ(1795)	ឃ(1783) ឈ(1788) ឍ(178D) ធ(1792) ភ(1797)
4. Fricative	ឞ(179E) ស(179F) ហ(17A0)	ឞ៊(179E 17CA) ស៊(179F 17CA) ហ៊(17A0 17CA)
5. Minor Fricative	ឝ៉(179D 17C9)	ឝ(179D)
6. Sonorant	ង៉(1784 17C9) ញ៉(1789 17C9) ន៉(1793 17C9) ណ(178E) ម៉(1798 17C9) យ៉(1799 17C9) រ៉(179A 17C9) ល៉(179B 17C9) ឡ(17A1) វ៉(179C 17C9)	ង(1784) ញ(1789) ន(1793) ម(1798) យ(1799) រ(179A) ល(179B) វ(179C)

Downshifting

Typographically rendering a vowel above a consonant with a consonant shifter can lead to visual problems in particular to the line height. For example:

ប៊ឹម

To resolve this, there is an orthographic principle that, in most contexts involving a consonant shifter followed by certain above diacritics, the consonant shifter is rendered as a -u vowel (17BB) instead of its default form. The syllable looks like it has two vowels. The list of diacritics that have this effect are the above-base vowels and samyok sannya. It may also involve -aa (17B6 ◌ា) followed by a nikahit (17C6 ◌ំ).

Theoretically, there are two ways to encode this behaviour:

1. A downshifted consonant shifter is encoded as 17BB ◌◌ុ, letting the user control whether it should be shown downshifted or not.
2. A downshifted consonant shifter is encoded as the original shifter, 17C9 ◌៉ or 17CA ◌៊, and rendered with the glyph ◌ុ when contextually necessary.

However, the Unicode Standard chose option 2 when encoding Khmer over twenty years ago. Fonts have implemented downshifting based on this choice, and most text is encoded based on it. This makes switching to option 1 unfeasible. Option 2 requires conversions in two directions: from character sequence to visual rendering, which is controlled by the font, and from visually oriented input (e.g., the user types 17BB ◌ុ and an above-base vowel) to correct character sequence (the contextually appropriate consonant shifter followed by the above-base vowel). This section examines that question: how to render/shape a consonant shifter with an upper vowel and how to convert an input of a -u vowel and upper vowel to the correct consonant shifter.

For the most part of this discussion, we presume that someone would only insert a consonant shifter for a consonant that takes a shifter. That is one that does not have a corresponding full consonant in the other series. Looking at table 2, the highlighted cells show those cells containing consonants with consonant shifters. Notice that the highlighted first series cells for the minor fricative and sonorant rows contain second series consonants with consonant shifters. Likewise, the second series cells for the first four rows contain a mix of second series consonants and shifted first series consonants. We can therefore say for Modern Khmer that if a well formed consonant cluster is followed by a muusikatoan (17C9), the consonant involved must be a minor fricative or sonorant. In addition we can say that because the muusikatoan applies to the whole cluster, the whole cluster must have a sonority that is row 5 or 6 and that there are no consonants from rows 1-4 in it, otherwise the cluster will be dominated by that other consonant and the muusikatoan would not apply to such a consonant.

So a first rule is that if a consonant cluster only consists of characters of sonority 5 or 6, then the consonant shifter is a muusikatoan. And therefore, if there are characters of sonority less than 5 in the cluster, the consonant shifter is a triisap.

This massively simplifies the whole structure, which would otherwise result in a huge complex regular expression to handle all the cases, most of which are not well formed Modern Khmer and so do not need to be 'correctly' supported. As it is, the regular expression needed to support even these relatively simple rules is complicated enough.

The downshifting rules therefore become:

1. If a cluster contains any consonants from sonority levels 1-4, a following triisap (17CA) will be downshifted.
2. If a cluster contains only consonants from sonority levels 5-6, a following muusikatoan (17C9) will be downshifted.

In reverse, if a -u is followed by an above-base vowel, then it will be converted to a triisap if the cluster contains any consonants from sonority levels 1-4, and to a muusikatoan otherwise.

This regularising of the encoding has some interesting implications for those wishing to step outside the bounds of Modern Khmer spelling. For example, someone might want to put a consonant shifter over a consonant for which there is a corresponding consonant in the other series. In Modern Khmer, therefore, such a consonant would never take a consonant shifter. If the user desires that the shifter they add downshifts, then they may need to insert the wrong consonant shifter for the series of the consonant they are putting it over. For example, for a consonant shifter after the sequence SA 179F, coeng KO 17D2 1782 (ស្គ) to downshift, it must be a triisap, even though KO is second series and so would 'typically' take a muusikatoan. But either sequence is anything but 'typical', given it could never occur in Modern Khmer. The linguistically correct cluster would consist of SA 179F, coeng KA 17D2 1780 (ស្ក), which is in the first series.

Handling BA (U+1794)

Ba has its own unique behaviour with respect to consonant shifters. The sequence 1794 17C9 is a different consonant, but of the same series as 1794. The sequence 1794 17CA is the same consonant but in the other series (from series 1 to series 2). One might expect, therefore, for the 17CA to downshift if followed by an upper vowel. But it is the other way around, with the 17C9 downshifting and the 17CA staying up. Ba is in sonority level 1 and so according to the rules above should be followed by a triisap (17CA). But we need the rule to change such that it is the 17C9 that downshifts and so is converted to -u. Thus if a consonant cluster contains a Ba then a muusikatoan will downshift and the triisap will not. Thus our rules change to:

1. If a cluster contains any consonant from sonority levels 1-4, and does not contain a BA, a following triisap (17CA) will be downshifted.
2. If a cluster contains a BA or only consonants from sonority levels 5-6, a following muusikatoan (17C9) will be downshifted.
3. If a BA occurs with a consonant from sonority levels 1-4, it still conforms to rule 2 (failing

rule 1) and it is a following muusikatoan (17C9) that is downshifted.

Regular Expressions

Can we express these rules as regular expressions? What is the regular expression that if matched will cause the consonant shifter to be downshifted?

For rule 1 we need a class of all consonants that are in sonority levels 1-4, excluding BA, and a class of all consonants of sonority levels 5-6. For convenience we also need a class of all consonants excluding BA and a class of all sonority level 5-6 and BA as well.

```
S1 = [1780-1783 1785-1788 178A-178D 178F-1792 1795-1797 179E-17A0 17A2]
S2 = [1784 1789 178E 1793 1798-179D 17A1]
SNB = [1780-1793 1795-17A2]
S2B = [1784 1789 178E 1793 1794 1798-179D 17A1]
BA = 1794
```

Rule 1 can be expressed with the regular expression::

```
(S1 (17D2 SNB){0,2} | SNB (17D2 S1) (17D2 SNB)? | SNB (17D2 SNB) (17D2 S1)) 17CA
```

If we create coeng classes, we can simplify this:

```
SC1 = (17D2 S1)
SC2B = (17D2 S2B)
SCNB = (17D2 SNB)
SCB = (17D2 1794)
```

```
Rule 1: (S1 SCNB{0,2} | SNB (SC1 SCNB? | SCNB SC1)) 17CA
Rule 2: (S2 SC2B{0,2} | BA SC2B{0,2} | B (SC2B C? | C SC2B)) 17C9
```

Notice that these rules are relatively relaxed. If a cluster contains no consonants that would take a consonant shifter, if the appropriate shifter occurs, then it will downshift. This includes if the consonants involved are already of the series being shifted to.

For example, the sequence 1782 17CA 17B7 will downshift, despite the sequence being linguistic nonsense. The reason is that if we were to cover all the nonsense cases appropriately, the regular expressions would become unmanageable. The important thing is that for any one visual representation, there is one unique sequence that will render to it. It makes little difference whether it is 17C9 or 17CA that downshifts after a 1782, since a consonant shifter should never occur after a 1782. If someone wants to have such a shifter in their name, it is not unreasonable to require them to use a 'wrong' shifter to get the visual representation they desire.

ZWNJ

It is not always desirable for a consonant shifter to downshift. Thus we allow the insertion of a ZWNJ to block the downshifting. But we can only allow ZWNJ in a context where downshifting would actually occur. If we allowed it in the context where a consonant shifter did not downshift, then it would have no effect and there would be two different sequences (one with the ZWNJ and one without) that would render the same. This would result in the encoding ambiguity that this whole paper is written to avoid.

As part of our main regular expression, therefore we can say that wherever a consonant shifter would downshift, we can add a ZWNJ:

```
  (S1 SCNB{0,2} | SNB (SC1 SCNB? | SCNB SC1)) 17CA 200C
| (S2B SC2B{0,2} | B (SCB C? | C SCB)) 17C9 200C
```

In the main regular expression, 17C9 and 17CA are always allowed, but ZWNJ (U+200C) is only allowed in the context where the consonant shifter would downshift. We also merge S2 SC2B{0,2} | BA SC2B{0,2} to become S2B SC2B{0,2}.

The Main Regular Expression

We are now in a position to integrate this regular expression into the main full syllable description. We do this by using forward and backward zero width constraints[11]. At the point in the regular expression where we match a consonant shifter, we have already matched the initial consonant cluster and have yet to match the following vowels and diacritics. Thus to constrain the consonant shifter sequences, we do an extra look behind assertion to look back into the consonant cluster we have just matched and to look ahead to the vowels we are about to match:

```
S = ( ((  (?<= S1 SCNB{0,2} | SNB (SC1 SCNB? | SCNB SC1)) 17CA ZWNJ
        | (?<= S2B SC2B{0,2} | BA (SCB C? | C SCB))       17C9 ZWNJ
       ) (?=[17C1-17C3]?[17B7-17BA 17BE 17D0 17DD] | 17B6 17C6))
      | [17C9 17CA]
    )
```

One characteristic of this expression that can help us simplify things further is that

```
WEAK = S2B SC2B{0,2} | BA (SCB C? | C SCB)
```

is the opposite of

```
STRONG = S1 SCNB{0,2} | SNB (SC1 SCNB? | SCNB SC1)
```

That is, within the character set domain of what may precede a consonant shifter in a syllable, matching WEAK is the same as not matching STRONG. Thus we can refactor our regular expression into something much simpler and clearer.

```
S = ((((?<= STRONG) 17CA ZWNJ | (?<! STRONG) 17C9 ZWNJ) (?=VA)) | [17C9 17CA])
```

```
STRONG = S1 R? SCNB{0,2} | SNB R? (SC1 SCNB? | SCNB SC1)# Contains strong consonant
VA = ([17C1-17C3]?[17B7-17BA 17BE 17D0 17DD] | 17B6 17C6)       # Above vowel
```

Unfortunately many regular expression engines do not support a variable length look behind assertion. It is often possible to refactor a look behind assertion into a simple unity replacement, but a negative look behind assertion is much harder. In such contexts the former expression should be used and refactored.

```
S = ( ((  (?<= S1 SCNB{0,2} | SNB (SC1 SCNB? | SCNB SC1)) 17CA ZWNJ
        | (?<= S2B SC2B{0,2} | BA (SCB C? | C SCB))       17C9 ZWNJ
       ) (?=VA)
      | [17C9 17CA]
    )
```

The use of an inverted match against STRONG versus matching against WEAK introduces a difference. This is in handling independent vowels. We consider all such vowels as WEAK and therefore add them to S2:

```
S2B = [1784 1780 178E 1793 1794 1798-179D 17A1 17A3-17B3]
SC2B = (17D2 S2B)
SCB = (17D2 1794)
```

The vowel context is complicated by allowing the restricted set of vowel sequences that the main regular expression supports. Thankfully it is not necessary here to restrict the vowel sequences to only those explicitly supported, since the rest of the main regular expression will do that. But we do need to not exclude sequences that may occur.

