OpenType Extension: ‘XVAR’ table

Microsoft, April 2016

This document describes a proposed new *Axes Variations (‘XVAR’)* table that provides variation data to allow non-linear variation across a font’s variation space with inter-axis effects.

# Background

The background for this is observations by designers that axes may not be completely independent. In particular, with default axis-scale normalization and ‘gvar’ processing, different interpolated instances for a fixed value on one axis, while values on other axes change, may be perceived as not being consistent in regard to that axis.

For example, consider a two-dimensional, weight + width variations font. The font designer may want to use the instance at (0.7, 0) (normalized scales) as the named instance for “Bold”, but find that (0.7, 1) feels too heavy for “Bold Extended”. As a result, they’d like to shift the position for “Bold Extended” to (0.68, 1). However, the side effect of that would be that the user-scale (‘fvar’) axis values or their mapping to externally-defined scales (e.g., CSS font-weight) no longer work as expected.

The desired outcome is that, at the user’s interface (‘fvar’ axes and externally-defined equivalents), the font’s axes should *feel* independent, with the normalization process and ‘gvar’ processing producing interpolated outlines that *feel right* to the font designer.

For example, imagine a (perhaps unlikely?) UI design in which one control allows a user to select a named instance, and then separate sliders allow them to tweak individual axis values, with the named-instance control snapping to a different named instance if the sliders are set to a matching value. Assuming the above example, if a user selects the “Bold” named instance and then moves the width slider up to 1, then it seems like the named-instance control should ideally snap to “Bold Extended” at that point, *but also* that the resulting design should feel (per the font designer’s judgement) to have the same degree of “boldness”.

In theory, it would be possible for a font designer to achieve desired results by adding intermediate-domain tuple variation tables (“TVs”) at different points in the variation space within the ‘gvar’ table. That approach really only makes sense if tweaks are needed for a small selection of glyphs, however. It’s expected that, in many cases, the designer will want to make adjustments that apply across all glyphs. Adding intermediates for all glyphs would not be efficient.

## Overview of the proposed solution

The ‘avar’ table allows for non-linear variation across any given axis. However, it does not allow for inter-axis effects. Support for inter-axis effects is assumed as a requirement.

One can think of the problem as needing to warp local regions with a font’s n-dimensional variation space, somewhat akin to gravitational warping of space-time in General Relativity. In terms of data representation, the desired effects might be represented by a relatively-minor extension of the ‘avar’ table in which from/to mappings of positions in the variation space are specified, rather than simple from/to mappings of values on a single axis. However, that simple solution for data representation leaves open the question of how the interpolation of intervening positions within the variation space would be calculated. Continuous variations would involve n-dimensional tensor calculus, or nth-order polynomial approximations that would require a large number of mappings to achieve desired results. Even segment-wise linear interpolation between the mapping points could require n-dimensional polyhedral tessellation. At first glance, the problem seems overly complex.

However, this proposal assumes that segment-wise-linear interpolation is adequate and that viable solution is already at hand: the tuple variation tables already being used in ‘gvar’, ‘cvar’ and other tables. These provide a way to represent adjustments, and the algorithm for processing is known and already used in implementations.

Each TV table can be thought of as a function that maps from target-value indices × n-dimensional variation space onto numeric deltas — *fi(u, v1, v2, …vn) → Z*. (In the case of the ‘gvar’ table, each TV provides *pairs* of deltas, and so can be thought of a pairs of functions, *fix* and f*i*y.) A set of TVs provides the overall adjustments of target values across the variation space, and the combination can be thought of as a single function that combines the individual functions: *F(u, v1, v2, …vn) → Z* where *F = f1 ∗ f2 … ∗ fm*. Also note that the format of TVs is agnostic with regard to what kind of target values are being adjusted. The format only assumes that the inputs of the adjustment functions are a target-value index plus a variation-space position.

Hence, this proposal uses TV tables to provide delta adjustments to any axis value at arbitrary input positions within a font’s variation space. The target-value index input for the function is an axis index, and the delta arrays always include N deltas where N is the number of axes, allowing adjustment to any or all of the axis values. The proposal requires that the end points of each axis must normalize to -1 and 1, and hence allows only for intermediate-domain TVs.

A key difference from TV tables in ‘gvar’ and other variation tables is that adjustments in the ‘XVAR’ table operate on the user-scale axis values as defined in the ‘fvar’ table. The default position on each axis still has special status in regard to having no scalar effect on deltas, but the particular value of zero is not special. The deltas are still peak adjustment values at specified positions, with scalar coefficients ranging from 0 to 1 being applied to the deltas at other positions.

# The Axes Variations (‘XVAR’) table

The ‘XVAR’ table uses structures similar to those in the ‘cvar’ and other variation tables to represent variation adjustments to variation-space axis values. This allows a font designer to achieve desired visual effects for different target positions within a font’s variation space where default ‘gvar’ processing would not provide desired results.

These targeted positions at which adjustments are made would typically correspond to interpolated, named instances, not positions for which the font designer created “master” designs.

