

A Soyombo test case for the Universal Shaping Engine

These slides were first presented by John Hudson (Tiro Typeworks) at the 38th International Unicode Conference, as part of a larger, joint presentation with Andrew Glass (Microsoft) entitled *Shaping in the Post-Tofu Era*.^{*} The overall presentation consisted of an introduction to the background and business case for the new Universal Shaping Engine, followed by the Soyombo test case as illustrated in these slides, and then a live demo of the Soyombo font in use in a virtual machine running a build of the new engine with preliminary Soyombo data.

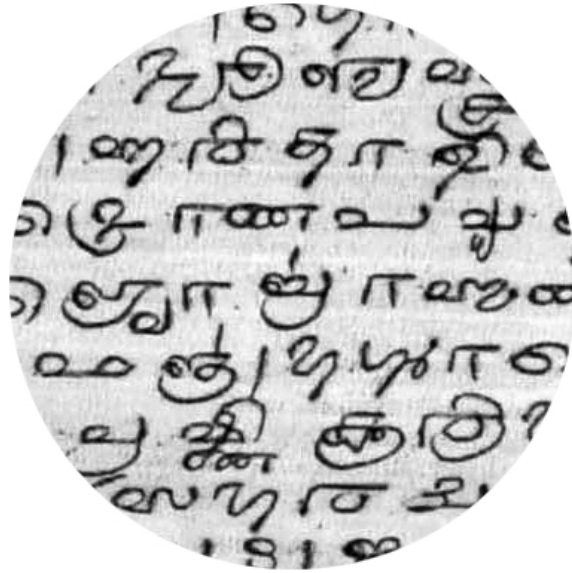
The Universal Shaping Engine (USE) is a new Unicode layout engine within the Microsoft Uniscribe layout component, introduced in the Windows 10 technical preview. The strategic goal of USE is to dramatically reduce the amount of time between the proposal of a script for encoding in the Unicode Standard and support for that script in software, which currently averages eight years. The business case for USE focuses on the diminishing returns on investment in dedicated shaping engines for individual scripts, especially for historical and minority scripts with few and sometimes impoverished users.

Unlike previous, script-specific shaping engines, the Universal Shaping Engine is based on a generic cluster model that requires no specialised knowledge of a particular writing system, but only a set of character property data from the Unicode Standard. This data is employed by USE to handle some aspects of cluster-level reordering—e.g. left-side vowels for Indic scripts—, but most of the responsibility for script behaviour resides with the font’s OpenType Layout (OTL) tables. In essence, this is a return to the original intent of the OpenType Layout architecture, in which script shaping behaviour is governed by the font lookups and their ordering, rather than by fixed feature ordering applied by the shaping engine.

At present, a small number of recently supported scripts are passed to the Universal Shaping Engine as implemented in the Windows 10 technical preview. In future, new OTL script tags could be defined to push Indic and other scripts to the new engine, enabling greater freedom for font developers in deciding how to handle layout.