[11] Also known as zero width assertions.

Since it is always possible to downshift a consonant shifter following any possible consonant cluster, there is no need for a way to instruct a consonant shifter to downshift, for example through the use of ZWJ. Notice though that in any context only one of the consonant shifters will downshift and it may not be the one that a user expects. But with the simpler regular expression context, it makes life easier for smart keyboards to insert the appropriate consonant shifter if so demanded.

Samyok Sannya

Samyok Sannya (17D0) only pushes muusikatoan (17C9) down. It typically does not push down triisap (17CA). We extend the regular expression relatively simply by changing some vowel contexts:

```
S = ((((?<= STRONG) SC? 17CA ZWNJ (?=VA) | (?<! STRONG) SC? 17C9 ZWNJ (?=VAS)) | SC)
```

```
STRONG = S1 R? SCNB{0,2} | SNB R? (SC1 SCNB? | SCNB SC1)# Contains a strong consonant
and no BA
VA = ([17C1-17C3]?[17B7-17BA 17BE ] | 17B6 17C6)        # Above vowel
VAS = (VA | [17C1-17C3]?[17D0 17DD])                    # Above vowel
SC = [17C9 17CA]
```

Final Coengs

Final coengs need to be ignored for the process of consonant shifter analysis. A final coeng is identified as containing a ZWJ between the coeng character and the following base. The simplest was to ignore them is to make them optional in the contexts:

```
S = ((((?<= STRONG FC?) SC? 17CA ZWNJ (?=VA) | (?<! STRONG FC?) SC? 17C9 ZWNJ (?=VAS))
| SC)
```

```
STRONG = S1 R? SCNB{0,2} | SNB R? (SC1 SCNB? | SCNB SC1)# Contains a strong consonant
and no BA|SC = [17C9 17CA]
FC = (17D2 200D NoRo)
NoRo = [1780-1793 1794-17A2 17A5-17B3]
```

Odd Consonant Shifters

Some people use a consonant shifter on a consonant that typically does not take one because there is a corresponding other series full consonant. Spelling conventions in Modern Khmer require that the other series full consonant is used. But people sometimes like to give their name identity through spelling and break this convention. This is already factored into the downshifting rules for consonant shifters and is discussed there.

Conclusion

The final result that is contributed to the main encoding structure for Modern Khmer is:

```
S = ((((?<= STRONG FC?) SC? 17CA ZWNJ (?=VA) | (?<! STRONG FC?) SC? 17C9 ZWNJ (?=VAS))
| SC)
STRONG = S1 R? SCNB{0,2} | SNB R? (SC1 SCNB? | SCNB SC1)# Contains a strong consonant
and no BA
SC = [17C9 17CA]
FC = (17D2 200D NoRo)
NoRo = [1780-1793 1794-17A2 17A5-17B3]
```

Ordering

The proposed structure defined here is the first that has a single fixed position for the consonant shifters. Previous structures have allowed the shifter to be stored immediately following the consonant it applies to, which, in effect meant anywhere after a base consonant. The difficulty with such a structure is that without a lot of disambiguation, the encoding is inherently ambiguous with two different encoded sequences rendering the same. This is probably the primary incompatibility between data stored in current encoding structures and the new structure defined here. Is the approach proposed here the best solution to the problem? In this section we consider two issues around ordering: Whether the fixed position following coengs is the most appropriate and the rendered position of consonant shifters.

Fixed Position

One of the weaknesses of using a fixed position following the coengs is that it can separate BA (ប1794) from its muusikatoan (17C9) when forming the consonant PA. This makes searching for the letter PA problematic. One cannot simply search for ប៉. Instead one has to use a regular expression: `\u1794(\u17D2[\u1780-\u17B3])*\u17C9`. It would be much simpler to allow muusikatoan to immediately follow BA. But since the muusikatoan after a BA is involved in downshifting, the rules can become quite involved:

- Only muusikatoan after BA
- Disallow muusikatoan after coeng sequence following a BA that has a muusikatoan
- Enable ZWNJ following muusikatoan after BA if followed by only 2nd series coengs and appropriate upper vowels

The regular expressions involved are not pretty. Thankfully there are very few occurrences of PA followed by a coeng in Modern Khmer. The ones known of are:

ប្រ៉ឹដ, ប្រ៉ុយ, ប្រ៉ុយណេ, ប្រ៉ែន, ប្រ៉ែនស៍, ប្ល៉ូ, ប្ល៉ូដាយ, អាំប្រ៉ីយ៉ា

An alternative solution might be to move the fixed position of the shifter from after the coengs to always coming immediately after before any coengs. This would solve the PA problem and provide a single consistent position for analysis. The difficulty here is that the complex contextual expression for STRONG and WEAK sequences gets split across the characters in question. This involves more rules. For example consider just the question of a triisap followed by a ZWNJ:

```
(?<=S1 R?) 17CA ZWNJ (?=SCNB{0,2} VA) | (?<=SNB R?) 17CA ZWNJ (?=(SC! SCNB? | SCNB
SC1) VA)
```

Here the sequence in question (17CA ZWNJ) occurs twice in the expression and the constraining context cannot be abstracted way into a single named contextual expression.

The choice to store the consonant shifter after the coengs also reflects the linguistics in which it is a consonant cluster as a whole that has a series, and the precise series of a character in the cluster is only as significant as its contribution to the series of the cluster as a whole. It is also noticeable that there are many more occurrences of words in Modern Khmer where the consonant shifter applies to a coeng than to a base with a following coeng.

Positioning the Consonant Shifter

For many years, the typewriter has been used to type Khmer script and with its fixed negative offset for diacritics, the position of a consonant shifter varies horizontally based on whether it is typed after a base character or after a spacing coeng. If after a base character, the shifter renders above the base. If it is typed after a spacing coeng, then the shifter appears between the base and the spacing coeng. The position of consonant shifter may follow a spelling convention, but it carries no linguistic value. There are no situations where the difference in position makes any difference to sound or meaning. In Modern Khmer, the convention is always place the consonant shifter over the consonant. This is different to the questions surrounding final coengs, in which the contrast does carry linguistic significance.

One example may suffice to understand why no encoding contrast is necessary. Consider the two forms of 1794 17D2 1799 17CA 17B6: ប្យ៊ា /pjaa/ and ប្យ៊ា /bjaa/. The first has the consonant shifter over the consonant, changing BA to PA. The second changes the series of YA from 1st to 2nd. The resulting consonant cluster is 2nd series. This contrast is only of interest because the consonant shifter has two different functions. In the first example, it changes the nature of the consonant. In the second it is changing the series of the cluster. But due to the sonority rules, if the consonant shifter were missing, the series of the cluster would still be 1st series because the BA (1794) is a 1st series character and has low sonority and so dominates the cluster, setting its series. Thus the second form never occurs.

What can be said is that there are rules that a font may apply if it wants to follow the typewriting style. The following rules may be applied to decide where to position a consonant shifter:

- If the base consonant in the cluster can take the consonant shifter, then position the consonant shifter over the base consonant, else
- If the spacing coeng can take the consonant shifter, then position the consonant shifter relative to the spacing coeng, else
- Position the consonant shifter over the base consonant.

“Can take the consonant shifter” means that the consonant shifter being used is an appropriate one to appear over the consonant in question.

Confusables and Undesirables

The Unicode Standard has several categories of characters or short character sequences that should be either treated as equivalent or avoided. Different categories may apply when discussing general text handling, where it's required that any text can be expressed, and processes where the need for security may require limiting what can be written. This section looks at the various categories and how they apply to Khmer.

Deprecated or discouraged

Characters are never removed from the Unicode Standard, but they can be deprecated or discouraged. "Deprecated" is defined as "strongly discouraged"; deprecated characters are retained only for compatibility with older Unicode versions and should no longer be used. "Discouraged" is not formally defined. Where for security reasons unambiguous identifiers are required, their use may be restricted.

The following Khmer characters are currently deprecated or discouraged. The Identifier_Type property can be used to restrict characters from use in identifiers; all types shown in this table mark the characters as restricted.

Characters	Rendering	Status	Identifier_Type
17A3	ឣ	Deprecated	Deprecated
17A4	ឤ	Deprecated	Deprecated
17B4	◌឴	Discouraged	Default_Ignorable
17B5	◌឵	Discouraged	Default_Ignorable
17D3	◌៓	Discouraged	Obsolete
17D8	៘	Discouraged	Obsolete Not_XID

These characters may still occur in text, but the syllable structure does not need to accommodate them. As all types Identifier_Type values used for Khmer deprecated and discouraged characters mark them as restricted, no change to them is needed.

Canonical equivalence

Two character sequences are canonically equivalent if their full canonical decompositions are identical. In general, software should treat canonically equivalent character sequences as if they were identical. The full canonical decomposition of non-Korean character sequences is determined by the Decomposition_Mapping and Canonical_Combining_Class property values of the characters in the two character sequences.

No Khmer characters have decomposition mappings. Two Khmer characters have combining classes that are not 0: 17D2 KHMER SIGN COENG has ccc=9, and 17DD KHMER SIGN ATTHACAN has ccc=230. The latter assignment by the editors of the Unicode Standard was a mistake, as it makes the character sequences <17D2 17DD> and <17DD 17D2> canonically equivalent when in reality they're not equivalent at all. <17D2 17DD> is nonsensical, because 17D2 should only be used as part of the 2-character sequences encoding coengs. <17DD 17D2>, on the other hand, is valid according to the syllable definition in section 16.4 Khmer of the Unicode Standard 15.0 if followed by a Khmer consonant or independent vowel. Normalization of a string, which maps <17DD 17D2> to <17D2 17DD>, can so lead to breaking up the 2-character sequence encoding a final coeng.

Due to the Unicode Normalization Stability policy, the erroneous assignment can not be corrected. Software that interprets Khmer character sequences with the Unicode 15 syllable structure, such as font rendering systems, needs to be aware of this equivalence and map <17D2 17DD> to <17DD 17D2> before interpreting strings. For a discussion of this issue in the context of the new syllable structure, see the section Final Coengs above.

