INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 3
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 3 m 64287
Geneva, Switzerland – July 2023

Title: Updating AVAR table in OFF to support the needs of modern computing platforms.

Author: Dave Crossland (Google Inc., dcrossland@google.com), Behdad Esfahbod (behdad@behdad.org), Laurence Penney (lorp@lorp.org), Liam Quin (Delightful Computing, liam@delightfulcomputing.com), Rod Sheeter (Google Inc., rsheeter@google.com)

Contents

Page

1. Introduction

The axis variations table 'avar' is an optional table used in variable fonts. Version 1 of 'avar' modifies aspects of how a design varies for different instances along a particular design-variation axis. It does this by piecewise linear remapping on a per-axis basis, with certain restrictions.

Version 2, proposed here, enables flexible axis remapping where each design-variation axis is modified according to the coordinates of multiple design-variation axes. In order to combine the effects of multiple input valuess, 'avar' version 2 uses the OFF variation mechanism itself to determine interpolated delta values and add them to design-variation axis coordinates. The final interpolated axis coordinates are used in all subsequent variation operations.

The efficiency of 'avar' version 2 enables fonts that:

- are significantly smaller;
- are easier for users to control;
- simplify source maintenance by avoiding redundant data.

Use cases include:

- warped variable font design spaces to reflect a typeface designer's intention accurately;
- parametric fonts with intuitive control methods and a much reduced data footprint;
- simpler methods of control for specialized variable fonts.

This proposal does not include glyph-specific transformations, for example for advanced text justification, but does not preclude data to support such such features being added to the table at a later revision.

This draft submission replaces Section 6 in m62937, *Proposed improvements and new functionality for the Open Font Format standard for global font support.*

2. The ‘avar’ version 2.0 header format

The 'avar' table is comprised of a small header plus segment maps for each axis.

Axis variation table

Type	Name	Description
uint16	majorVersion	Major version number of the axis variations table — set to 2.
uint16	minorVersion	Minor version number of the axis variations table — set to 0.
uint16	<reserved>	Permanently reserved; set to zero.
uint16	axisSegmentMapCount	The number of axisSegmentMaps for this font. If this is not 0, it shall be the same as axisCount in the 'fvar' table.
SegmentMaps	axisSegmentMaps[axisSegmentMapCount]	The segment maps array — one segment map for each axis, in the order of axes specified in the 'fvar' table.
Offset32To<DeltaSetIndexMap>	axisIndexMapOffset	Offset from beginning of the table to axisIndexMap.
Offset32To<ItemVariationStore>	itemVariationStoreOffset	Offset from beginning of the table to varStore.

The table format for ‘avar’ version 2 is the same as avar version 1, but offsets to two extra structures are appended to it: axisIndexMap and varStore.

axisIndexMap is a DeltaSetIndexMap structure that maps the axis indices implied in ‘fvar’ to indices used in varStore. The outer index identifies an ItemVariationData structure in varStore. The inner index identifies a deltaSet within an ItemVariationData.

varStore is an ItemVariationStore structure that points to a VariationRegions array and a list of ItemVariationData structures. Each ItemVariationData specifies a subset of VariationRegions and an array of deltaSets. Each deltaSet specifies a delta value for each region.

Delta values are typically in (but not limited to) the range [-1.0, 1.0].

Delta values are stored as if they were signed integers by multiplying their true value by 16384. Thus 1.0 is stored as 16384; -1.0 is stored as -16384.

The DeltaSetIndexMap and ItemVariationStore formats are given in OFF “Font variations common table formats”.

2.1. Processing

Processing of axis values in an ‘avar’ version 2 table happens in 3 stages for a given instance:

1. Initial normalization to convert user coordinates of each axis to initial normalized coordinates in the range [-1.0, 1.0].

2. Remap initial normalized coordinates of each axis via the axisSegmentMaps of ‘avar’ version 1, providing intermediate coordinates also in the range [-1.0, 1.0].

3. Calculate interpolated deltas for each axis via ‘avar’ version 2 and add them to intermediate coordinates, providing final coordinates that are clamped to the range [-1,1].

In more detail, step 3 proceeds as follows. Considering the coordinates provided by step 2, a scalar is determined for each variationRegion by the standard variation interpolation algorithm. Then, the deltaSets of each ItemVariationData are processed such that each delta value is multiplied by its associated scalar. Summing those products gives an interpolated delta to add to a particular axis coordinate, the axis index being defined in axisIndexMap. After the summing operations for all ItemVariationData structures is complete, axis values are clamped to to the range [-1.0, 1.0], giving final axis coordinates.

The final axis coordinates obtained by step 3 are subsequently used in the standard variation process described in Algorithm for Interpolation of Instance Values, applying to all ‘gvar’ data and all ItemVariationStore data elsewhere in the font.