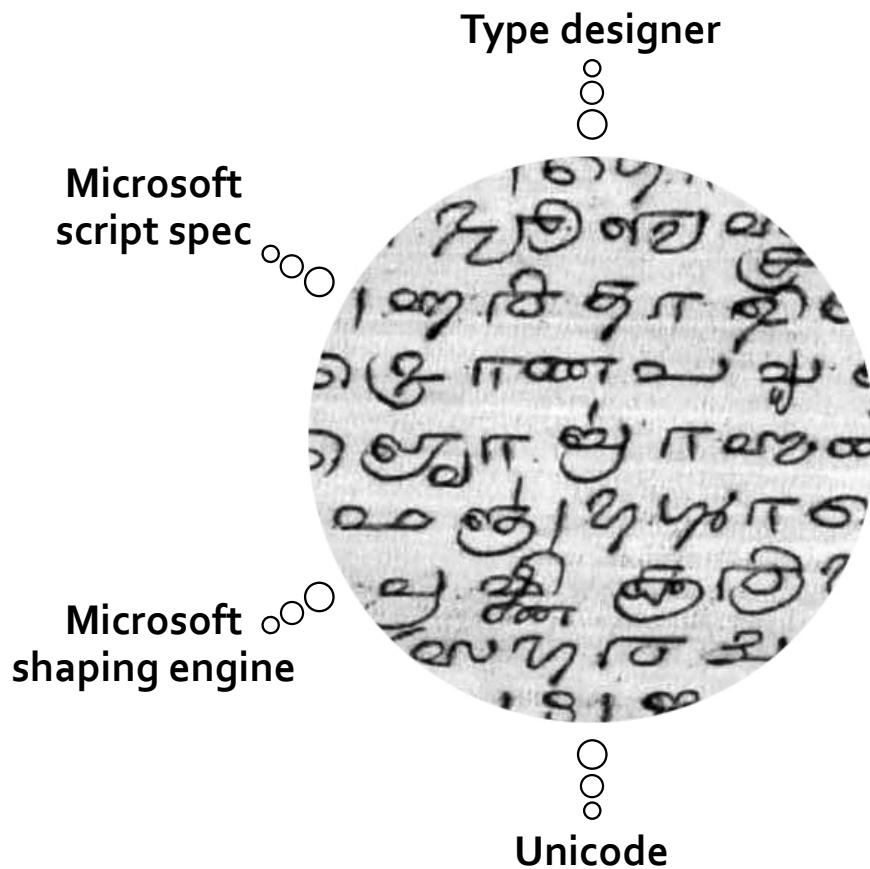
^{*} ‘Tofu’ is a term popularised by Google to refer to the .notdef glyph that represents unsupported characters when the selected font or fallback fonts do not contain glyphs for them. The .notdef glyph frequently consists of an empty rectangle □, hence ‘tofu’. The ‘post-tofu era’ implies a software environment where every character is supported but, as Andrew Glass’ part of this presentation stressed, for many writing systems simply displaying a glyph is inadequate.