Do not use

“Do not use” tables are used in several sections of the Unicode Standard, such as 12.1 Devanagari, to describe sequences of (otherwise valid) characters that should not be used because there’s a more appropriate character or character sequence that results in the same rendering but is not canonically equivalent. An example in Devanagari would be a consonant with inherent vowel, which should be encoded by just its own code point, not its code point followed by those for a virama and the dependent vowel *-a*. The Unicode Standard does not associate any specific implementation requirements with “do not use” sequences, and so implementation support for them is generally weak. However, the Universal Shaping Engine supports them by inserting dotted circles into disallowed sequences. It’s also reasonable to say that smart keyboards should prevent their input, and dictionaries for predictive input and spelling checkers should not use them in their reference data. In scripts where an extended Backus-Naur form (EBNF) or regular expression (RE) is used to describe the structure of their orthographic syllables, “Do not use” tables can be used complementary to a simpler EBNF/RE, or they can integrated into a then more complicated EBNF/RE, as has been done in the EBNF developed in this document

The Khmer section has so far not used “Do not use” tables. The following table presents a core set of character sequences that should not be used. Many of the disallowed character sequences are in fact currently allowed by at least some shaping engines. They assume that for Khmer below-base vowels are encoded before above-base vowels, the order preferred by HarfBuzz, the main shaping engine allowing such combinations.

For	Use	Do Not Use	Comments
◌េ◌ី	17BE	<17C1 17B8>	
◌េ◌ា	17C4	<17C1 17B6>	
◌◌ុ◌ិ	<17C9 17B7>, <17CA 17B7>	<17BB 17B7>	Above-base vowels cause U+17C9 MUUSIKATOAN and U+17CA TRIISAP to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on base consonant). Shaping engines may or may not insert dotted circle before second vowel.
◌◌ុ◌ី	<17C9 17B8>, <17CA 17B8>	<17BB 17B8>	
◌◌ុ◌ឹ	<17C9 17B9>, <17CA 17B9>	<17BB 17B9>	
◌◌ុ◌ឺ	<17C9 17BA>, <17CA 17BA>	<17BB 17BA>	
◌◌ុេ◌ី	<17C9 17BE>, <17CA 17BE>	<17BB 17BE>	
◌◌ុេ◌ី	<17C9 17BE>, <17CA 17BE>	<17BE 17BB>	

For	Use	Do Not Use	Comments
◌◌ុ◌ា◌ំ	`<17C9 17B6 17C6>`, `<17CA 17B6 17C6>`	`<17BB 17B6 17C6>`	2-character vowel ◌◌ាំ causes U+17C9 MUUSIKATOAN and U+17CA TRIISAP to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on base consonant). Shaping engines may or may not insert dotted circle before second vowel.
◌◌ុ◌័	`<17C9 17D0>`	`<17BB 17D0>`	U+17D0 SAMYOK SANNYA causes U+17C9 MUUSIKATOAN to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on base consonant).
◌◌្ក◌្រ etc.	`<17D2 179A 17D2 17xx>`	`<17D2 17xx 17D2 179A>`	For any 17xx in [1780-17B3]. Fonts generally don’t distinguish between a sequence of *coeng ro* followed by another *coeng* and the sequence of that other *coeng* followed by *coeng ro*.
◌ើា	`<17C4 17B8>`	`<17BE 17B6>`	The two sequences are visually indistinct.

Confusables

“Confusables” are character sequences that are visually similar when rendered with commonly used fonts, but are not canonically equivalent. Confusable character sequences may be fine or even necessary to use in normal text, but may have to be prevented in situations where unambiguous identifiers are needed. Unicode Technical Standard #39, Unicode Security Mechanisms, discusses confusables and is accompanied by data files listing a large number of confusables, but restricts itself to confusability between a single character and a character sequence. An extension to confusability between short fixed character sequences has been proposed in L2/22-107 and L2/22-108. Confusability enabled by overly permissive syllable structures, as discussed in this document, has not yet been considered for UTS #39 (it is to some extent addressed in the Root Zone Label Generation Rules published by ICANN). “Intentional confusable mappings” in UTS #39 are confusables where typefaces would generally use the same glyph(s).

The confusables shown in the tables of this section should be added to the UTS #39 data files, if not already present.

Confusable character sequences should also be considered in the design of smart keyboards and spelling checkers, as users may unintentionally enter an incorrect sequence that then needs to be corrected...

Intentional confusable mappings

Much of this table parallels the “Do not use” table above. Entries involving combinations of 17BB and 17BE could be omitted from the following table because current shaping engines seem to reliably insert dotted circles into combinations of 17BB and 17BE. Also added are entries for deprecated or discouraged characters, and the final one is discussed in a separate section on *coeng ta* and *coeng da*.

Source	Rendering	Target	Rendering	Comments
17BE	◌ើ	17C1 17B8	◌េ◌ី	Shaping engines may or may not insert dotted circle before second vowel; fonts use same component glyphs.
17C4	◌ោ	17C1 17B6	◌េ◌ា	
17C9 17B7	◌៉◌ិ (e.g. រ៉ិ)	17BB 17B7	◌ុ◌ិ (e.g. រុិ)	Above-base vowels cause U+17C9 MUUSIKATOAN and U+17CA TRIISAP to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on the base consonant). Shaping engines may or may not insert a dotted circle before the second vowel.
17CA 17B7	◌៊◌ិ (e.g. ស៊ិ)	17BB 17B7	◌ុ◌ិ (e.g. សុិ)	
17C9 17B8	◌៉◌ី (e.g. រ៉ី)	17BB 17B8	◌ុ◌ី (e.g. រុី)	
17CA 17B8	◌៊◌ី (e.g. ស៊ី)	17BB 17B8	◌ុ◌ី (e.g. សុី)	
17C9 17B9	◌៉◌ឹ (e.g. រ៉ឹ)	17BB 17B9	◌ុ◌ឹ (e.g. រុឹ)	
17CA 17B9	◌៊◌ឹ (e.g. ស៊ឹ)	17BB 17B9	◌ុ◌ឹ (e.g. សុឹ)	
17C9 17BA	◌៉◌ឺ (e.g. រ៉ឺ)	17BB 17BA	◌ុ◌ឺ (e.g. រុឺ)	
17CA 17BA	◌៊◌ឺ (e.g. ស៊ឺ)	17BB 17BA	◌ុ◌ឺ (e.g. សុឺ)	
17C9 17BE	◌៉◌ើ	17BB 17BE	◌ុ◌ើ	As above, but in this case current shaping engines seem to reliably insert a dotted circle before the second vowel.
17CA 17BE	◌៊◌ើ	17BB 17BE	◌ុ◌ើ	
17C9 17BE	◌៉◌ើ	17BE 17BB	◌ើ◌ុ	
17CA 17BE	◌៊◌ើ	17BE 17BB	◌ើ◌ុ	
17C9 17B6 17C6	◌៉◌ាំ (e.g. រ៉ាំ)	17BB 17B6 17C6	◌ុ◌ាំ (e.g. រុាំ)	2-character vowel ◌ាំ causes U+17C9 MUUSIKATOAN and U+17CA TRIISAP to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on base consonant). Shaping engines may or may not insert dotted circle before the second vowel.
17CA 17B6 17C6	◌៊◌ាំ (e.g. ស៊ាំ)	17BB 17B6 17C6	◌ុ◌ាំ (e.g. សុាំ)	
17C9 17D0	◌៉◌័ (e.g. រ៉័)	17BB 17D0	◌ុ◌័ (e.g. រុ័)	U+17D0 SAMYOK SANNYA causes U+17C9 MUUSIKATOAN to be rendered with the glyph of U+17BB VOWEL SIGN U (in some fonts dependent on base consonant).
17D2 179A 17D2 17xx	◌្រ្ក etc.	17D2 17xx 17D2 179A	◌្ក្រ etc.	For any 17xx in [1780-17B3]. Fonts generally don't distinguish between a sequence of *coeng ro* followed by another *coeng* and the sequence of that other *coeng* followed by *coeng ro*.
17A3	ឣ	17A2	អ	Already in UTS #39 data files.
17A4	ឤ	17A2 17B6	អា	
17D8	៘	17D4 179B 17D4	។ល។	
1791 17D2 17A1	ទ្ឡ	17A1	ឡ	Identical representation in many fonts.

Source	Rendering	Target	Rendering	Comments
17D2 178A	◌្ដ	17D2 178F	◌្ត	See discussion of *coeng ta* and *coeng da* below.

Other confusables

Many of the confusable pairs below are easily distinguished when looking at a high-resolution rendering with a good font in isolation, but easily confused when looking at a low-resolution screen (which are still common in Cambodia) with expectations derived from the context.

Source	Rendering	Target	Rendering	Comments
1791 17D2 1794	ទ្ប	17A1	ឡ	Subtle visual difference
17D2 17A1	◌្ឡ	17D2 1794	◌្ប	
17D3	◌៓	030A	◌̊	
17C6	◌ំ	030A	◌̊	
17C8	◌ៈ	003A	:	See discussion of *yuukaleapintu* below.
17BE 17D2 1799	◌ើ្យ	17BF	◌ឿ	
17D2 1799 17BE	◌្យើ	17BF	◌ឿ	
17C0	◌ៀ	17C1 17D2 1799	◌េ្យ	
17AB	ឫ	1794 17D2 1789	ប្ញ	
17AD 17B6	ឭា	1789	ញ	
17AE 17B6	ឮា	1789	ញ	
1796 17B6 17D2 1789	ពា្ញ	1789	ញ	
17AD	ឭ	1796 17D2 1789	ព្ញ	
17B0	ឰ	1796 17D2 178B	ព្ឋ	
17D2 1792	◌្ធ	17D2 178B	◌្ឋ	
17AA	ឪ	17B1	ឱ	
17B3	ឳ	17B1	ឱ	
17A7 17B7	ឧិ	17B1	ឱ	
17A7 17CC	ឧ៌	17B1	ឱ	
17A7 17CD	ឧ៍	17B1	ឱ	
17A7 17CA	ឧ៊	17A8	ឨ	
17A7 17D1	ឧ៑	17A8	ឨ	
1789 17D2 179C	ញ្វ	1796 17D2 179C 17B6	ព្វា	
17E8 17D3	៨៓	19E0	᧠	

Coeng ta and *Coeng da*

In modern Khmer script, coeng da and coeng ta look identical[12]. One might expect that users know which one to enter and there should be no problem. But, as Appendix 1 shows, confusion is rife. Users regularly type the wrong one. Since they look identical, it is not possible for a user to see how they have mistyped just by looking at the text. This issue arises every so often in the Khmer press: "Method for distinguishing coeng da and coeng ta to avoid confusion" and "Beware confusing coeng da and coeng ta". One solution would be for a keyboard input method to analyze the context and store the right coeng even if a user typed the wrong one. The problem is that the rules are too complex, with too many special cases for a keyboard to do that.

Because there are two ways of encoding the same visual representation here, we need to resolve this. The easiest solution is to just use a single underlying code for the two coengs. There are no minimal contrasts between the two coengs and therefore there is no need for contrastive encoding. Since both coengs look identical, it is impossible for someone looking at a text to tell whether two different encodings or a single encoding has been used. When it comes to keyboard entry, the keyboard still allows users to type a coeng da or coeng ta as before. The only difference is that it stores one of the two possible encodings, and the user is none the wiser. The user does not have to change their behavior and they have the advantage that they cannot type the wrong one by accident.

