INTERNATIONAL ORGANISATION FOR STANDARDIZATION

ORGANISATION INTERNATIONALE DE NORMALIZATION

ISO/IEC JTC 1/SC 29/WG 3

CODING OF MOTING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 3 mNNNN

Online — April 2024

Title: VARC Hint Guidance for CFF2

Author: Skef Iterum (Adobe Inc., siterum@adobe.com)

The table in 5.3.1 is modified to add a CFSH row:

Tag	Name
CFF	Compact Font Format 1.0
CFF2	Compact Font Format 2.0
CFSH	CFF2 Supplementary Hint Table (for VARC)
VORG	Vertical Origin (optional table)

The first paragraph in 5.3.3.11.2 CharStringINDEX is modified to read:

The CharStringINDEX is an INDEX (5.3.3.5.2) that contains the CharStrings for all CFF2 glyphs in the font. The location of the CharStringINDEX is specified by the CharStringINDEXOffset key in the TopDICT table (5.3.3.7). Each CharString is accessed by glyph ID. The first CharString (glyph ID 0) shall be the `.notdef` glyph. The number of glyphs defined in the CFF2 table shall be determined from the CharStringINDEX count field. When there is no VARC table in the font the value of this field shall be equal to the value of the numGlyphs field in the 'maxp' or 'MAXP' table (5.1.6).

NOTE When a VARC table is present in the font the value of the CharStringINDEX count field may be less than the maxp/MAXP count.

In section 5.3.3.8.5, the two sentences (one for format 3, one for format 4) describing the value of the sentinel field are changed to the following:

The sentinel glyph ID provides a final range in the array. When there is no VARC table it shall be set equal to *numGlyphs*, the number of glyphs in the font, taken from MAXP if present or from maxp otherwise. When a VARC table is present the value of the sentinel glyph ID shall be less than or equal to *numGlyphs* and greater than or equal to the count field in the CharStringINDEX.

The following CFSH section is added as 5.3.4, pushing the VORG section to 5.3.5. All of the text in this section is new so highlighting is omitted.

5.3.5 CFSH — Compact Font Format Supplementary Hint Table

CFSH is an optional table used to provide supplementary CFF2-format PrivateDICT structures for VARC composite glyphs, and to map each VARC glyph to those PrivateDICTs or to PrivateDICTs in the CFF2 table. Its contents are CFF2-style subtables, in some cases slightly modified.

CFSH Header

Type	Name	Description
uint16	major_version	Table major version number (set to 1)
uint16	minor_version	Table minor version number (set to 0)
Offset32	privateDICTIndexOffset	Offset (from start of CFSH table) to a CFF2 INDEX of Private DICTs. 0 if no PrivateDICTIndex.
uint16	initialPrivateDICT	The FontDICT index associated with of the first entry in the PrivateDICTIndex (default is 0).
Offset32	fdSelectOffset	Offset (from start of CFSH table) to the FontDICTSelect subtable. Must not be 0.
Offset32	itemVarStoreOffset	Offset (from start of CFSH table) to the Item Variation Store table (may be 0)

5.3.5.1 Private DICT Index and initialPrivateDICT

In the CFF2 table a Font DICT INDEX contains FontDICT (5.3.3.8.4) structures, each of which encodes the size and offset of a PrivateDICT (5.3.3.9). CFSH also encodes CFF2 Private DICTs but eliminates the indirection through the FontDICT. Instead, in contains a Private DICT Index, which is a CFF2 Index structure (5.3.3.5.2) that stores a list of Private DICTs directly.

The initialPrivateDICT field is the "FontDICT index" associated with the first entry in the Private DICT Index. Thus if initialPrivateDICT is 24, the index of the first CFSH private dict is 24, the index of the second is 25, and so on.

The initialPrivateDICT field shall be set to the size of the CFF2 FontDICT INDEX (5.3.3.8.2), so that the index of the first CFSH Private DICT is one greater than the index of the last CFF2 Private DICT. The indexes of the two sets of Private DICTs shall not overlap.

If the itemVarStoreOffset field is non-zero, then the `vsindex` and `blend` operators shall relate to the Item Variation Store it points to. If the field is zero then those operators shall relate to the Item Variation Store in the CFF2 table.

A PrivateDICT in the CFSH table shall not include the `Subrs` operator.