Type designer



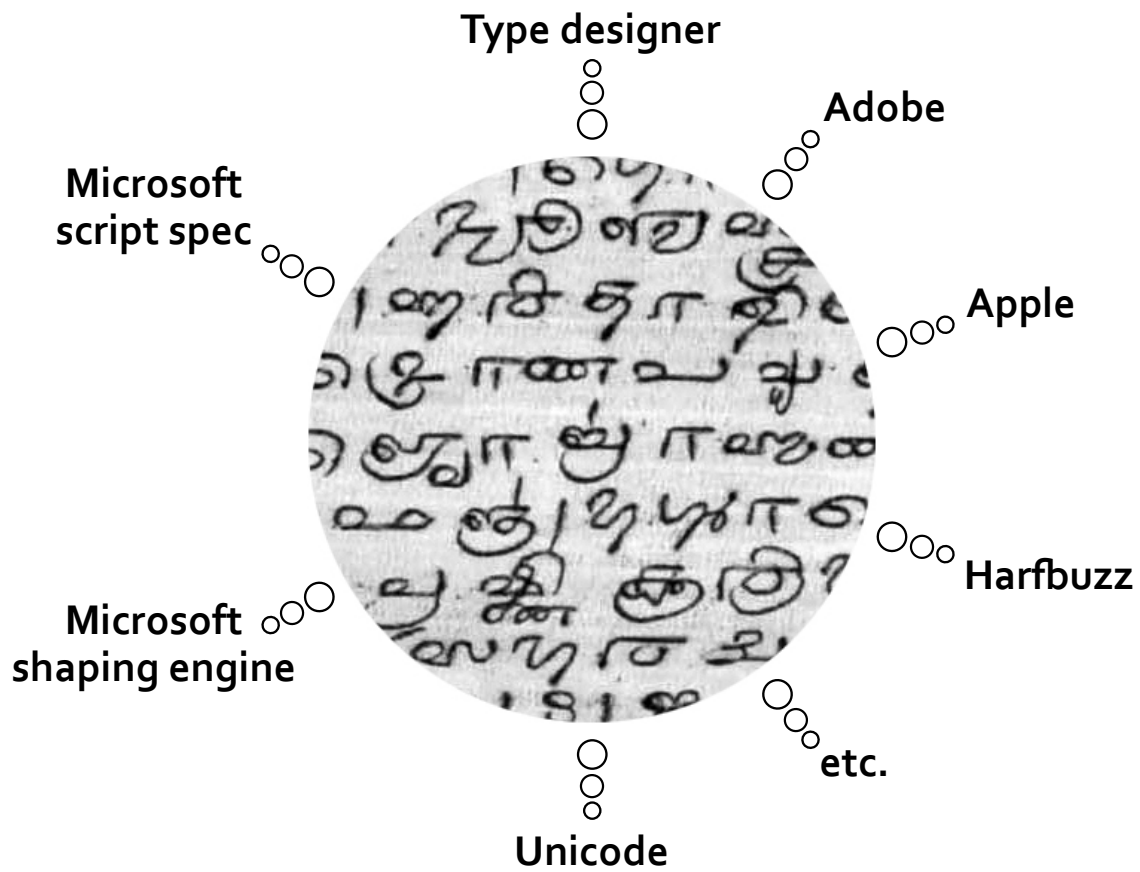
Unicode

Font development for complex scripts often involves reconciling different ways of thinking about a writing system. At the most basic level, there is the way I think about a script as a type designer, taking into account graphical and cultural aspects, and how the Unicode Standard thinks about a script, as embodied in how it encodes that script and what properties are assigned to its characters.