The preferred encoding is that **both coeng da and coeng ta are stored as `17D2 178F` (coeng ta).** The storage of coeng da (`17D2 178A`) is not disallowed in Middle Khmer where there is a visual contrast. But where the text is known to be Modern Khmer, coeng da should never be stored, in favor of coeng ta. Keyboards for Modern Khmer should only output coeng ta and never coeng da. For Old or Middle Khmer where there may be a visual contrast, then both may be used, but fonts need to show a visual contrast between the two.

Other processes working on Khmer text, such as spelling checkers and text-to-speech systems, will have to be aware of the linguistic ambiguity that has been introduced by folding the two characters together.

We propose to change the representative glyph for *coeng da* in TUS section 16.4, table 16-8, to an older form, as Kent Karlsson already proposed in L2/20-174. However, because current fonts generally follow the Unicode Standard and use the same glyph for *coeng ta* and *coeng da*, we also need to treat the two as intentional confusables. This has already been proposed in L2/22-108.

Yuukaleapintu and Colon

These two characters are very similar and widely confused. To help distinguish them, Khmer users commonly insert a space before the colon. Keyboards should do this automatically to help users avoid ambiguity.

The UTS #39 data files already record numerous other characters as confusable with colon. Yuukaleapintu should be added.

[12] Also raised in https://www.unicode.org/L2/L2020/20174-pubrev.html.

Transition

We would like to eventually see the new orthographic syllable structure applied wherever text in the Khmer script is created or interpreted, from keyboards and fonts to search tools and natural language processing systems. However, at present, a lot of Khmer text exists already, and systems generate and interpret text based on the various existing Khmer orthographic syllable structures. We therefore need a transition plan that gets us from the current situation to the desired new situation.

For the most part, the new orthographic syllable structure is a subset of the existing ones. That is, text that conforms to the new structure generally also conforms to the old ones and will, for example, render without dotted circles and according to user expectations. On the other hand, the new structure disallows some character sequences that the old ones allowed, so that, if and when the new structure is enforced by rendering systems and fonts, some old text will render with dotted circles.

There are some cases where the new structure requires character sequences that the old ones did not foresee:

- The new rules for consonant shifters mean that a consonant shifter can follow a consonant that does not have the series expected for that shifter, and still require downshifting (the consonant with the expected series would occur earlier in the cluster). There may be fonts that check whether consonant and consonant shifter agree, and only downshift when that's the case. In addition, the ICANN root zone label generation rules for Khmer require agreement between consonant and consonant shifter.
- The new rules for final coengs mean they are not supported by any fonts. We selected the position of ZWJ between conjoiner and base character because current OpenType implementations do not insert dotted circles for it. However, fonts are not designed to form coengs from conjoiner-ZWJ-base sequences, and don't move the resulting coengs to after the vowel.
- The new rules support multiple vowels. Clearly these are not used in Modern Khmer, but are needed for Middle Khmer. The sequences are constrained to not require any more visual complexity than is already needed for Modern Khmer.
- Minority languages often reappropriate one or two diacritics for use as vowels. They then need to combine these with actual diacritics. The result is the need to store up to two diacritics.

We propose the following transition plan:

1. Enable key text consumers to accept Khmer text encoded according to the new orthographic syllable structure:
   - Update any fonts that check for agreement between a consonant shifter and the series of the immediately preceding consonant shifter to remove such checks.
   - Update the ICANN root zone label generation rules for Khmer to require agreement between a consonant shifter and the series of the entire sequence of preceding consonants, as described in the section Consonant Shifters.
   - Update fonts designed for Middle Khmer to support the new encoding of final coengs, by forming coengs from conjoiner-ZWJ-base sequences and moving the resulting coengs to after the vowel.
2. Transition text producers to generate Khmer text encoded according to the new orthographic syllable structure:

    - Develop smart keyboards / input methods and dictionaries used for predictive input that generate Khmer text encoded according to the new orthographic syllable structure.
    - Update other text producers, such as OCR systems or translation software, to generate Khmer text encoded according to the new orthographic syllable structure.
3. Enable text consumers to verify text according to the new orthographic syllable structure on an opt-in basis:
    - Add support for Khmer using the new orthographic syllable structure to the OpenType Universal Shaping Engine, using a new script tag “khm1”. When using fonts with the new script tag, text that does not conform to the new orthographic syllable structure will have dotted circles inserted.
    - Create fonts (likely initially variants of existing Khmer fonts) that use the new script tag.
    - Use the new fonts in text editing components that are dedicated to creating new text, such as the input fields in messaging apps or in web forms.
    - Update spelling checkers to verify text according to the new orthographic syllable structure.
4. Convert existing text that needs to be maintained long-term to the new orthographic syllable structure.
5. Eventually, change text consumers to verify text according to the new orthographic syllable structure by default:
    - Use fonts that use the new script tag by default in operating systems and apps.

The following sections provide more detail on some of these steps.

Compatibility

There are two approaches that can be taken in transitioning correct data to the new structure:

- Require data to be transformed because currently correct data is not correct in the new structure.
- Change the new structure, if needed, to ensure that existing correct data does not need to be transformed, where possible.

The latter is much preferred, because requiring the re-encoding of all existing data would be an unacceptably costly activity. Therefore we need to compare the new structure with the existing. We only do this for Modern Khmer and minority orthographies. Middle Khmer with its final coengs is fundamentally incompatible with the new structure and there is no harmonizing the two. This is discussed elsewhere, particular with regard to final coengs. We can take a simplified view of the new structure:

```
SylN = B R? C{0,2} S? V? MS? MF? | O
```

There are a number of existing syllable structures both in standards and implementations. We examine them in terms of the classes and structures already defined. The mappings are not precise, but sufficient for our analysis. SylE is the specification from the introduction. SylU is the current Unicode standard (v14). SylMS is the Microsoft Khmer shaper specification. SylH is the Harfbuzz Khmer shaper implementation.

```
SylE =  B (R | (ZWNJ? SC)) C* (ZWNJ? SC)? (ZWNJ? V)? (MS | MF)? (ZWJ C)?
SylU =  B (R | SC) (C R?)* (Z? V)? (MS | MF)? C?
SylMS = B C{0,2} (VP | VB)? SC? VA? MS? VF? MF?
SylH =  B Z? (SC | R)? (C (Z? (SC | R))?)*
        (ZMS* VP? ZMS* VB? ZMS* (Z? VA)? ZMS* VF? ZMS* C? MF* | 17D2)
```

```
Z = [200C 200D]                    # ZW(N)J
ZMS = (Z* MSNR)                    # ZW(N)J plus modifier sign
R = 17CC                    # Robat
SC = [17C9 17CA]            # Shifter Character
```

From the description above, the SylMS syllable structure should not support the strings that its implementation actually does. We assume that the specification is out of date with respect to the implementation. Lindenberg (2019) provides a more detailed survey of how implementations actually validate Khmer syllables, at that time.

In order to decide whether we can transition without having to transform existing good data, we need to examine the intersection between these syllable structures and the one defined earlier. Are there required strings that are not in that intersection? There are a number of places:

- Unicode doesn't currently allow a consonant shifter after a coeng. But it is unique in that.
- The Microsoft shaper allows a consonant shifter only after a pre or below vowel. This is a very strange place to put one.
- ZWNJ is stored before the consonant shifter.

Incompatibilities

Shifter order: very often a shifter is stored before the coeng when it is now required after.

Data Transition

The following classes of sequences that have been used in the past become illegal under the new regular expression:

- Digit coeng kahn sequences are no longer valid, the corresponding lunar symbols from the U+19E0-U+19FF block are to be used. For example, the sequence ០្។ <17E0 17D2 17D4) was often rendered the same as ᧠ <19E0>. The new structure does not allow for the legacy sequence and rendering will break.
- Coeng ro occurring first in a sequence of coengs are required to be stored second.

Normalization

Here we describe normalization actions (not normalization as defined in UAX #15) where sequences in currently conformant Unicode orders are folded into a single sequence using the structure described here.

Reference Implementation

The enclosed reference implementation provides two functions. One fuction, khnormal, normalizes Modern Khmer text to the encoding described here in such a way that it looks the same as the input text. Thus if there is bad spelling in the original (for example inappropriate multiple vowels), this code does not fix that or mark an error, it simply passes it on for other processes to handle appropriately. A second function, khtest, tests whether a string is conformant to the Khmer encoding structure, returning false if it does not.

Khnormal does resolve some confusables, in particular only the "do not use" sequences are replaced with their equivalents. As a result the output string is never longer than the input string.

The reference implementation is in python3 using only core modules and is written to be easily translatable into other languages. The code is not written for speed, but for clarity.