The following algorithm implements step 3 above, producing the final normalized axis coordinates:

```
// let coords be the vector of current normalized coordinates.

std::vector<int> out;
for (unsigned i = 0; i < coords.size(); i++)
{
  int v = coords[i];
  uint32_t varidx = i;

  if (axisIdxMap != 0)
    varidx = (this+axisIdxMap).map(varidx);

  float delta = 0;

  if (varStore != 0)
    delta = (this+varStore).get_delta (varidx, coords);

  v += std::roundf (delta);
  v = std::clamp (v, -(1<<14), +(1<<14));

  out.push_back (v);
}
for (unsigned i = 0; i < coords.size(); i++)
  coords[i] = out[i];
```

2.2. Construction

The way ItemVariationStores are built is typically by using a variation modeler, that takes a series of master values at certain locations in the design-space, and produces VariationRegions and deltaSets to be stored in one or more ItemVariationData structures. This usage is identical, except that delta values apply to normalized axis coordinates rather than distances measured in font units. To build the avar version 2 mapping tables, the designer will need to produce a mapping of input axis locations and their respective output axis locations. This data then will constitute the set of masters to be fed to the variation modeler and populate the ItemVariationStore that will go into the avar version 2 table. The variation index for each axis will be stored in axisIndexMap.

2.3. Use cases

2.3.1. Designspace warping

In the OFF variations specification, registered axes offer a standard way to offer users control of a variable font on reasonably well defined scales. For example, users learn that the Regular version of a font has Weight=400, Width=100, while the Bold has wght=700, wdth=100. A typical 2-axis variable font with `wght` and `wdth` axes might have the following nine Named Instances:

	wdth=75	wdth=100	wdth=125
wght=300	Light Condensed	Light	Light Extended
wght=400	Condensed	Regular	Extended
wght=700	Bold Condensed	Bold	Bold Extended

However, in fonts with more than one design axis, this approach lacks the flexibility of older methods of interpolation, where the type designer used a font design application to specify freely the axis values for, say, the Bold Condensed. Notably, the wght coordinate of Bold Condensed did not need to match the wght coordinate of Bold, nor did the wdth coordinate of Bold Condensed need to match the wdth coordinate of Condensed. A Bold Condensed with wght,wdth of (677,81) rather than (700,75) was perfectly reasonable. Our grid of instances in such as case would look something like this:

	Condensed	Regular	Expanded
Light	300,75	300,100	300,115
Regular	400,75	400,100	400,120
Bold	677,81	695,100	700,125

While with the existing Open Font Format specification it is possible to set up a designspace like this, it has the significant drawback that many applications and systems expect all Bold weights to be at `wght`=700 and all Condensed weights to be at wdth=75, whatever the values of other axes. In practice, many OFF fonts encode numerous additional masters that ensure the design is as intended at all designspace locations. This not only increases the data footprint significantly, but also adds to the maintenance burden of the font.

Furthermore, there are many cases where a type designer does not intend to offer instances in certain regions of the design space. For example, a Black Condensed instance is often difficult to design, and many type families omit it. Desired behaviour, in case a user requests wght=900, wdth=75, may be for the font to provide the same instance for the Black Condensed as it does for the Bold Condensed.

By applying deltas to axis values anywhere in the designspace, the avar table version 2 “warps” the designspace and so resolves the issues described.

As with variation deltas in general, we also benefit from interpolation when axis values are influenced by a delta but are not exactly at the location of the delta. Each delta value is encoded as F2DOT14. Its interpolated value is determined by the standard OFF variations algorithm, then added to the value obtained from the standard normalization process. Thus, assuming the Bold Condensed instance at (700,75) has normalized coordinates (1,-1) and assuming the designspace location (677,81) has normalized coordinates ((677-400)/(700-400), -1 + (81-75)/(100-75)) = (0.9233,-0.76), then the ItemVariationStore needs a delta set with peaks at (-1,1), thus with region ((-1,-1,0),(0,1,1)) and the following delta values:

- wght -0.0767
- wdth +0.24

In practice there may be other delta sets required at the partial default locations (-1,0) and (0,1), which then influence the delta values required at (-1,1), in line with normal variations math.

The result of implementing designspace warping this way is that we preserve the original axis values where applications and systems expect them, namely all Bold instances at wght=700 and all Condensed weights at wdth=75, while at the same time minimizing the data footprint.

2.3.2. Duplication of axis values for non-linear interpolation

Certain variable fonts encode delta sets in such a way that, when multiple axes are synchronized, outline points move in curves rather than in straight lines. This technique has been referred to as HOI (higher order interpolation). The details of the non-linear encoding need not concern us here, but a practical drawback under the existing Open Font Format specification is that it requires users to set multiple design axes to identical values in order to obtain valid instances. When axes are not synchronized, the resulting glyphs are severely distorted and useless. Synchronization using JavaScript has been proposed as a solution. A clever exploit of the CFF2 format, allowing a single axis to perform all the necessary non-linear adjustment of outline points, has also been demonstrated (but this method cannot be adapted to adjust delta values in an ItemVariationStore non-linearly).