For example, consider a font with weight, width and optical size variation axes. For a given value on the weight axis, it should be possible to change values for weight or optical size and have the visual results at those other positions have a desired apparent weight. In practice, however, a designer may find that a weight value that works for normal width and 12-point optical size produces results that are a little too light for extended width, or a little too heavy for 9-point optical size. They might want to tweak the weight to be a bit heavier in the one case, and a bit lighter in the other case. The ‘XVAR’ table allows for this by allowing a designer to apply adjustments to any or all axis values when a particular instance is selected.

These adjustments get applied in the interpolation processing and are invisible to the end user. Ideally, the end user will perceive smooth variation of glyph designs across the entire variation space with no unexpected interactions between axes.

Note that the structures within the ‘XVAR’ table are similar to those in other variation tables, but not identical. In particular, whereas in other variation tables the adjustments are associated with positions in the variation space that are represented using normalized coordinate values, in the ‘XVAR’ table the positions within the variation space are represented in terms of the pre-normalized scales defined in the ‘fvar’ table. Also, whereas in other tables the adjustment delta values are integer values, in the ‘XVAR’ table they are fractional values. There are minor changes in processing associated with these format differences. In general, however, the concepts are the same.

## Related and co-requisite tables

A font that includes an axes variations table must also include other tables required in a variations font: a [[font variations (‘fvar’) table]], a [[glyph variations (‘gvar’) table]], and a [[metrics variations (‘MVAR’) table]].

Fonts that support variations may optionally include an [[axis variation (‘avar’) table or a [[CVT variations (‘cvar’) table.

Note that a font can contain an ‘avar’ table as well as an ‘XVAR’ table. The ‘avar’ table is designed for similar purposes to the ‘XVAR’ table, but it has more limited capabilities. The effects of an ‘avar’ table can always be achieved using an ‘XVAR’ table instead, but not vice versa. The two tables are used at different stages in processing, and so it is possible to have both tables without ambiguity as to how they should be processed. It should never be necessary for a font to have both, however.

## Terminology

The OpenType specification uses certain tables and structures that have the same format as corresponding tables and structures in Apple’s TrueType specification, but with certain differences in terminology and labels. In particular, note the following differences:

* This document uses “n-tuple” to refer to an ordered set of variation-axis coordinates that specify a position within in a font’s variation space. Apple’s ‘gvar’ table specification refers to this as “coordinates”.
* This document uses “tuple variation table” to refer to structures that Apple’s ‘gvar’ and ‘cvar’ table specifications refer to as a “tuple”. Each tuple variation table has variation data associated with a particular region within the variation space, referenced using n-tuples.

For more details on terminology related to variation fonts, see [[OpenType Font Variations Overview]].

## Table formats

The axes variations table as the following format.

*Axes variations table:*

| Type | Name | Description |
| --- | --- | --- |
| Fixed | version | Version — set to 0x00010000 |
| SHORT | tupleVariationCount | A packed field. The high 4 bits are flags (see below), and the low 12 bits are the number of tuple variation tables — between 1 and 4095. |
| SHORT | axisCount | The number of variation axes for this font. This must be the same number as axisCount in the ‘fvar’ table. |
| Offset | offsetToSerializedData | Offset from the start of this table to the serialized tuple variation data. |
| struct | tupleVariationHeaders[] | Array of tupleVariationHeader records |

The axes variation table includes one or more tuple variation tables (“TVs”). Each *logical* TV is divided into two portions stored separately: a header, and serialized data. All of the TV headers are stored in one array at the end of the axes variations table header. The serialized TV data is store later in the axes variations table.

The tupleVariationCount field contains a packed value that includes flags and a count. The format for this member is as follows:

|  |  |  |
| --- | --- | --- |
| *Value* | *Name* | *Description* |
| 0xF000 | reservedFlags | Reserved for future use. |
| 0x0FFF | countMask | Mask for the low bits to give the number of tuple variation tables. This determines the number of tuple variation headers and corresponding sections within the serialized data. |

Each TV will provide adjustment values for axis coordinate values to be applied at some nominal position in the font’s variation space. Delta values are always provided for all axes; the delta values can be set to zero if no adjustment is required.

The tupleVariationCount field that corresponds in other variation tables, such as the glyphVariationData table within ‘gvar’, uses flags to indicate that certain packed-number data can be shared across multiple adjustment data tables. The equivalent within the ‘XVAR’ table would be to have packed numbers to indicate for which axes delta adjustment values are provided. Since the size of adjustment data within an ‘XVAR’ table will never be large, these mechanisms are not used. Hence, the flag bits of the tupleVariationCount field are ignored, but in other respects the format of values for this field are the same.

The header portion of a tuple variation table has the following format.

*tupleVariationHeader record:*

| Type | Name | Description |
| --- | --- | --- |
| SHORT | variationDataSize | The size in bytes of the packed adjustment data for this tuple variation table. |
| SHORT | flags | Flags — see details below. |
| Fixed | peakNTuple[axisCount] | Peak-change n-tuple for this tuple variation table. Required within the axes variation table. |
| Fixed | intermediateMinNTuple[axisCount] | Intermediate min n-tuple for this tuple variation. Required within the axes variation table. |
| Fixed | intermediateMaxNTuple[axisCount] | Intermediate max n-tuple for this tuple variation table. Required within the axes variation table. |

In the ‘XVAR’ table, delta values are represented as Fixed values, and there is one delta for every axis. Hence, the variationDataSize field must always be set to *sizeof(Fixed) \* axisCount*.