The core regular expression module (re) does not support variable length negative look behind assertions. We refactor them into positive look behind assertions.

```
#!/usr/bin/python3
# Copyright (c) 2021-2022, SIL International.
# Licensed under MIT license: https://opensource.org/licenses/MIT

import enum, re

class Cats(enum.Enum):
       Other = 0; Base = 1; Robat = 2; Coeng = 3; ZFCoeng = 4
       Shift = 5; Z = 6; VPre = 7; VB = 8; VA = 9
       VPost = 10; MS = 11; MF = 12

categories =  ([Cats.Base] * 35    # 1780-17A2
              + [Cats.Other] * 2   # 17A3-17A4
              + [Cats.Base] * 15   # 17A5-17B3
              + [Cats.Other] * 2   # 17B4-17B5
              + [Cats.VPost]       # 17B6
              + [Cats.VA] * 4      # 17B7-17BA
              + [Cats.VB] * 3      # 17BB-17BD
              + [Cats.VPre] * 8    # 17BE-17C5
              + [Cats.MS]          # 17C6
              + [Cats.MF] * 2      # 17C7-17C8
              + [Cats.Shift] * 2   # 17C9-17CA
              + [Cats.MS]          # 17CB
              + [Cats.Robat]       # 17CC
              + [Cats.MS] * 5      # 17CD-17D1
              + [Cats.Coeng]       # 17D2
              + [Cats.MS]          # 17D3
              + [Cats.Other] * 9   # 17D4-17DC
              + [Cats.MS])         # 17DD

khres = {   # useful regular sub expressions used later
       "B":   "[\u1780-\u17A2\u17A5-\u17B3\u25CC]",
       "NonRo":   "[\u1780-\u1799\u179B-\u17A2\u17A5-\u17B3]",
       "NonBA":   "[\u1780-\u1793\u1795-\u17A2\u17A5-\u17B3]",
       "S1":  "[\u1780-\u1783\u1785-\u1788\u178A-\u178D\u178F-\u1792"
              "\u1795-\u1797\u179E-\u17A0\u17A2]",
       "S2":  "[\u1784\u1780\u178E\u1793\u1794\u1798-\u179D\u17A1\u17A3-\u17B3]",
       "VAA": "(?:[\u17B7-\u17BA\u17BE\u17BF\u17DD]|\u17B6\u17C6)",
       "VA":  "(?:[\u17C1-\u17C5]?{VAA})",
       "VAS": "(?:{VA}|[\u17C1-\u17C3]?\u17D0)",
       "VB":  "(?:[\u17C1-\u17C3][\u17BB-\u17BD])",

       # contains series 1 and no BA
       "STRONG":  "{S1}\u17CC?(?:\u17D2{NonBA}(?:\u17D2{NonBA})?)?|"
              "{NonBA}\u17CC?(?:\u17D2{S1}(?:\u17D2{NonBA})?|\u17D2{NonBA}\
u17D2{S1})",

       # contains BA or only series 2
       "NSTRONG": "(?:{S2}\u17CC?(?:\u17D2{S2}(?:\u17D2{S2})?)?|\u1794\u17CC?
{COENG}?|"
              "{B}\u17CC?(?:\u17D2{NonRo}\u17D2\u1794|\u17D2\u1794(?:\u17D2{B}))?)",
       "COENG":   "(?:(?:\u17D2{NonRo})?\u17D2{B})",

       # final right spacing coeng
       "COENGR":  "(?:(?:[\u17C9\u17CA]\u200C?)?(?:{VB}?{VAS}|{VB}))",

       # final all coengs
       "COENGF":  "(?:(?:[\u17C9\u17CA]\u200C?)?[\u17C2-\u17C3]?{VB}?{VA}?"
              "[\u17B6\u17BF\u17C0\u17C4\u17C5])",
       "COENGS":  "(?:\u17C9\u200C?{VAS})",
       "FCOENG":  "(?:\u17D2\u200D{NonRo})",
       "SHIFT":   "(?:(?<={STRONG}{FCOENG}?)\u17CA\u200C(?={VA})|"
              "(?<={NSTRONG}{FCOENG}?)\u17C9\u200C(?={VAS})|[\u17C9\u17CA])",
       "V":   "(?:\u17C1[\u17BC\u17BD]?[\u17B7\u17B9\u17BA]?|"
              "[\u17C2\u17C3]?[\u17BC\u17BD]?[\u17B7-\u17BA]\u17B6|"
              "[\u17C2\u17C3]?[\u17BB-\u17BD]?\u17B6|\u17BE[\u17BC\u17BD]?\u17B6?|"
              "[\u17C1-\u17C5]?\u17BB(?![\u17D0\u17DD])|"
              "[\u17BF\u17C0]|[\u17C2-\u17C5]?[\u17BC\u17BD]?[\u17B7-\u17BA]?)",
       "MS":  "(?:(?:[\u17C6\u17CB\u17CD-\u17CF\u17D1\u17D3]|"
              "(?<!\u17BB[\u17B6\u17C4\u17C5]?)[\u17D0\u17DD])"
              "[\u17C6\u17CB\u17CD-\u17D1\u17D3\u17DD]?)"
}

# expand 2 times: CEONGS -> VAS -> VA -> VAA
for i in range(3):
       khres = {k: v.format(**khres) for k, v in khres.items()}

def charcat(c):
       ''' Returns the Khmer character category for a single char string'''
       o = ord(c)
       if 0x1780 <= o <= 0x17DD:
              return categories[o-0x1780]
       elif o == 0x200C:
              return Cats.Z
       elif o == 0x200D:
              return Cats.ZFCoeng
       return Cats.Other

def khnormal(txt, lang="km"):
       ''' Returns khmer normalised string, without fixing or marking errors'''
       # Mark final coengs in Middle Khmer
       if lang == "xhm":
              txt = re.sub(r"([\u17B7-\u17C5]\u17D2)", "\\1\u200D", txt)

       # Categorise every character in the string
       charcats = [charcat(c) for c in txt]

       # Recategorise base or ZWJ -> coeng after coeng char
       for i in range(1, len(charcats)):
              if txt[i-1] in "\u200D\u17D2" and charcats[i] in (Cats.Base,
Cats.ZFCoeng):
                     charcats[i] = Cats.Coeng

       # Find subranges of base+non other and sort components in the subrange
       i = 0
       res = []
       while i < len(charcats):
              c = charcats[i]
              if c != Cats.Base:
                     res.append(txt[i])
                     i += 1
                     continue
       # Scan for end of syllable
       j = i + 1
       while j < len(charcats) and charcats[j].value > Cats.Base.value:
              j += 1

       # Sort syllable based on character categories
       # Sort the char indices by category then position in string
       newindices = sorted(range(i, j), key=lambda e:(charcats[e].value, e))
       replaces = "".join(txt[n] for n in newindices)

       replaces = re.sub("(\u200D?\u17D2[\u17D2\u200C\u200D]+",
                            r"\1", replaces)     # remove multiple invisible chars
        replaces = re.sub("\u17BE\u17B6", "\u17C4\u17B8", replaces)  # confusable
vowels
```

```
       # map compoound vowel sequences to compounds with -u before to be converted
       replaces = re.sub("\u17C1([\u17BB-\u17BD]?)\u17B8", "\u17BE\\1", replaces)
       replaces = re.sub("\u17C1([\u17BB-\u17BD]?)\u17B6", "\u17C4\\1", replaces)
       replaces = re.sub("(\u17BE)(\u17BB)", r"\2\1", replaces)
       # Replace -u + upper vowel with consonant shifter
       replaces = re.sub("({STRONG}{FCOENG}?[\u17C1-\u17C5]?)\u17BB"
                            "(?={VAA}|\u17D0)".format(**khres), "\\1\u17CA", replaces)
       replaces = re.sub("({NSTRONG}{FCOENG}?[\u17C1-\u17C5]?)\u17BB"
                            "(?={VAA}|\u17D0)".format(**khres), "\\1\u17C9", replaces)
       replaces = re.sub("(\u17D2\u179A)(\u17D2[\u1780-\u17B3])",
                            r"\2\1", replaces)   # coeng ro second
       replaces = re.sub("(\u17D2)\u178A", "\\1\u178F", replaces)  # coeng da->ta
       res.append(replaces)
       i = j
       return "".join(res)

def khtest(txt):
       ''' Tests normalized text for conformance to Khmer encoding structure '''
       import regex
       syl = regex.compile("({B}\u17CC?{COENG}?(?:\u17D2\u200D(?={COENGR})|{FCOENG}"
       "(?={COENGF})|(?<={NSTRONG})\u17D2\u200D{S1}(?={COENGS}))?{SHIFT}?{V}{MS}?"
       "[\u17C7\u17C8]?|[\u17A3\u17A4\u17B4\u17B5]|[^\u1780-\u17D2])".format(**khres))
       res = []
       passed = True
       while len(txt):
       m = syl.match(txt)          # match a syllable
       if m:
              res.append(m.group(1))  # add matched syllable to output
              txt = txt[m.end(1):] # update start to after this syllable
              continue             # go round for the next syllable
       passed = False                            # will return a failed string
       m = syl.match("\u25CC"+txt)               # Try inserting 25CC and matching
that
       if m and m.end(1) > 1:
              res.append(m.group(1))      # yes then insert 25CC in output
              txt = txt[m.end(1)-1:]
       else:
              res.append("!{}!".format(txt[0]))   # output failure character
              txt = txt[1:]
       if not passed:              # if the output is different, return it
               return "".join(res)
       return None                 # return None as sentinal for pass

if __name__ == "__main__":
       import argparse, sys

       parser = argparse.ArgumentParser()
       parser.add_argument("infile",nargs="+",help="input file")
       parser.add_argument("-o","--outfile", help="Output file")
       parser.add_argument("-u","--unicodes",action="store_true")
       parser.add_argument("-f","--fail",action="store_true",
                     help="Only print lines that fail the regex after normalising")
       parser.add_argument("-l","--lang",default="km",help="Language specific
processing")
       args = parser.parse_args()

       if args.unicodes:
              instr = "".join(chr(int(x, 16)) for x in args.infile)
              res = khnormal(instr, lang=args.lang)
              If argos.fail:
                     res = khtest(res)
```

```
                if res is not None:
                        print(" ".join("{:04X}".format(ord(x)) for x in res))
        else:
                infile = open(args.infile[0], encoding="utf-8") if args.infile[0] != "-"
\
                                        else sys.stdin
                outfile = open(args.outfile, "w", encoding="utf-8") if args.outfile \
                                        else sys.stdout
        for l in infile.readlines():
                res = khnormal(l, lang=args.lang)
                if args.fail:
                        tested = khtest(res)
                        if tested is not None:
                                outfile.write(tested)
                else:
                        outfile.write(res)
```

Shaping and Font Development

A primary concern of the font is that there be a visual distinction between any two different strings. While the regular expression describes what constitutes a 'legal' sequence, it gives no indication of how illegal sequences are to be indicated. Where should a dotted circle be inserted? This section does not distinguish responsibilities between the shaper and the font. In effect it says that ensuring good rendering, including marking of errors, is the responsibility of the font, supported as it may be, by a shaping engine. The reason for this is that at the time of writing, shaping support for Khmer can be very different across different shapers and it is unknown what shaping support will be available to fonts in the future.

An important consideration is the difference in shaping needs between Modern Khmer and Middle Khmer. Thankfully, much of the Middle Khmer encoding structure is now incorporated into the main encoding structure, including such things as:

- Vowel sequences
- Final coengs

There are Middle Khmer sequences that may not be supported by the proposed encoding structure and would be considered illegal by that sequence. It is beyond the scope of this document to identify them and decide what to do about them. But in general there are two approaches to handling such sequences:

1. Extend the structure to support the extra sequences. This is problematic since it may involve code changes and the waiting for all applications to update. Font designers will also need to support the extra needs of Middle Khmer even if they do not want to, just to stay conformant to the encoding structure standard. Any extensions to this structure must ensure that the common use case of Modern Khmer is not weakend and ambiguities introduced by the Middle Khmer sequence needs.
2. Add special shaping rules for Middle Khmer that are identified by tagging the text for the language. This can be done either by language tagging (marking the text for language) or font tagging (selecting an appropriate font for text in Middle Khmer).

Language Tagging

Some systems facilitate the tagging of text for language well. For example Microsoft Windows always marks all text for language on entry, whether the user is aware of it or not. Whether applications retain that information is another matter. Others, for example plain text, SMS, social media platforms; do not. Thankfully, Middle Khmer is not a living language and is unlikely to be used in social media. The language is usually stored for analysis or in a rich text environment like a word processor. Thus language marking is an option.

Inside the font, it is possible to change behaviour based on language. But currently, no shaper implementations support changing the syllable structure based purely on a language contrast. Adding such support would require a compelling reason. Since the Middle Khmer user community is small and specialist, it is unlikely that such a development will be carried out for this situation.

