Shaping Ideographic Description Sequences (v1) — TSWG 2020-01

Fredrick R. Brennan <copypaste@kittens.ph>
July 21, 2020

I have one main desire for text shaping: the creation of a standard for an OpenType table that implements a Chinese character description language. Despite the excellent and wonderful work done by Ken Lunde, Qin Lu, and many, many others over the years, it is still not possible to type an ideograph not in Unicode until it gets in Unicode, which is a long process for most ideographs. You might think, surely no popular ideograph is left out. Actually, many popular ideographs remain unencoded. Indeed, an ideograph so popular it has a lengthy description of itself on Wikipedia (not Wiktionary), 𰻞 (U+030EDE; Pinyin: *biáng*), has only been officially part of Unicode since 13.0. This year! It was only proposed in 2015, in L2/15-223.[1] I have been lamenting this in public since at least October 2018.[2]

Right now, thanks to a long series of proposals to the ISO/IEC JTC 1/SC 2 and UTC beginning with one from China on November 7, 1995, X3L2/95-111,[3] “Ideographic Structure Symbol (additional request)”, Ideographic Description Characters (IDC’s) exist.

We can see from an October 22, 1998 proposal, SC2 N3186, from Japan's liaison to SC2, “The IDS describes the ideograph in the abstract form. It is not interpreted as a composed character and does not have any rendering implication.” However, this has changed. Chapter 18, Section 2, page 735[4] of The Unicode Standard v13.0 reads in part: “An implementation may render a valid Ideographic Description Sequence either by rendering the individual characters separately **or by parsing the Ideographic Description Sequence**

[1] This document is for CJK Unified Ideographs Extension H, but U+030EDE ended up in Extension G.
[2] Brennan, Fredrick (2018). “Answer to ‘What are the disadvantages of typography?’”. Quora.
[3] This proposal seems unfortunately lost to time.
[4] In linked PDF, page 28.

and drawing the ideograph so described." [emphasis mine] There was a lot of precedent, for this, though. Indeed, John Jenkins was already doing it in 1999.[5]

Indeed, no less than Dr. Lunde himself has released fonts which do this, first being one just for *biáng* on March 24, 2014.[6] That font replaces the IDC sequence ⿺辶⿱穴⿰月⿰⿰⿱幺長⿱言馬⿱幺長刂心 with *biáng*. However, this is very much an "old-style" GSUB simple substitution, stuck in `ccmp`, and not what I propose. I propose extending OpenType shaping to make a best effort on arbitrary IDC's. Of course, fonts would need to opt in, and as the Unicode Standard says, encodings are always preferred to IDC's. Indeed, human designers adding `ccmp`'s are preferred, but unrealistic given an unlimited number of IDC's exist. I will save more details for later if requested, but here's a sketch.

The shaper needs to do three main things, using ⿰⿱口止⿱夂口 (路) as an example:

- Set up necessary glyph classes for all IDC's. Essentially, one class for the eleven IDC "functions", one mega class of all "normal" IDC arguments the font supports containing the variation that takes up the full ideograph. (口, 止, …) Then, mega classes containing the same number of glyphs for all IDC argument types. So, for 口:
    - 口_Plain.
    - ⿰: 口_Left; 口_Right.
    - ⿱: 口_Above; 口_Below.
    - ⿲: 口_LeftOf3; 口_HMiddleOf3; 口_RightOf3.
    - ⿳: 口_TopOf3; 口_VMiddleOf3; 口_BottomOf3.
    - ⿴: 口_Surrounding; 口_Surrounded.
    - ⿵: 口_SurroundingFromAbove; 口_SurroundedFromAbove.
    - ⿶: 口_SurroundingFromBelow; 口_SurroundedFromBelow.
    - ⿷: 口_SurroundingFromLeft; 口_SurroundedFromLeft.
    - ⿸: 口_SurroundingFromUpperLeft; 口_SurroundedFromUpperLeft.
    - ⿹: 口_SurroundingFromUpperRight; 口_SurroundedFromUpperRight.
    - ⿺: 口_SurroundingFromLowerLeft; 口_SurroundedFromLowerLeft.

[5] John H. Jenkins (1999). "New Ideographs in Unicode 3.0 and Beyond". San Jose, CA: 15th International Unicode Conference. pp. 12–21.
[6] Lunde, Ken (2014). "IDS + OpenType: Pseudo-encoding Unencoded Glyphs". Adobe, CJK Type Blog.