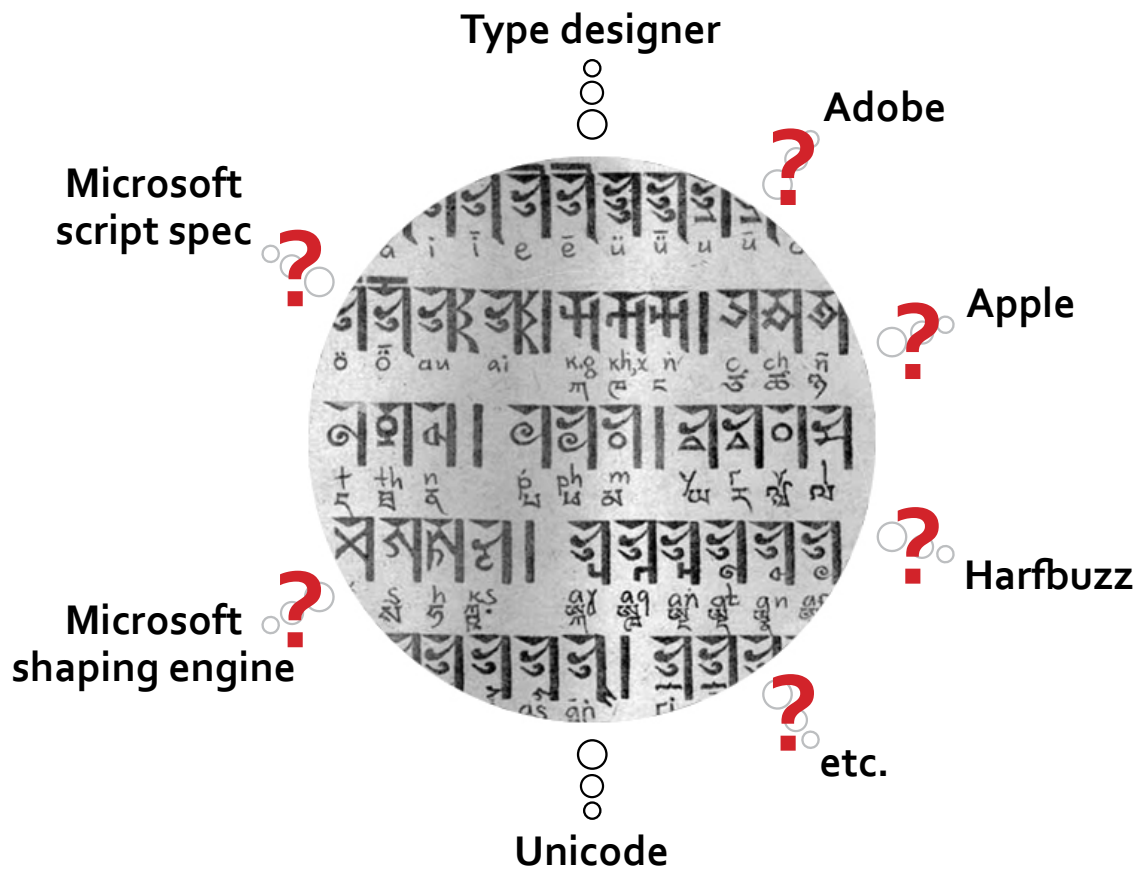


Until now, I've also needed to take into account how Microsoft's written specification thinks about the individual script, and how the shaping engine for that script actually works. *These don't always correspond.*

[In practice, the script specs have tended to be written after the fact, and tend to reference each other rather than record what the shaping engine is actually doing. This means, of course, that the specifications are sometimes unhelpful or misleading as a guide to making fonts that work correctly with the shaping engines.]



Of course, I also need to take into account how Adobe thinks about the script, how Apple thinks about it, and the makers of the open source Harfbuzz engine. Every once in a while I see evidence of some other layout process at work, some unknown engine. Unsurprisingly, since there has never been a specification for how to implement OpenType Layout, these various engines have incompatible behaviours for some scripts, which can make it difficult or impossible to make a font that works equally well in all software.



When it comes to a script that is newly encoded in Unicode, or one that is in the proposal stage, I simply don't know how any of these shaping engines is going to think about the script. This means that before I can start making a font with any reasonable hope that it will work, I need to wait for at least one of these shaping engines—usually Microsoft's—to support it, and then pray that others will do so compatibly, sooner rather than later, or at all.


This is one of the reasons why it can take many years for a script to make its way from the Unicode proposal stage to actually being supported in fonts and applications.



The Universal Shaping Engine greatly simplifies the situation. It is data-driven, which means that it can be easily and quickly updated when new scripts are added to Unicode. It relies on Unicode character property data, which means that the presumptions it makes about characters are documented and can be confirmed by the font developer as soon as the script encoding is published by the Unicode Consortium. The generic cluster model at the heart of the Universal Shaping Engine is also documented, meaning that it is possible for the font developer to make reliable predictions of the results of script shaping. Finally, because the engine puts few constraints on OpenType Layout feature and lookup ordering and structures, it is possible for the font maker to use tools like Microsoft's Visual OpenType Layout Tool (VOLT) and similar font development applications to test layout behaviour for a script even before the Universal Shaping Engine is updated with the new script data.

Soyombo an indigenous script of Mongolia
invented in 1686 by Öndör Gegēn Zanabazar (Jñānavajra)



Soyombo is an indigenous Mongolian script, invented in the 17th Century by Zanabazar, the spiritual leader of Tibetan Buddhism among the Kalkh Mongols. His name is a Mongolianisation of a Sanskrit word meaning 'He who has the thunderbolt of knowledge'. Soyombo text is easily recognised by the standard feature of a frame around each letter or orthographic syllable, consisting of a head triangle and a bar down the right side. The script is not in modern use, and has few competent readers. It remains important, though, as a cultural signifier in Mongolia, appearing on banners and prayer flags, and in the official symbol of the nation: 

The Soyombo script is not yet encoded in Unicode (as of November 2014). There have been two draft proposals prepared by Anshuman Pandey through the Script Encoding Initiative, and the Soyombo test case for the Universal Shaping Engine is based on the most recent of these. Needless to say, this font will not be finalised or released until the script is formally encoded and the standard codepoints and character properties confirmed.

-

11A84

iLA



[In the first part of the Unicode conference presentation, Andrew Glass had discussed, with diagrams, the generic cluster model. The slides now step through the elements of a Soyombo cluster, which can be understood as a subset of the generic cluster model used by the Universal Shaping Engine.]

The first element of a Soyombo cluster is an optional prescript initial consonant. There are four of these, each taking the form of a small sign on the upper left of the graphical cluster. This is illustrated here by the prescript *La*, written as a horizontal stroke. In Anshuman Pandey's most recent draft encoding proposal, these four initial consonants are atomically encoded, since the use of prescript vs full form is not orthographically predictable.