Font Tagging

It is not uncommon to set the font of a text. This is particularly true given the nature of the use of Middle Khmer text where the text is clearly identified as such even in plain text contexts (through out of band knowledge of the contents of a file or some other data structuring). The limitation is that a font can either support Middle or Modern Khmer but not both. The preferred way OpenType supports different shaping rules (or syllable structures) is through the use of different

script tags. While a font may provide support for multiple script tags, it makes little sense for a font to support both Middle and Modern Khmer script tags given the shaper has to choose one of the two. Probably in this case it would choose the least common: Middle Khmer.
The difficulty here is that since the user community for Middle Khmer is so small, it is very probable that it would take a long time to find bugs in the Middle Khmer shaper. The turnaround for fixing bugs in a shaper is very long and outside the control of a font developer or the user community. So another alternative is for the Middle Khmer community to use the Modern Khmer shaper, complete with it inserting 'error' dotted circles, and then for the font to remove these error glyphs appropriately. This requires a change in the implementation of the shaping engine to insert the error glyphs before executing GSUB features rather than after GSUB and before GPOS. It also requires in effect a standardised approach to error marking of Khmer text.

Shaping

The description here does not attempt to use any existing shaping specification. Instead it outlines the needs of a shaper without specifying the actual implementation.

The primary issue in creating a font is to ensure that no two different strings look the same. This is a shared responsibility between the shaping engine and the font. There is a tendency to not want to overburden the shaping engine with a lot of detailed rules, but equally to provide helpful support to the font.

The full disambiguating regular expression is large and complex. But it is unlikely that the regular expression will be implemented directly, and would be expressed in code in a different way. This is especially true since most regular expression engines cannot handle variable length zero width look behind assertions. That is one way of identifying 'illegal' strings. But another way allows for a more nuanced insertion of error marks. This approach is to do a very basic syllable analysis and then use negative rules to identify and mark bad strings.

USE Implementation

Any text that conforms to the structure specified in this document, will render appropriately using current smart font shaping engines, in particular Harfbuzz. The encoding structure is within the current bounds of existing encoding structures.

The difficulties arise regarding text that does not conform to the structure specified here. Such text should be rendered in such a way as to indicate that the encoding of that text is not appropriate for the rendering the user intended. In short, text that does not conform to this structure should render to show some kind of error. No shaping engine does this sufficiently today.

Consonant Shifters

As an example, let's consider the consonant shifter. The regular expression for this particular part of the string is certainly complex. How might a font (in conjunction with a shaper) handle this? It already has to address such sequences to identify when the shifter downshifts, so perhaps we can do all of the work of downshifting and error marking together. The examples below will use the FEA syntax and presumes no shaper support.

For the sake of this example, we will identify all base forms by their unicode value with glyph names such as `u179A`. Notice that at this point we are dealing with glyphs and not codepoints. In addition, every consonant has a coeng form: the glyph that is used when the consonant is preceded by a coeng (`17D2`) and this is named as a variant glyph: thus `u179A.coeng`. We also presume that each of the classes in the regular expression has a corresponding class of glyphs. If the class is preceded by a coeng (`17D2`), then the class is simply a class of `.coeng` forms. Thus:

```
@SCS = (u1784.coeng u1789.coeng u1799.coeng u179A.coeng u179C.coeng u179D.coeng)
@SS = (u1784 u1789 u1798 u1799 u179A u179C u179D)
```

The task of the lookup we are to write is to process `u17C9` and `u17CA`. There are two actions we can take: we can convert the shifter into a -u vowel (`u17BB`) or insert a dotted circle before it (`u25CC`). There is also a lookup to strip the zwj or zwnj. In the case of zwnj, if it is legal, then it is downshifting and we replace it with a -u glyph, whereas with zwj we leave the consonant shifter in place.

```
lookup shiftu {
    sub u17C9 by u17BB;
    sub u17CA by u17BB;
} shiftu;

lookup shifterr {
    sub u17C9 by u25CC u17C9;
    sub u17CA by u25CC u17CA;
} shifterr;

lookup shiftstrip {
    sub u200D u17C9 by u17C9;
    sub u200D u17CA by u17CA;
    sub u200C u17C9 by u17BB;
    sub u200C u17CA by u17BB;
} shiftstrip;
```

These lookups are ‘called’ from a contextual chaining lookup that uses strings of glyphs to call the appropriate lookup on the shifter. Because of the size of this lookup here, we only consider u17CA. Handling u17C9 is left as an exercise for the reader. We also leave out the u17B6 u17C6 vowel context after the first few example rules.

```
lookup doshift {
    # unmarked downshift
    sub @SF u17CA’ lookup shiftu @VS;
    sub @SF u17CA’ lookup shiftu u17B6 u17C6;
    sub @B @SCF u17CA’ lookup shiftu @VS;
    sub @B @SCF u17CA’ lookup shiftu u17B6 u17C6;
    sub @B @SCF @C u17CA’ lookup shiftu @VS;
    sub @B @SCNF @SCF u17CA’ lookup shiftu @VS;

    # ZWNJ stops downshift
    sub @SF u17CA’ u200C’ lookup shiftstrip @VS;
    sub @B @SCF u17CA’ u200C’ lookup shiftstrip @VS;
    sub @B @SCF @C u17CA’ u200C’ lookup shiftstrip @VS;
    sub @B @SCNF @SCF u17CA’ u200C’ lookup shiftstrip @VS;

    # ZWJ should not occur
    sub @SF u17CA’ u200D’ lookup shifterr @VS;
    sub @B @SCF u17CA’ u200D’ lookup shifterr @VS;
    sub @B @SCF @C u17CA’ u200D’ lookup shifterr @VS;
    sub @B @SCNF @SCF u17CA’ u200D’ lookup shifterr @VS;

    # we only get here if all the other rules fail, so our classes can be vague.
    # explicit ZWJ downshifts
    sub @B u17CA’ u200D’ lookup shiftu @VS;
    sub @B @C u17CA’ u200D’ lookup shiftu @VS;
    sub @B @C @C u17CA’ u200D’ lookup shiftu @VS;
    # explicit ZWNJ should not occur
    sub @B u17CA' u200C' lookup shifterr @VS;
    sub @B @C u17CA' u200C' lookup shifterr @VS;
    sub @B @C @C u17CA' u200C' lookup shifterr @VS;
} doshift;
```

Tests

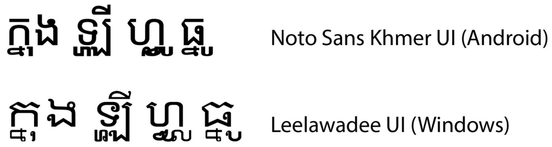
- Ensure coeng sequences are contrastive based on order (reversing the order should result in a different rendering)
- Coeng ta and coeng da should render contrastively.
- If a coeng ro is stored first in a coeng sequence, the font should show a visual contrast (however ugly that may be) compared to the coeng ro coming second.
- If a consonant shifter precedes a spacing coeng, the font (or ideally the shaper) should show a visual contrast compared to when the consonant shifter is after the coeng.

Design Issues

Coeng Stacking

The stacking of below diacritics is a particular design issue for Khmer fonts. Long stacks of say two coengs and a lower diacritic vowel do exist, but they are rare. A typesetter does not want to allocate lots of space below the baseline just for the rare cases that may occur once or twice in a book. Therefore there is a desire by font designers to want to reduce the height of the lower diacritic stack. For example, stacking lower diacritics horizontally across the bottom of the base character. But this flies in the face of readability where readers need to know the order of the coengs in order to read well. In addition, fonts need to show coeng order so that they show a visual distinction for a different character sequence.

If a font designer wants to design with horizontal stacking, they need to ensure that reversing the order of the coengs in the stack leads to a different visual representation. Such stacks occur with borrowed words. Thus 'Florida': ហ្វ្លរីដា[13]? ហ្លវរីដា? ផ្លរីដា? Or 'free': ហ្វ្រី, or 'three': ស្រី. Another good example is ទឡ្ហីករណ៍.



Some other examples of the problems of combining coengs show the problems that users face on various platforms.

Independent Vowels and Coengs

The definition of bases includes independent vowels. This means that independent vowels should be able to take coengs. But some of the independent vowels have below baseline components which are problematic when combining with coengs. But there are no known cases of such sequences occurring in any use of the Khmer script. Thus font designers are free to make such sequences render poorly, either through collision or inserting a dotted circle, for example. The only independent vowel and coeng sequences which occur are [`17B1 17B2`] `17D2` [`1799 1798`]. This problem also applies to `17A1` although there are a rare examples of coengs applying to `17A1`: ទឡ្ហីករណ៍. It is unknown whether Middle Khmer has independent vowels with coengs.

[13] Note that this document was written in Google docs and there are no known fonts available in Google Fonts that render these words correctly.

Final Coengs

Final coengs, marked by a following ZWJ (`200D`), must visually contrast with non-final coengs in their positional relationship with the vowel.

Modifier Stacking

The syllable structure allows there to be 2 modifier signs in a sequence. Fonts should ensure that such signs do not overlay each other. In particular if two identical signs occur a font must ensure that they are both visible and distinct from each other.

Acknowledgements

The research and development of this document was done because a number of technical experts saw key technical problems with the Khmer encoding. This document is written to provide input both to the Unicode Technical Committee and also to the Cambodian Government, in particular the Cambodia Academy of Digital Technology[14]. Thanks are due to SIL International and the National Polytechnic Institute of Cambodia Language Software Development Unit (NPIC LSDU) for their authorship. Martin Hosken's research is also undertaken under the auspices of the Payap University Linguistics Institute, ChiangMai Thailand. Most of the underlying research was undertaken in conjunction with Makara Sok and Didi Kanjahn and based on the encoding proposed by Javier Solá (Open Institute). Thanks are also due to the following people for their technical contributions and review: Peter Constable, Sovichet Tep, Nathan Willis. Trent Walker (Stanford University) reviewed and contributed to the section on Middler Khmer.

Further thanks is given to Lyheang Ung and Hour Kaing, of the Cambodia Academy of Digital Technology, who reviewed the document in detail, and to the National Council of Khmer Language (NCKL) of the Royal Academy of Cambodia (RAC) for clarification on some Khmer linguistic matters.

[14] បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា https://www.cadt.edu.kh/about/board-of-trustees/.

Appendix 1 - Current Confusion

The problem that the Khmer script faces is that there is often more than one way to encode a visual string and it be valid, and yet only one way is the ‘correct’ way (whatever that might be). For those very experienced in understanding how the Khmer script is stored in Unicode, and with a strong linguistic awareness, there seems to be no problem. But for many users there are problems. Here we present some simple statistics taken from page counts in Google for different encodings of the same word.

Example Words

‘Woman’

The word ស្ត្រី `woman` consists of 4 characters. The initial consonant is not in confusion. The coeng ta is confusible with coeng da and the 3 characters following the base consonant may occur in any order. The result is 12 possible ways of encoding the word. They are listed here along with the number of pages Google found of that spelling, in popularity order. Notice that the most popular spelling is not the ‘correct’ spelling. The best that someone searching for this word can hope for is 53% by typing their search term wrongly.