    - ⿻: 口_OverlaidA; 口_OverlaidB.
- Do necessary substitutions:
    - `sub ⿰ @_Plain' by @_Above;`
    - `sub @_Above @_Plain' by @_Below;`
    - `sub ⿵ ⿰ @_Above' @_Below' by @_AboveInTop @_AboveInBottom;`
    - … and so on.
        - It's immediately clear that there's some arbitrary recursion limit here, as we must define classes for all types, and in any event, make glyphs for them. This is where we must extend the world of normal OpenType for best results.
- On ⿵, defined with width 0, and with its two anchors, "L" and "R", attach ⿰_InLeft to "L" and ⿰_InRight to "R". On ⿰_InLeft, with its two anchors, "T" and "B", attach 口_TopInLeft to "T" and 止_BottomInLeft to "B". On ⿰_InRight, attach 夂_TopInRight and 口_BottomInRight. Voilà, 路.

It would be nice if we could determine the start and end of a discrete IDC describing one character as our font should be able to handle rows of them. ⿵ is essentially a "function" with two "position arguments". So, in C notation, `do {⿵(⿰(口, 止), ⿰(夂, 口))} while (0)`.[7]

It would also be nice if we didn't need so many classes. My friend Simon Cozens wrote a program called fontFeatures which implements a file format analogous to, but better than, Adobe Feature Files (.fea), called "FEE".[8] This gets us much of the way there, at least in terms of removing the tedium of writing these deeply nested class definitions like `@_LeftInRightInBottomInBottomOf3InOverlaidA`. 😵! Let's not forget our good friend *biáng*. In an OpenType font that could shape arbitrary IDS's, assuming it had no special glyph for *biáng*, (presumably, many such fonts would have special glyphs for common IDS's that look better,) and assuming that we change nothing about the standard to help, the glyph sequence would need to be so long I put it in its own section. (§ *Biáng* as input to a shaper)

[7] In case you forgot, a `do {} while (cond)` loop is guaranteed to run at least once regardless of `while` condition.
[8] Font Engineering (with) Extensibility

It would be nice if we could tell Fontconfig, CoreText, etc., what runs our font can actually handle, perhaps via a new OpenType name ID, (in the form of, I don't know, a PECL compatible regexp? Some kind of formal grammar? Comments welcome.), so we get skipped even if we have all the glyphs for a run but we can't handle it. We don't want Fontconfig to pass HarfBuzz a run our font can't handle. For example, imagine a simple version of the font with everything implemented up to a maximum recursion depth of 2. Fontconfig will just see we have glyphs for 𰻞, 𰻝, et cetera, and all the radicals, and assume we should handle the run. So, we will spit up junk back to the user when really we should've just been skipped.

It would be *very* nice if OpenType had an ability to do simple linear scales of referenced glyphs. If I'm not mistaken, PostScript Type 3 could. Now, of course, I'm *not* saying most classes should be implemented via scales. They shouldn't! However, we'd need far fewer glyphs if we could do even the most simple of rectangular scales of even up to ±20%. But, my ask is really, if we're dreaming big dreams: *allow us to define anchor classes of a glyph at a certain point along their linear interpolation gradient*. Imagine a variable font with an axis for each of the twenty-six basic ways to set any given ideograph with IDS's. That is to say, when I add an anchor to the base glyph, besides position, I can also set any or all of its variable font axes.

Biáng as input to an arbitrary IDS shaper

Note: This is just an example. Per the Unicode Consortium, IDS's should not be used for existing characters in documents once they have encodings.

Expected output: 

Input IDS:[9] ⿺辶⿱穴⿰月⿰⿰幺長⿱言馬⿰幺長刂心

List of glyphs after OpenType shaping:

```
⿺辶_SurroundingFromLowerLeft
    ⿱穴_TopOf3InSurroundedFromLowerLeft
        ⿰月_LeftOf3InHMiddleOf3InSurroundedFromLowerLeft
        ⿰⿰幺_AboveInLeftOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
           長_BelowInLeftOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
          ⿱言_AboveInVMiddleOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
           馬_BelowInVMiddleOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
          ⿰幺_AboveInRightOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
           長_BelowInRightOf3InVMiddleOf3InHMiddleOf3InSurroundedFromLowerLeft
        刂_RightOf3InHMiddleOf3InSurroundedFromLowerLeft
    心_BottomOf3InSurroundedFromLowerLeft
```

[9] A normalization step is required. Details TBD.