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 the 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

7. OFF font variations

This document summarizes changes to 7.3.1 ‘avar’—Axis variations table.

In 7.1.4 Coordinate scaled and normalization, replace

If an 'avar' table is present, then an additional normalization step is performed for each axis to compute the final normalized value

with:

If an 'avar' table is present, then additional normalization steps are performed for each axis to compute the final normalized value, described in this section and also in 7.3.1 *‘avar’—Axis variations table*.

7.3. Font variations tables

The following text replaces 7.3.1. avar—Axis variations table.

7.3.1. avar—Axis variations table

The axis variations table ('avar') is an optional table used in variable fonts. It can be used to vary a design for different instances according to the coordinates of one or more design-variation axes. In order to combine the effects of multiple input values, ‘avar’ uses the variation mechanism to determine interpolated delta values which are then added to design-variation axis coordinates. The final interpolated axis coordinates are used in all subsequent variation operations. Specifically, 'avar' allows modification of the coordinate normalization that is used when processing variation data for a particular variation instance.

The 'avar' table is used in combination with a font variations ('fvar') table and other required or optional tables used in variable fonts.

7.3.1.1. Overview

The ‘fvar’ table defines a font’s axes and the numeric range for each axis supported by the font. The numeric range for each axis is defined in terms of a “user” scale appropriate to each axis. When processing a font’s variation data to derive glyph outlines or other values for an instance, the instance coordinate for each axis must be mapped from the user scale to a normalized scale. A default normalization is defined that maps the minimum, default and maximum user values in the ‘fvar’ table to -1, 0, and 1, respectively. (See 7.1.4 for a detailed specification.)

```
if (userValue < axisDefaultValue)
{
  defaultNormalizedValue = -(axisDefault - userValue) / (axisDefault – axisMin);
}
else if (userValue > axisDefaultValue)
{
  defaultNormalizedValue = (userValue – axisDefault) / (axisMax – axisDefault);
}
```

```
else
{
  defaultNormalizedValue = 0;
}
```

If an 'avar' table is present, then the output of the default normalization can be further modified by allowing mappings to be defined for additional positions along each scale, creating multiple segments, with other values within each segment interpolated. The following figure illustrates an example of such a modification —the blue line shows the default mapping, with two segments, and the orange line shows the modified mapping, with additional segments.

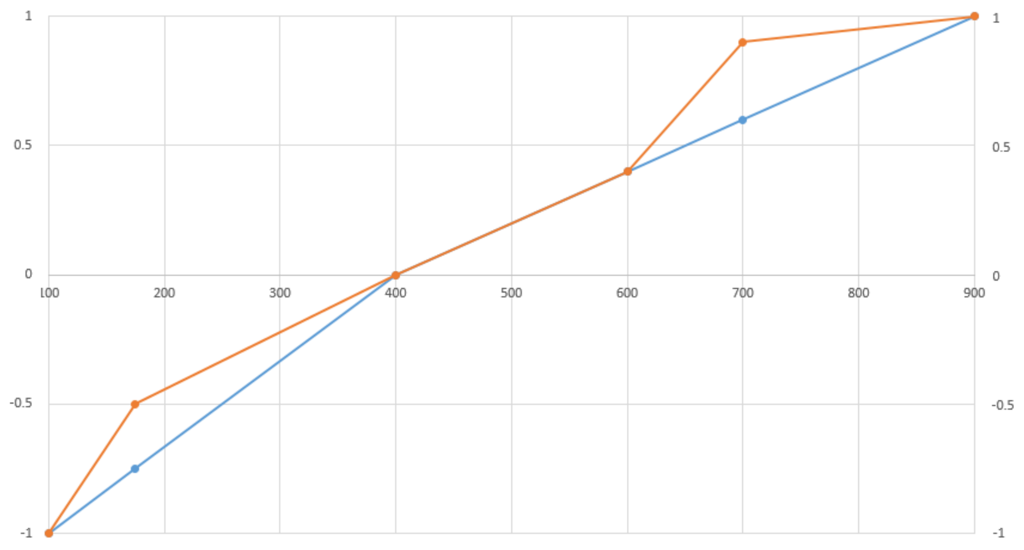

Figure 7.3: Example of 'avar'-modified normalization – horizontal axis is user scale, vertical axis is normalized scale

The conceptual effect of these additional scale mappings is to make the variation along an axis less linear. Values change linearly within each segment, but additional segments make the way that values change across the entire axis range less linear overall. The effect might also be described as compressing some portions of the scale while making other portions less compressed.

The visual effects of additional axis-value mappings in the 'avar' table are seen in how glyph outlines or other visual elements change as the user-scale values for an axis change. The same amount of change in the user-scale value may correspond to a subtle visual change in one portion of the axis range, but more dramatic changes in another portion of the axis range.

Note that it may be possible to achieve the same or similar effects by adding variation data for additional regions of the variation space. That approach requires more work and more font data, however, and tedious design iteration may be needed to obtain the desired results. The 'avar' table may provide a simple and light-weight way to achieve a particular effect.

Note also that the variation data created by the font designer will also have a significant effect on whether user-scale values and visual elements change uniformly. When an 'avar' table is used, the 'avar' table and the variation data are both factors in the variation behavior that the user will see.