Text	Romanised	Popularity
ស្រ្តី	srti	8,950,000
ស្ត្រី	stri	4,950,000 (correct spelling)
ស្រ្ដី	srdi	1,340,000
ស្ដ្រី	sdri	893,000
ស្រី្ត	srit	620,000
ស្តី្រ	stir	25,900
ស្រី្ដ	srid	19,600
ស្ដី្រ	sdir	6,190
សី្ត្រ	sitr	10
សី្ដ្រ	sidr	3
សី្រ្ត	sirt	1
សី្រ្ដ	sird	1

‘Detect/investigate’

ស៊ើប ‘to detect/investigate’ consists of a confusable vowel (ើ 17BE) and a consonant shifter (៊ 17CA) which is down shifted to be a glyph which looks like that of 17BB in this context. There are 15 possible ways of typing this word. The two most noticeable errors are (1) 17BE is perceived as a combination of two separate vowels, i.e. 17C1 and 17B8 and (2) 17CA is thought to be 17BB because of how it looks.

Text	Sequences	Popularity
ស៊ើប	ស ៊ ើ ប	759,000 (correct spelling)
សើុប	ស ើ ុ ប	38,500
ស៊េីប	ស ៊ េ ី ប	11,800
ស៉ើប	ស ៉ ើ ប	11,600 *
សេីុប	ស េ ី ុ ប	6,410
សុើប	ស ុ ើ ប	5,120
សេុីប	ស េ ុ ី ប	30
សុេីប	ស ុ េ ី ប	21
ស៊ីេប	ស ៊ ី េ ប	14
សើ៉ប	ស ើ ៉ ប	11 *
សុីេប	ស ុ ី េ ប	8
ស៉េីប	ស ៉ េ ី ប	8 *
ស៉ីេប	ស ៉ ី េ ប	6 *
សីេុប	ស ី េ ុ ប	5
សីុេប	ស ី ុ េ ប	3

Entries marked with * are not automatically fixed by the reference normalization code.

‘Eat’

Here, there is considerable downshifting confusion.

Text	Sequences	Popularity
ស៊ី	ស ៊ ី	3,250,000 (correct spelling)
សុី	ស ុ ី	3,040,000
ស៉ី	ស ៉ ី	632,000 *
សីុ	ស ី ុ	400,000
សី៊	ស ី ៊	364
សី៉	ស ី ៉	9 *

Entries marked with * are not automatically fixed by the reference normalization code.

‘Bread’

More downshifting confusion with a less common word.

Text	Sequences	Popularity
បុ័ង	ប ុ ័ ង	34,700
ប៉័ង	ប ៉ ័ ង	26,400 (correct spelling)

ប័ុង	ប ◌័ ◌ុ ង	16,700

‘One sort’

For the most part people realise that the downshifter goes after the coeng.

Text	Sequences	Popularity
ម្យ៉ាង	ម ◌្យ ◌៉ ◌ា ង	2,530,000 (correct spelling)
ម៉្យាង	ម ◌៉ ◌្យ ◌ា ង	480,000

‘Don’t’

People tend not to have a problem with typing a vowel before a final.

Text	Sequences	Popularity
កុំ	ក ◌ុ◌ំ	6,050,000 (correct spelling)
កំុ	ក ◌ំ ◌ុ	284,000

‘Wait’

People tend not to fall into the trap of typing ‘am’ the Thai way.

Text	Sequences	Popularity
ចាំ	ច ◌ា◌ំ	5,860,000 (correct spelling)
ចំា	ច ◌ំ ◌ា	208,000

‘Mr’

Most people only use one vowel for split vowels.

Text	Sequences	Popularity
លោក	ល ◌ោ ក	8,190,000 (correct spelling)
លេាក	ល ◌េ ◌ា ក	422,000
លាេក	ល ◌ា ◌េ ក	15,600

‘Khmer’

People do well at spelling the language name correctly.

Text	Sequences	Popularity
ខ្មែរ	ខ ◌្ម ◌ែ រ	29,100,000 (correct spelling)
ខែ្មរ	ខ ◌ែ ◌្ម រ	372,000

Coeng Ta vs Coeng Da

Here we see some example words and their various popularities for the confusion between coeng ta and coeng da. In each case the ‘correct’ spelling is emboldened.

ដ spelling	ដ popularity	ត spelling	ត popularity
កណ្ដាល	**2,120,000**	កណ្តាល	3,550,000
ក្ដី	**1,660,000**	ក្តី	3,560,000
ស្ដាំ	**445,000**	ស្តាំ	903,000

While students are taught the correct usage of coeng ta and coeng da, it does not mean that the lessons are always remembered into adulthood.

Appendix 2 - Middle Khmer

In order to arrive at a syllable structure that can support all data needs, it is necessary to analyse all the orthographies that use Khmer script. For the most part, the minority language use of the Khmer script is conservative with few innovations. Any such innovations are discussed in the main text. But there is one other major set of orthographies that use the Khmer script and these are often lumped under the simplified names of Old Khmer and Middle Khmer, even though they cover many different orthography and language stages in the development of Khmer from Old Khmer in the 7th-8th century, Angkor Khmer and Middle Khmer. Upon examination of these different orthographies, it is the Middle Khmer period, leading into the Modern Khmer period, that requires the greatest complexity in encoding. During this period, there is a growth in orthographic complexity along with little or no standardised spelling.

The concern here is to ensure that they can be adequately encoded. Again the principles of a single encoding for a single graphical representation are necessary. This carries with it an implication that the encoding does not need to directly represent the linguistic structure of the syllable. For example, syllable final coengs (following a vowel) do not necessarily have to be encoded after the vowel.

The analysis here can only be considered a sketch that is looking for particular exemplars that might challenge the proposed encoding structure. A much more thorough analysis would be required to come anywhere close to a full description of the orthography of Middle Khmer.

The first step in such an analysis is to find examples of words that do not fit the currently proposed Khmer encoded syllable structure and see what they say to us. There may be a way to make them fit into the current model or they may call for a change of the model. Care must be taken that the needs of historic data do not overwhelm the far more prevalent modern data. The conversation needs to be one of mutual respect and concern.

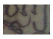
The analysis in this section was undertaken before the final form of the main syllable was agreed. Thus its argumentation may not flow well and has been placed as an appendix.

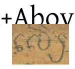
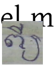
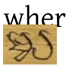
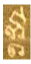
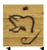
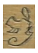
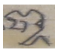
Final Coengs

Final coengs occur after the vowel as a final sound in a phonetic syllable and they are often visually indistinguishable from prevowel coengs. There are some contexts in which the final coeng is visually contrastive with a before vowel coeng and some not. The table shows - if data exists and is not visually contrastive between the coeng occurring before or after the vowel, and + if it is visually contrastive. ? Indicates no examples have been found.

Coeng/Vowel	Left	Left+Above	Left+Right	Right	Above	Below
Coeng Left	-	?	?	-	-	-
Coeng Below	-	-	?	+	-	?
Coeng Right	-	-	+	+	+	+

We examine some examples in detail:

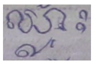
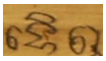
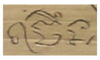
1. Where the vowel is spacing, final spacing coengs are clearly contrastive. Consider:  ហ្ឫ្‍យា `17A0 17D2 179B 17D2 200D 1799 17B6`. Thus we do need to encode spacing final coengs in conjunction with most vowels. Spacing coengs have no visual contrast following pre vowels.

2. Left+Above is non contrastive regarding final coengs: Compare 179B 17D2 1799 17BE and  179B 17BE 17D2 1799[15]. These may be contrastive in a font, but are non-contrastive in handwriting.
3. In Modern Khmer, a spacing coeng may take a vowel over it, indicating the vowel comes after the coeng: ព្យី 1796 17D2 1799 17B8. In Middle Khmer, the vowel may render over the consonant, indicating the spacing coeng comes after the vowel:  ព្‍យី 1796 17D2 200D 1799 17B8. Therefore there is a visual contrast between an initial and a final spacing coeng with an above vowel. Where the coeng is initial, the vowel occurs over the coeng, slightly to the right of the base. For a final coeng the vowel is positioned directly over the consonant. Not all fonts show this, but that is because in modern Khmer there is no contrast necessary. However,for a Middle Khmer font, such contrast is required.
4. For lower vowels (-u, -uu) the visual contrast is akin to that between multiple coengs, where the -u/-uu vowel interacts just like a non-spacing coeng would in a coeng sequence:  ម‍្យូ 1798 200D 17D2 1799 17BC. For comparison, the corresponding Modern Khmer encoding of this sequence would be: ម្យូ 1798 17D2 1799 17BC, which looks different and would sound different. Therefore, we cannot simply say that the Modern Khmer spelling with the coeng before the vowel is sufficient.
5. There are final coengs that are indistinguishable from the corresponding coeng before the vowel: ក‍្ដី 1780 200D 17D2 178A 17B8, which is visually indistinguishable from the corresponding Modern Khmer encoding: 1780 17D2 178A 17B8. There is one base for which there is a visual distinction: ញ‍្ងំ 1789 200D 17D2 1784 17C6 contrasts with the Modern Khmer encoding: ញ្ងំ 1789 17D2 1784 17C6. Typically the lower swash under the base is removed in Modern Khmer when a coeng occurs after it. But this swash is not removed if the coeng comes after a vowel.
6. With a spacing vowel, a non-spacing coeng may or may not have contrastive positioning. For example:   ទាំ្ង 1791 17B6 17C6 17D2 1784. These two examples are not contrastive with the second being a stylistic variant of the first and so the Modern Khmer encoding of 1791 17D2 1784 17B6 17C6 is sufficient. But  ប‍្ទា 1794 200D 17D2 1791 17B6 is contrastive and so a contrastive encoding is necessary.

Given the extended encoding structure for Modern Khmer, final coengs are not in principle a problem and most of these cases are adequately supported. The only problem is number 5 where a non contrastive coeng sequence becomes contrastive for final coengs due to the swash. Further analysis is required to decide whether the presence of the swash is in fact the primary indicator of a final coeng in contrast to a non final coeng or merely a stylistic feature of the swash never disappearing in the presence of a lower diacritic (coeng or vowel).

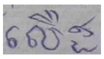
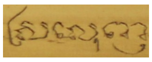
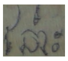
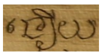
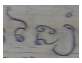
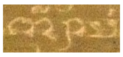
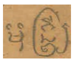
Multiple Vowels

While Unicode has encoded all the modern Khmer vowels as units, there are vowel combinations in Middle Khmer that are not available as a unit in Unicode.
These need reviewing since most of them only have one occurrence and come from OCR[16].

3.	ឈ្លុាះ	1788 17D2 179B 17BB 17B6 17C7	
4.	ទេិព	1791 17C1 17B7 1796	
5.	ច្រេឹន	1785 17D2 179A 17C1 17B9 1793	

[15] Although I am unsure how the OCR recognised a difference in encoding.

[16] See Valy D., et al.