The avar version 2 table provides a solution to the synchonization problem, such that a user adjusts one "primary" axis and that value is cloned to other subordinate axes. Ideally the subordinate axes have the Hidden flag set in fvar to discourage manual operation. For a font with 3 axes requiring identical values, if the first axis is considered primary, then the other two can encode delta sets in avar's ItemVariationStore to clone the value of the first axis. Assuming the subordinate axes have the same minimum, default and maximum values as the primary axis, each subordinate axis requires either one or two delta sets to clone the negative and positive regions of the normalized value of the primary axis:

- If the axes use only the normalized region [0,1] then one delta set per subordinate axis is needed, referring to the region (0,1,1) of the primary axis and having the delta value of 1. This is the common case.

- If the axes use only the normalized region [-1,0] then one delta set per subordinate axis is needed, referring to the region (-1,-1,0) of the primary axis and having the delta value of -1.

- If the axes use both the normalized regions [-1,0] and [0,1] then two delta sets per subordinate axis are needed, one referring to the region (0,1,1) of the primary axis and having the delta value of 1, the other referring to the region (-1,-1,0) of the primary axis and having the delta value of -1.

2.3.3. Simplified controls for parametric fonts without redundant data

Parametric fonts, explored by Donald Knuth in Metafont and by others, offer fine-tuned control of specific font-wide aspects of their design. Such aspects typically include the thickness of vertical strokes, thickness of horizontal strokes, the overall width of the font, and direct control over vertical zones including cap-height, ascender height, x-height and descender height.

In an OFF variable font these parameters can be encoded as design-variation axes. The Type Network Variations Proposal[1] of 2017 provides the basis for parametric OFF fonts Amstelvar, Roboto Flex and others, demonstrating the potential for usable parametric fonts with large character sets. That flexibility comes from a designspace of many axes – sometimes 10 or more – which raises issues of control. Faced with 10 sliders, users are unlikely to be able to find the instance they want and are likely to get "lost" in a unusable region of the designspace.

In existing OFF fonts, the solution to getting "lost" is to include "blended" axes that combine the deltas of parametric axes in such a way that they function in exactly the same way as they would in a non-parametric font. The blended axes are exposed using the registered axes of wght, wdth, opsz, etc., or well specified custom axes, and they are encoded in the standard OFF manner using gvar and related data. The major problem with such fonts is their large data footprint, as the blended axes effectively replicate much of what the parametric axes do. For a font containing both parametric and blended axes, the problem is particularly acute, so the parametric axes may be omitted to save space — thereby removing core functionality. In such cases the parametric nature of the font is thus reduced to a convenience of sources.

The avar table version 2 provides a method to deploy a purely parametric font with user-friendly axes, while avoiding the data bloat of blended axes. The user axes, for example wght and wdth, on their own

[1] https://variationsguide.typenetwork.com/

do nothing (i.e. they have no gvar or similar data), but they control the parametric axes by means of avar. The blending of parametric values into user-facing values effectively happens live in the font engine. The parametric axes are expected to have the Hidden flag set in fvar to discourage (but not block) manual operation.

2.4. Inverse avar processing

Normally it is not possible for an application to obtain normalized axis values directly, these remaining private to the font engine. In fonts with an avar version 2 table, however, it can be useful to know the axis values that would invoke a given instance if the font engine did not support avar version 2. Use cases include:

- informing users of the effective settings of parametric axes;
- providing a polyfill for avar version 2.

The solution is for the application itself to implement the avar algorithms, both version 1 and version 2. This requires not only access to the binary avar table, but also knowledge of the font's axis extents as defined in the fvar table. Once an application has a complete set of final normalized values as defined above, the inverse of the avar version 1 algorithm can be applied to obtain user coordinates for all axes. (Processors shall be written in such a manner as to avoid a divide-by-zero error in implementing an inverse avar version 1 mapping, in the case where consecutive toCoordinate values are identical.)

Almost all fonts can provide a valid set of user axis coordinates for a given set of final normalized axis coordinates in this way. Exceptions are those fonts that encode deltas in a region that is not accessible without avar version 2, namely those fonts where the fvar definition of an axis normally precludes an active negative region or a active positive region by having its default equal to its minimum or its maximum respectively. Such cases are expected to be rare.

3. Recommendations and notes

3.1. Hidden axes

Since many avar version 2 fonts have axes not not intended for manual adjustment, it is recommended that such axes set the "hidden" flag in VariationAxisRecord of the fvar table.

3.2. Efficient axisIdxMap and ItemVariationData construction

In order to reduce proliferation of zero deltas, it is recommended to store in axisIndexMap only those axes that are remapped by avar version 2. For the same reason, if there are multiple different types of axis mapping affecting different sets of axes, consider using multiple ItemVariationData structures.

3.3. Other notes

- The set of axes involved in adjusting the coordinate for a given axis may include the axis itself.
- It is expected that implementations that handle only avar version 1 will ignore the entire table by rejecting the majorVersion value.