There are two versions of the ‘avar’ table. Version 2 extends Version 1, so that every Version 1 ‘avar’ table remains valid. Version 2 enables a more flexible axis remapping where each design-variation axis is modified according to the coordinates of multiple design-variation axes. This can make fonts significantly simpler internally, and smaller, as well as giving font designers and tools control over unwanted axis value combinations.

The effects of 'avar' are font-wide, and not specific to individual glyphs.

7.3.1.2. Table formats

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

Axis variation table version 1.0

Type	Name	Description
uint16	majorVersion	Major version number of the axis variations table – set to 1.
uint16	minorVersion	Minor version number of the axis variations table – set to 0.
uint16	<reserved>	Permanently reserved, set to 0.
uint16	axisCount	The number of variation axes for this font. This must be the same number as axisCount in the 'fvar' table.
SegmentMaps	axisSegmentMaps[axisCount]	The segment maps array – one segment map for each axis, in the order of axes specified in the ‘fvar’ table.

Axis variation table version 2.0

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 0.
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[axisCount]	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.

There must be one segment map for each axis defined in the 'fvar' table, and the segment maps for the different axes shall be given in the order of axes specified in the 'fvar' table. The segment map for each axis is comprised of a list of axis-value mapping records.

When the ‘avar’ table has majorVersion = 2, it contains at the end offsets to two extra structures not present in earlier versions: 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”.

SegmentMaps record

Type	Name	Description
uint16	positionMapCount	The number of correspondence pairs for this axis.
AxisValueMap	axisValueMaps[positionMapCount]	The array of axis value map records for this axis.

Each axis value map record provides a single axis-value mapping correspondence.

AxisValueMap record

Type	Name	Description
F2DOT14	fromCoordinate	A normalized coordinate value obtained using default normalization.
F2DOT14	toCoordinate	The modified, normalized coordinate value.

Axis value maps can be provided for any axis, but are required only if the normalization mapping for an axis is being modified. If the segment map for a given axis has any value maps, then it must include at least three value maps: -1 to -1, 0 to 0, and 1 to 1. These value mappings are essential to the design of the variation mechanisms and are required even if no additional maps are specified for a given axis. If any of these is missing, then *no modification* to axis coordinate values will be made for that axis.

All of the axis value map records for a given axis shall have different fromCoordinate values, and axis value map records shall be arranged in increasing order of the fromCoordinate value. If the fromCoordinate value of a record is less than or equal to the fromCoordinate value of a previous record in the array, then the given record may be ignored.

Also, for any given record except the first, the toCoordinate value must be greater than or equal to the toCoordinate value of the preceding record. This requirement ensures that there are no retrograde behaviors as the user-scale value range is traversed. If a toCoordinate value of a record is less than that of the previous record, then the given record may be ignored.

7.3.1.3. Processing

The initial axis coordinate normalization process, with or without an 'avar' table, is described in 7.1.4. The process begins with a default normalization computation. Then if an 'avar' table is present, the normalized value is modified using data in this table.

Each pair of consecutive AxisValueMap records for a given axis defines a segment within the range for that axis, and a mapping from default normalized values for start and end points of the given segment to the final normalized values. Within a segment, intermediate values are interpolated linearly. For complete details of the normalization process, as well as an example using 'avar' data, see 7.1.4.

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’, providing intermediate coordinates also in the range [-1.0, 1.0].

3. If ‘avar’ version is 2, and either axisIndexMapOffset or itemVariationStoreOffset is non- zero, then calculate interpolated deltas for each axis 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:

For each axisId, determine an outer index and an inner index by looking up the axisId in axisIdxMap.

a) The outer index identifies an ItemVariationData within varStore.

b) The inner index identifies a delta set within that ItemVariationData.

c) Obtain the delta value based on processing the delta set identified above.

Next, apply the delta values obtained for each axisId to modify the intermediate coordinates obtained in step 2 above.

Rounding each item to the closest 2.14 value yields the coordinates presented to subsequent internal variation operations, including ‘gvar’ and all other ItemVariationStore structures.

Note that ItemVariationStore deltas are expressed as integers, but in ‘avar’ they ultimately represent 2.14 values.

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:

```
coordsAvar1 = […]
```

```
// the tuple of normalized coordinates, remapped by axisSegmentMaps,
// expressed as rounded int16 values

coordsAvar2 = [];

for (i = 0; i < coordsAvar1.length; i++) {
    v = coordsAvar1[i];

    if (varStore) {
        if (axisIdxMap) {
            [outer, inner] = axisIdxMap[i];
        } else {
            outer = i >> 16;
            inner = i & 0xFFFF;
        }

        if (!(outer == 0xFFFF && inner == 0xFFFF)) {
            delta = varStore.getDelta(coordsAvar1, outer, inner);
            // outer refers to an ItemVariationData structure
            // inner refers to a delta set

            v += delta; // in general, delta is not an integer
            v = clamp(v, -16384, 16384);
            // clamp to [-1,1] for 2.14
        }
    }
    coordsAvar2[i] = round(v);
}
```