-
11A84
iLA

𑖅
11A5C
KA

𑖅

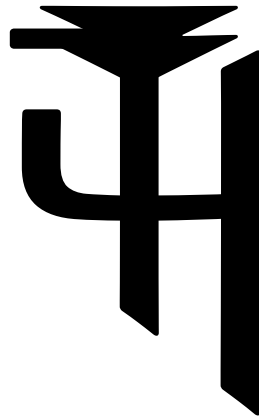
The second element of the cluster—and the only non-optional element—is the base letter, in this case *Ka* (Sanskrit; the Mongolian pronunciation is *Ga*). As in Brahmi-derived scripts, each consonant letter carries an inherent short *-a* vowel.

An independent vowel letter *A* (not shown) serves as a carrier for dependent vowel signs. In the draft encoding, only this one vowel letter is encoded, and other vowels represented by applying vowel signs to this base.

-
11A84
iLA

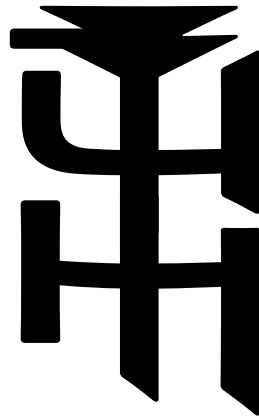
𑂔
11A5C
KA

𑂔̣
11A96
gemin.



The third element is an optional gemination sign. This is encoded as a triangular mark above the base letter, but is graphically represented by a merging of two triangles. Note that this slightly pushes down the base letter shape, and also affects the height of the top of the bar on the right.

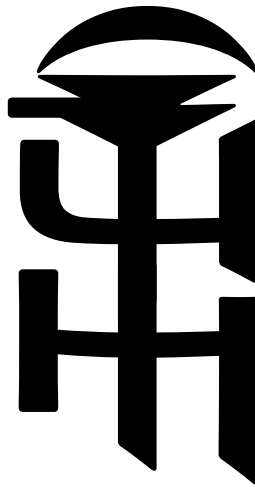
-	𑍑	◌̣	[SBJ]	𑍑
11A84	11A5C	11A96	11A97	11A60
<i>iLA</i>	<i>KA</i>	<i>gemin.</i>	<i>subj.</i>	<i>NGA</i>



The fourth element comprises two characters: a subjoiner control character and a second consonant letter. Graphically, the subscript letter is usually simply a lowered form of the base letter shape minus its triangle. In this case, though, the base letter is shortened vertically, and the sequence takes a special, broken form of the bar on the right.

A Soyombo syllable may include more than one subscript letter (not shown), each encoded as the subjoiner control character followed by the letter character.

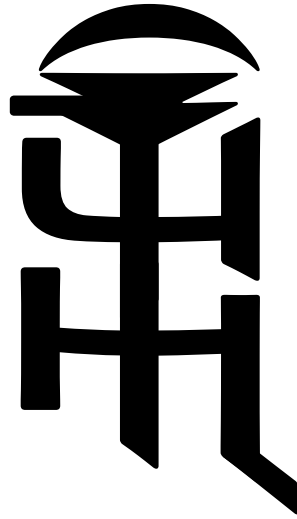
-	𑍑	◌̣	[SBJ]	𑍑	◌̣
11A84	11A5C	11A96	11A97	11A60	11A51
<i>iLA</i>	<i>KA</i>	<i>gemin.</i>	<i>subj.</i>	<i>NGA</i>	<i>sl</i>



The fifth element is an optional vowel sign, in this case the short *-i*. Vowel signs may appear at the top or bottom, or to the right of the cluster.

