Variable OTF Fonts: Adding an interface default

Adobe, Inc.

July 26, 2023

1 The Issue

A central aspect of support for variable fonts in OpenType is the default location, which has a normalized value of `0` on every axis. Variable glyph outline data and other values (e.g. for spacing and kerning) are specified as deltas relative to the value at the default location.

Many TTF variable fonts, with `glyf` outlines and deltas stored in the `gvar` table, can still be used on systems that have not yet been upgraded to support variable font functionality. Indeed, the split between `glyf` and `gvar` was specified to allow this. When designing a glyph for this sort of use, one would typically position the most "general" font instance as the default. For example, in a font with one `wght` axis, the "Regular" weight would be a good choice of default.

However, not all fonts are designed this way. The `CFF2` table is not backward compatible, except in the sense that it is possible to include both a `CFF` and a `CFF2` table in the same font, with the former serving as a fallback if the latter is not supported. And although one can choose to put the most general instance of a `glyf`-based font at the default location, one may not want to. For example, if the starting point in the design is two axes with four masters at four "extreme" locations, it might be preferable to choose one of those masters as the default, because doing so would substantially decrease the font file size compared with adding an extra pre-generated master there.

It is already possible to build a self-consistent variable font in this way. However, *not* putting the "most regular" instance at the default location poses an additional problem, as the OpenType *format* default location is also, at present, also the *interface* default location. That is, it is the location:

1. Rendered by default when no other location is specified.
2. Presented as the initial instance in a UI or font picker.

This hard link between the interface default and the format default is not inherent to variable font technology; it is just how things work now because there is currently no means of adjusting one without adjusting the other. Because providing that option would allow some fonts to be significantly smaller with no other drawback, we are proposing a format extension to do so.

2 The Proposal

An InstanceRecord in an OpenType `fvar` table includes a `uint16` called `flags`, which is currently unused. We propose that **Bit 0** of flags be used to indicate that the instance is the interface default.

A conforming font must use this bit for at most once instance. If it is present, that instance:

1. Will be presented initially in a UI or font picker,
2. Will be the location rendered if no other location is specified, and
3. Will provide the initial position of any axis not otherwise specified.

If the flag is not present the format default will be used for those purposes.

3 Alternatives

Obviously the currently available alternative is to interpolate a master corresponding to the desired interface default and make that the format default, at the expense of file size.

The proposed avar 2 specification possibly provides another alternative, in that it has enough flexibility to remap the default location. We see no problem with using the avar 2 facilities in this way but feel it is a complex and potentially confusing solution to a simple problem. Even if a font compiler were able to automate the translation we worry the result could be hard to understand and debug, and mixing that convention with simple, “avar 1 style” adjustments might be difficult.

It has also been suggested that modifications to the standard more in the spirit of avar 1, and therefore without the complexities of avar 2, could be sufficient.

4 References

- TypeDrawers discussion
- avar 2 option (and discussion of this proposal)