The flags value uses the following bit assignments:

|  |  |  |
| --- | --- | --- |
| *Value* | *Name* | *Description* |
| 0x8000 | embeddedPeakNTuple | Flag indicating that the header includes an embedded, peak n-tuple. Within an axes variations table, this must be set. |
| 0x4000 | intermediateRegion | Flag indicating that this tuple variation table applies to an intermediate region in the variation space. Within an axes variations table, this must be set. |
| 0x3FFF | reserved\_flag | Unused. |

Note that the format of the tuple variation header is nearly the same as that in other variation tables. The differences are that the n-tuple coordinate values use Fixed values rather than shortFrac, and there are more constraints imposed meaning that fewer portions of the flags field used. Because of the additional constraints, the size of the tuple variation headers is constant for a given font, varying only based on the number of variation axes.

**Note:** Within an ‘XVAR’ table, the peakNTuple, intermediateMinNTuple and intermediateMaxNTuple values within any TV header must conform to certain constraints. For each axis, the following must be true:

* The intermediateMinNTuple coordinate value must be greater than or equal to the minimum value for the axis given in the ‘fvar’ table.
* The intermediateMinNTuple coordinate value must be ***strictly less than*** the peakNTuple coordinate value.
* The peakNTuple coordinate value must be ***strictly less than*** the intermediateMaxNTuple coordinate value.
* The intermediateMaxNTuple coordinate must be less than or equal to the maximum value for the axis given within the ‘fvar’ table.

These constraints, along with the requirement that all TVs apply to intermediate regions, ensures that the minimum and maximum values for every axis remain unchanged and are mapped to -1 and +1 in the normalized scale, and ensures that the outer bounds of the overall variation space remain rectilinear.

The serialized variation data follows the headers. Whereas in other variation tables, such as ‘gvar’, the serialized data may include certain data that is shared across structures, these mechanisms are not used within the ‘XVAR’ table. As a result, the serialized data includes only delta values.

The ‘XVAR’ table also differs from other variation tables with regard to how delta values are represented. In other variation tables, the delta values are represented as variable-length runs of packed integer values. In contrast, delta values within the ‘XVAR’ table are represented using constant-length arrays of Fixed values represented directly, without any form of packing. For every TV, there is a constant number of deltas, one for each axis, and the size of the data is *sizeof(Fixed) \* axisCount*.

## Processing

When performing the glyph-interpolation process for a variations font, a first phase is to apply a transformation of variation-space coordinate values into normalized scales. Processing of the axis variations table occurs during this phase.

Normalization transforms user values using the scales in the ‘fvar’ table into a normalized scale with values from -1 to +1. The default normalization algorithm maps a given instance typeface into normalized values by mapping the minimum user-scale value of each given axis to -1, mapping the default value to 0, mapping the maximum value to +1, and using linear interpolation for intervening values between these.

When an ‘XVAR’ table is present, this default normalization process is preceded by first processing the ‘XVAR’ table to make adjustments to the selected user-scale coordinate values. For example, a user might select an instance with a weight value of 700 and a width value of 400; processing of the ‘XVAR’ table might adjust the weight value for this nominal position within the variation space to 689. The default normalization algorithm would then be applied to map the value 689 into a normalized-scale value between -1 and +1.

Processing of the ‘XVAR’ tuple variation tables (“TVs”) is very similar to processing of TVs within other variation tables such as ‘cvar’: A subset of the TVs is selected as being applicable, based on the user-selected variation instance. A scalar coefficient between 0 and 1 is then calculated for each TV, based on the selected instance and the min, peak and max coordinates for the given TV. Each scalar is then applied to the variation delta values specified in the given TV for each axis. The scaled delta values for each axis are then added up to provide a net adjustment to the user-scale, variation-space coordinates for that instance.

As in other variation tables, each TV is associated with a region within the variation space. Within the ‘XVAR’ table, every TV must cover an intermediate region.

As in other variation tables, for a selected variation-space instance, the process for selection of TVs and calculation of a scalar is done on an axis-by-axis basis, with the different scalars for each axis multiplied together to yield a net scalar for that TV. For a given axis, the selection process is to test each header and to ignore any if the instance value for that axis is not between the intermediate-min and intermediate-max values. Ignoring a TV in relation to a given axis implies that the scalar value for that table and axis is 1 (hence it has no effect on delta values). If the TV is selected, then a scalar is interpolated linearly, as follows:

If instance value *v* > peak value, then  
   
else

This yields a scalar between 0 and 1. After all of the axis scalars are calculated, they are multiplied together to yield a net scalar for the given TV. The net scalar is then applied to each of the delta values given in that TV to produce the net adjustment to the input instance coordinate values.