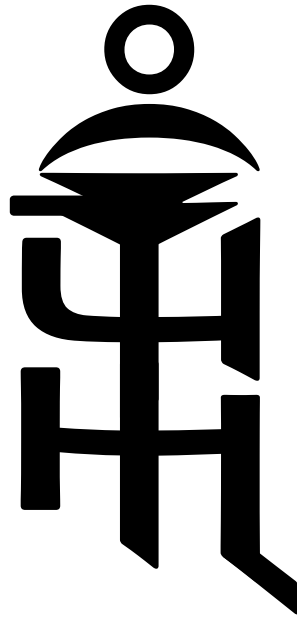
Soyombo clusters may include more than one vowel sign (not shown), to indicate a diphthong.

-	𑀓	𑀓̄	[SBJ]	𑀓	𑀓̄	𑀓̄
11A84	11A5C	11A96	11A97	11A60	11A51	11A5B
<i>iLA</i>	<i>KA</i>	<i>gemin.</i>	<i>subj.</i>	<i>NGA</i>	<i>sl</i>	<i>v.length</i>



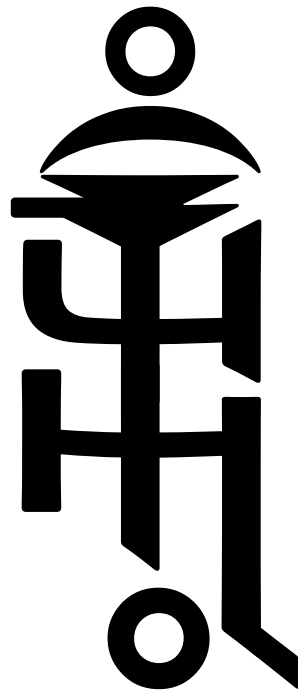
The sixth element is a vowel lengthening mark: a short diagonal stroke at the bottom of the bar on the right.

-	𑀓	𑀅	[SBJ]	𑀓	𑀆	𑀇	𑀈
11A84	11A5C	11A96	11A97	11A60	11A51	11A5B	11A94
<i>iLA</i>	<i>KA</i>	<i>gemin.</i>	<i>subj.</i>	<i>NGA</i>	<i>sl</i>	<i>v.length</i>	<i>nasal</i>



The seventh element is an optional nasalisation mark, an *anusvara*, which appears as a ring at the top of the cluster.

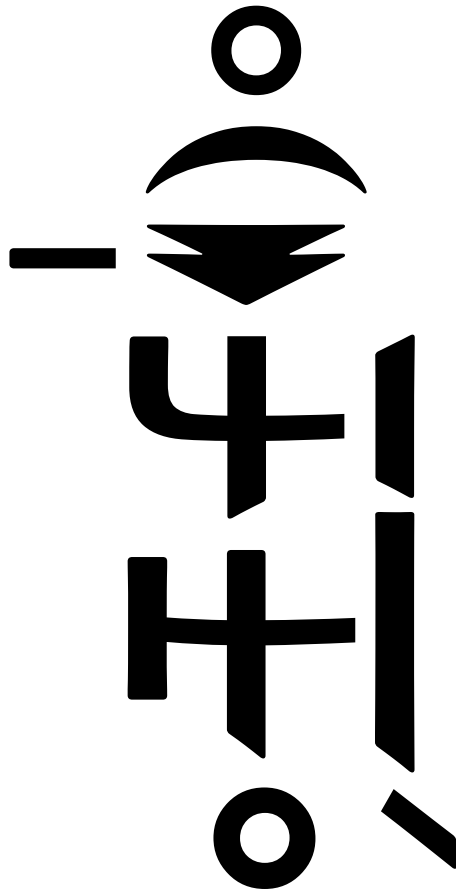
-	𑍑	◌̃	[SBJ]	𑍑	◌̂	◌̇	◌̈́	◌̈́
11A84	11A5C	11A96	11A97	11A60	11A51	11A5B	11A94	11A8E
<i>iLA</i>	<i>KA</i>	<i>gemin.</i>	<i>subj.</i>	<i>NGA</i>	<i>sl</i>	<i>v.length</i>	<i>nasal</i>	<i>fM</i>



The final element is an optional final consonant. Unlike Indic scripts, in which a syllable-terminating consonant sound is usually indicated by the presence of a *virama* ‘vowel-killer’ sign, Soyombo uses special syllable-final letters. These are typically small in size and sit close to, or connect with, the righthand bar. Note that the length of the bar on the right extends downwards to accommodate the final consonant within the frame.