6.	លេឺង	179B 17C1 17BA 1784	
7.	ស្រលេុញ	179F 17D2 179A 179B 17C1 17BB 1789	
8.	នៃាះ	1793 17C2 17B6 17C7	
9.	នៀីយ	1793 17C0 17B8 1799 (or 1793 17BF 1799)	
12.	នៃ្យំ	1793 17C3 17D2 1799 17C6	
13.	ពា្យាប់	1796 17B6 17D2 1799 17B6 1794 17CB	
14.	មុន្្រត្‍យ	1798 17BB 1793 17D2 17D2 179A 178F 17D2 200D 1799	

Example 13 is noticeable because, in effect, it has two syllables in one, with two vowels separated by a coeng. This raises the spectre of fully chained syllables. Whereby a new full syllable is built on a final coeng. There are various options in how to address this:

- Fully recursive definition whereby the final coeng is also treated as the initial of another full syllable. Thus FinalC would be something like `CF Rhyme?`.
- Chain syllables by allowing the final coeng sign character (17D2) to hang as the last of the previous syllable, then its consonant becomes the start of the next syllable.
- List out all the possible options for a limited single extra syllable.

The first option is hard to implement. Recursive regular expressions are nigh on impossible to work with. The second option is merely a way of removing the recursion while keeping the full chaining. The third option is easiest to implement but only supports a very limited following syllable structure with no further chaining. Given there is no evidence of any more extensive chaining than the last example above, we choose the simplest option. For this we allow final spacing coengs to take some limited vowel or signs.

Example 14 shows two initial coengs and a final coeng.

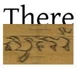
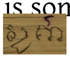
Chuon Nath uses a double pre vowel 17C1 17C1 to represent the corresponding Thai vowel. There are two options on how to handle this:

1. Add 17C1 17C1 to the syllable structure
2. Encode a new codepoint for the double letter as per Thai.

The first is a minimal engineering cost. The second involves a full proposal. While both are possible and acceptable solutions, solution 2 is simpler than upsetting the encoding structure in ways that could confuse Modern Khmer users.

Diacritics

Robat

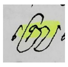
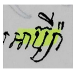
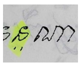
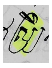
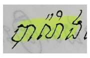
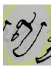
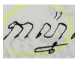
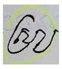
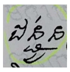
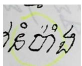
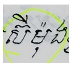
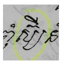
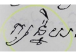
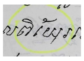
There is some disagreement about where Robat (17CC) renders in relation to the -aa vowel (17B6):   សួគា៌ 179F 17BD 1782 17B6 17CC, which does not help in deciding whether the robat should go early in the sequence or late. It seems this example is stylistic or even a slipped pen. We propose that nothing needs to be done about it.

Triisap

In addition to its function as a consonant shifter, in Middle Khmer triisap may also represent the consonant ង (`1784 NGO`) super joined above another base. This is akin to robat or kinzi in Myanmar script. Thus សឃ៊ (`179F 1783 17CA`) may be written more conventionally as សង្ឃ (`179F 1784 17D2 1783`). A triisap used in this way is not treated any differently to a normal triisap. Thus if the context calls for the triisap to shift down, it will, unless a ZWNJ is inserted to prevent that happening.

More Examples

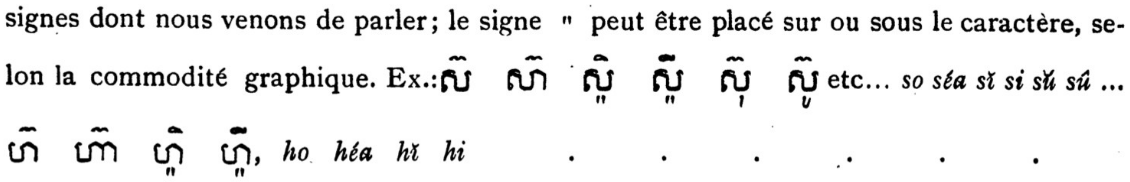
Handwritten 1899 Luke[17]

lk2a		ក្រ‍្យោ	1780 17D2 179A 200D 17D2 1799 17C4	
lk2b		អាប‍្យីា៉	17A2 17B6 1794 200D 17D2 1799 17B8 17B6 17C9	Syllable chaining with muusikatoan on the ‘final ya’. This is currently unsupported and a font would need to handle this especially.
lk3a		ន៉ណា	1793 17C9? 178E 17B6	Variant downshifted muusikatoan or a double vowel. Not currently supported by Unicode
lk3b		ស្រ‍្យី	179F 17D2 179A 200D 17D2 1799 17B8	
lk4		បាល៉ាំង	1794 17B6 179B 17C9 17B6 17C6 1784	Muusikatoan on ល and stays up
lk6		ឥ្យ	17A5 17D2 1799	
lk8		កាល្ន៉ា	1780 17B6 179B 17D2 1793 17C9 17B6	
lk43a		ស្រ‍្យា	179F 17D2 179A 200D 17D2 1799 17B6	
lk43b		ជន្ជូ់ន	1787 1793 17D2 1787 17BC 17CB 1793	Novel context for bantoc with a double acting final.
lk50		ប៉‌ាំង	1794 17C9 200C 17B6 17C6 1784	Muusikatoan stays up
lk63		ស៊‌ីម៉ុង	179F 17CA 200C 17B8 1798 17C9 17BB 1784	Triisap stays up
lk104		ស៊‌ី្យ	179F 17CA 200C 17B8 17D2 1799	Triisap stays up
lk127		កូន្ន៉ើយ	1780 17BC 1793 17D2 1793 17C9 17BE 1799	Muusikatoan stays up
lk311		ពិយ៉ែរ	1796 17B7 1799 17C9 17C2 179A	Muusikatoan goes down

[17] BFBS 1899

Of particular note is lk43b. The bantoc (17CB) is rendered above the second base consonant to indicate that it is the final of the syllable and that, therefore, the following coeng is the start of the next syllable. Linguistically, therefore, the Bantoc comes before the coeng. But visually it comes after the sequence of base + coeng + vowel, in keeping with its position as a diacritic in the syllable structure. While this is not the most convenient place for it, linguistically, it is in an unambiguous position and its function, therefore, is clear. It marks the base consonant as a final.

lk3a shows an alternative form of the pushed down muusikatoan. The basis for this analysis comes from Bernard (Bernard, 1902) in which he states:



signes dont nous venons de parler; le signe " peut être placé sur ou sous le caractère, selon la commodité graphique. Ex.: ស៊ ស៊ា ស៊ិ ស៊ី ស៊ឹ ស៊ឺ etc... *so séa sĭ si sŭ sû* ...
ហ៊ ហ៊ា ហ៊ិ ហ៊ី, *ho héa hĭ hi* . . . . . .

An approximate translation states that the muusikatoan is sometimes placed above or below the character according to the graphical context. This implies that the lower double vertical stroke is merely a stylistic variant of a downshifted consonant shifter (since either shifter may resolve to this form) and is therefore in the realm of font variation. The problem here is that there is no upper vowel in this example. It is probably easiest to allow further analysis to decide precisely what is happening here and probably to propose a new character to give the appropriate result.

Khom Thai

The first writing system used for writing Thai was based on Khmer script. The 'Khom' (Thai for Khmer) Thai script has a long history which is not of concern here. The issue is how Khom Thai may be encoded using the Khmer Unicode block. It seems most natural to encode Khom Thai using Khmer characters due to the similarities in the writing systems. Many of the characters look very similar and Khom Thai has coengs which the Thai script does not. The main extensions needed are:

- Addition of Thai tones as diacritics, to Khmer. This can either be done through encoding extra characters or by allowing the sharing of Thai tone marks with the Khmer script using script extensions.
- Extra vowel combinations and/or extra characters.

There are a number of characters needed to encode Khom Thai in Unicode using the Khmer block. Analysis is required for these. As it stands a further Unicode proposal will be necessary to enable Khom Thai to be adequately encoded in Unicode. Such a proposal will also need to include any changes to the Middle Khmer orthographic syllable structure to ensure it facilitates the encoding of Khom Thai.

Conclusion

The encoding structure for Modern Khmer does a good job of supporting Middle Khmer. The difficulty with working with Middle Khmer is that writing in that period was both far from standardised and also a period of innovation. The result is that there will inevitably be strings found that do not fit within the encoding structure. It will be necessary then to decide how important it is that those strings are representable using the standard encoding structure. For example, there is only one known word where there is a follow on vowel after a final coeng. How important is it that this word is representable? The identified issues with the Modern Khmer encoding structure for Middle Khmer are:

1. Special casing of final coengs following 1789.
2. Vowel and following coeng after a final coeng.

Bibliography

Antelme, Michel, 2007 "INVENTAIRE PROVISOIRE DES CARACTÈRES ET DIVERS SIGNES DES ÉCRITURES KHMÈRES PRÉ-MODERNES ET MODERNES EMPLOYÉS POUR LA NOTATION DU KHMER, DU SIAMOIS, DES DIALECTES THAÏS MÉRIDIONAUX, DU SANSKRIT ET DU PĀLI ∗" (Projet “Corpus des inscriptions khmères” —CIK), url: https://web.archive.org/web/20190814131541/http://aefek.free.fr/iso_album/antelme_bis.pdf

Bernard, J. B., 1902 “Dictionnaire cambodgien-français” (HongKong)

British and Foreign Bible Society, 1899 “The Gospel According to Luke”, url: https://books.google.com.kh/books?id=ZrQUAAAAYAAJ

Chuon Nath, 1967 “Khmer Dictionary” (Buddhist Institute 5th ed, 1967/68), url: windows version http://krou.moeys.gov.kh/kh/article/item/1135-%E1%9E%9C%E1%9E%85%E1%9E%93%E1%9E%B6%E1%9E%93%E1%9E%BB%E1%9E%80%E1%9F%92%E1%9E%9A%E1%9E%98%E2%80%8B%E1%9E%81%E1%9F%92%E1%9E%98%E1%9F%82%E1%9E%9A.html#.YLmnD_kvPIU, online version http://dictionary.tovnah.com/help

Lindenberg, Norbert, 2019: Issues in Khmer syllable validation. (Lindenberg Software, 2019), url: https://lindenbergsoftware.com/en/notes/issues-in-khmer-syllable-validation/

Sok, Makara, 2016 “Phonological Principles and Automatic Phonemic and Phonetic Transcription of Khmer Words” (Payap University, MA Thesis, 2016), url: https://drive.google.com/file/d/1c_FXNy90pv06StsBMQz4Rzk87ulMqXyM/view?usp=sharing

Sok, Makara, 2021 “Khmer Character Specification/Usages” (SIL, 2021), url: https://github.com/sillsdev/khmer-character-specification/blob/master/specification.md#ApplicationofKhmerScripttoOtherLanguages

Solá, Javier, 2004 “Issues in Khmer Unicode 4.0” (Open Forum of Cambodia, version 2.0, 21/Oct/2004), url: https://sourceforge.net/projects/khmer/files/Documents%20about%20Khmer%20script/Documents%20about%20Khmer%20Script%20and%20about%20Khmer%20Unicode%20v1.0/IssuesInUnicode40-v2.0.pdf/download

Valy, D., Verleysen, M., Chhun, S., & Burie, J. C., 2017 “A New Khmer Palm Leaf Manuscript Dataset for Document Analysis and Recognition - SleukRith Set” (In 4th International Workshop on Historical Document Imaging and Processing (HIP), DOI 10.1145/3151509.3151510, Data: https://github.com/donavaly/SleukRith-Set).