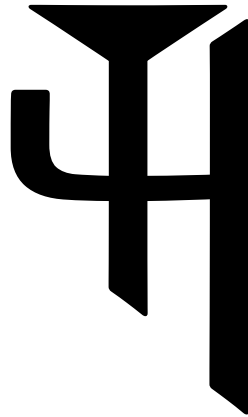
When displayed with a below vowel sign (not shown), the final consonant is vertically aligned with the vowel (typically of reduced width) and sits to its right.

So that’s how a Soyombo cluster is structured, and those are the elements that need to be displayed by a font. Looking at the overall typeform, it is possible to conceive of a number of ways this could be handled, from a highly inefficient precomposed glyph for each complete cluster, to some combination of precomposed base glyph plus combining marks. As I examined the graphical behaviour of the script, though, it seemed to me that the most flexible approach would be...



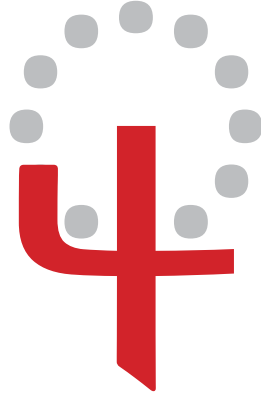
...to explode the cluster, to break it down into separate glyphs for each element, and then compose them dynamically, using OpenType GPOS anchoring. This enables shape details, such as the shape and height of the right bar, or of the upper triangle (there are three forms of simple and geminated triangle heads in the font), to be contextually varied in the font GSUB relative to adjacent shapes, sometimes across intervening contexts.

The following slides show the glyph processing for another sample cluster, and illustrate how I get from the text encoding to the display.



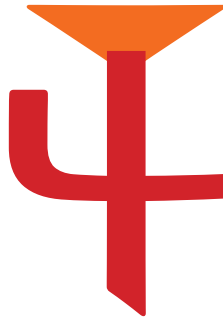
11A5C
SOYOMBO LETTER KA
• Mongolian ga

This is the Soyombo character *Ka* as we can expect it to appear in the Unicode Standard code chart. I think it's safe to say that for the vast majority of fonts for every script in Unicode, the default glyph for a character—*i.e.* the glyph mapped to that character in the font cmap table—basically corresponds, allowing for stylistic differences, to the representation published in the code chart.



11A5C
/soKa/

In my Soyombo font, though, this red shape is the encoded glyph for the *Ka* character in the font cmap table. The glyph represents only the distinctive portion of the letter, minus the frame components, and the glyph is treated as a combining mark (as indicated by the dotted circle, not part of the glyph). Note that this is only a glyph level mark assignment, recorded in the font GDEF table; at the character level, this remains a letter.



11A5C

/soTriangleA /soKa /

The first thing that happens in glyph processing is that the head triangle is inserted into the glyph string *before* the letter mark. This is done with a one-to-many substitution in the <ccmp> layout feature, and the shape of the triangle deployed depends on the letter mark.



11A5C 11A97 11A71
/soTriangleA /soKa.head /soPha.sub /

A subjoiner plus letter sequence is resolved to a subscript letter glyph using the <blwf> feature. Note that this triggers a contextual substitution of the base letter in the <pres> feature, activating a vertically shortened form; this is a behaviour of a small number of Soyombo letters.



11A5C 11A97 11A71
/soTriangleA /soKa.head /soPha.sub /soRightbarA2 /

At this stage, I insert the righthand bar component of the frame. This is actually done in a series of contextually controlled substitutions in the <rclt> feature that address the length of the bar, the shape of the top relative to the head triangle, and whether the bar is continuous or broken, depending on the letters in the stack.

Note that the bar length lookups include both backwards and forwards context strings to accommodate subscript vowels and syllable final consonants (not shown).



11A5C 11A97 11A71 11A58
/soTriangleA /soKa.head /soPha.sub /soRightbarA2 /soSignAi /

A vowel sign is added, in this instance a right-side -ai vowel sign.



11A5C 11A97 11A71 11A58 11A94
/soTriangleA /soKa.head /soPha.sub /soRightbarA2 /soSignAi /soAnusvara /

And finally an *anusvara* nasalisation sign is added above the cluster, and we have completed glyph processing for this cluster.



11A5C 11A97 11A71 11A58 11A94
/soTriangleA /soKa.head /soPha.sub /soRightbarA2 /soSignAi /soAnusvara /

The whole thing works because the only glyph in the whole sequence that has an advance width (indicated by the orange rectangle) is the head triangle. Every other glyph is treated as a combining mark, and most of these are directly anchored to that triangle in the GPOS <mark> feature. The subscript letter glyph is anchored to the preceding letter glyph, but everything else is positioned relative to the triangle. Since in some cases there are intervening glyphs between the triangle and the mark to be anchored, processing requires careful filtering of mark groups in the lookup flags.

In the GPOS <dist> feature, width is added to the right-side vowel mark (indicated by the blue line) so that the whole cluster has an advance width that avoids collision with adjacent clusters or punctuation. The <dist> feature can also be used to contextually kern clusters to each other based on wider or narrower elements in the letter stack.

ön - dör

ge - gēn

za - na - ba - zar

Here is a sample of the font in use, recording the name of Soyombo's inventor, Zanabazar. Note that the length of the righthand bar in each cluster is consistent across the line, determined by the deepest stack regardless of the depth of elements in the individual clusters. This is handled automatically for the line in the <calt> feature by filtering mark groups and not processing base glyphs in the lookup flags. This is very clever. Because line breaks terminate OpenType Layout glyph runs, it's only possible to automatically affect this kind of substitution at the line level, not the page or document level; some form of user-controlled bar length selection will be necessary for the latter.

It should be obvious from what has preceded that this is a font that is entirely dependent on shaping support to display in a legible manner. If shaping is not available, the results are not even minimally decipherable...

ᠣᠨ ᠳᠣᠷ

ön - dör

ᠭᠡ ᠭᠡᠨ

ge - gēn

ᠵᠠ ᠨᠠ ᠪᠠ ᠵᠠᠷ

za - na - ba - zar



splat!

This is perhaps better feedback that something has gone wrong than a less graphically drastic display, such as sometimes leads to the publication of examples of unshaped Arabic or Indic text.

I hope it is clear that this test case represents just one way to make a Soyombo font. It is the way that made most sense to me as I analysed the behaviour of the script and the kinds of graphical dependencies within each cluster. It is a way that allowed me to reconcile how I came to think about the writing system with how Anshuman's proposed Unicode encoding thinks about it, as translated through the predictable behaviour of the Universal Shaping Engine.

It's worth noting that at no stage in the development of the Soyombo font did I have access to a testing environment in which the Universal Shaping Engine was active. I built the font based on an understanding of the generic cluster model and how the shaping engine works. When the font was ready, Andrew installed it in his virtual machine, and it just worked.

Afterword & Acknowledgements

At the time of writing, documentation of the Universal Shaping Engine is being drafted. In order to be universal not merely in the sense of supporting any new script added to Unicode but also in enabling predictable and compatible font outcomes everywhere, the new layout model needs to be implemented by software developers beyond Microsoft. There is an encouraging degree of cooperation around OpenType Layout currently in evidence, as major software makers realise that the period of competition to support the world's major scripts has passed. There are still outstanding compatibility issues from that period, but despite my usual caution I am optimistic that the future will consist of better compatibility, increased predictability, and more freedom—and more responsibility—for font developers.

As noted in the introduction to these slides, the average delay between proposal of a script to Unicode and system support in software is currently eight years. Adding preliminary Soyombo character properties to the Universal Shaping Engine, designing and building the test font, and creating a test environment and Soyombo keyboard took about eight days.

With thanks to Andrew Glass and Ali Basit at Microsoft, and to Anshuman Pandey at the Script Encoding Initiative. Please consider financially supporting the work of the Script Encoding Initiative, which aims to have all the world's scripts included in Unicode.

John Hudson, 11 November 2014