5.3.5.2 The FontDICTSelect Offset

The FontDICTSelect offset points to a CFF2 FontDICTSelect subtable (5.3.3.8.3). Clients shall ignore this field when it is set to 0, in order to support future minor extensions of the table. However, in a version 1.0 CFSH table the offset shall not be 0. As of version 1.0 only FontDICTSelect Format 4 is supported, but with one modification: the fontDICTID field in the first Range4 record can have a value other than 0.

The FontDICTSelect subtable in CFSH can overlap with the FontDICTSelect subtable in the CFF2 table but there shall not be gap between the last glyph mapped in CFF2 and the first glyph mapped in CFSH. The sentinel field in CFSH shall be the highest GID defined in the font (from maxp or MAXP).

5.3.5.3 The itemVarStore Offset

When not zero, this field points to an Item Variation Store used for the `vsindex` and `blend` operators for any PrivateDICTs in the CFSH table.

The following text is placed as section 7.3.10.4 in the VARC section, pushing "Processing of Variable Composite Glyphs" to 7.3.10.5. All of the text in this section is new so highlighting is omitted.

7.3.10.4 Hinting of Variable Composite Glyphs

When a VARC composite glyph is built from glyf components that include TT instructions (5.2.4.1.1), or from CFF2 components that include hinting parameters (5.3.3.9.2.1), that data shall be considered part of the composite. Whether and how hint data is used when rasterizing the glyph can depend on a number of factors, including the transforms applied to the component both within a variable composite or "externally" (e.g. using a CSS transform).

The glyf table includes its own composite format (5.2.4.1.2) that makes use of the TT instructions of component glyphs. That mechanism can serve as a model for when and how to apply instructions when rasterizing a glyf-based variable composite glyph.

Neither CFF nor CFF2, in contrast, has internal support for compositing, and while the COLR table implicitly adds such support, COLR leaves the question of hinted rasterization open. The rest of this section explains how to adapt CFF2 hinting parameters when rasterizing a VARC composite glyph. Those parameters can then be applied as they would be when rasterizing a hinted CharString in a CFF2 table.

7.3.10.4.1 Hinting CFF2 Components in a Variable Composite Glyph

The hinting parameters in the CFF2 table evolved from related information included in PostScript Type 1 fonts, and consist of a combination of PrivateDict parameters—each of

which is associated with a particular subset of glyphs—and per-glyph parameters. For a variable composite glyph the PrivateDict parameters are taken from the PrivateDICT associated with the GID of the composite—or if there are multiple layers of compositing, the PrivateDICT associated with the "outer-most" composite. The per-glyph parameters are adapted from the hint operators included in the CFF2 CharString of a component glyph, and therefore in VARC are technically per-component rather than per-glyph. These must be adapted to account for the transformations and translations applied in each layer of compositing.

The Private DICT

When a CFF2 glyph is included in a VARC composite as a component, both the PrivateDICT of the component and that of the composite shall be consulted in order to render it.

The PrivateDICT of the component glyph will always be in the CFF2 table (5.3.3.9) and is mapped via the component's GID in the CFF2 FontDICTSelect subtable (5.3.3.8.3) (unless all CFF2 glyphs use the same PrivateDICT, in which case there will only be one). That PrivateDICT may contain `Subrs` or `vsindex` operators needed to desubroutinize the component's CharString, and to resolve any blends it contains relative to the instance of the component to be rendered.

The PrivateDICT of the (top-level) variable composite glyph will either be in the CFF2 table (5.3.3.8.2) or the CFSH table (5.3.5.1). It is found using this procedure:

1. If there is a CFSH table and it contains a FontDICTSelect structure, that is checked for the GID of the variable composite glyph. If the GID is mapped, the PrivateDICT index it maps to shall be the index for the composite.
2. If there is no CFSH table, no FontDICTSelect structure in the table, or the GID of the composite is not mapped in that structure, the FontDICTSelect structure in the CFF2 table is checked for the GID, and the PrivateDICT index it maps to shall be the index for the composite.
3. If neither FontDICTSelect structure maps the GID the font is malformed. If there is a CFSH table, the value of the initialPrivateDICT field shall be the fallback index for the composite. Otherwise the fallback index shall be 0.
4. If there is a CFSH table and the index for the composite is greater than or equal to its initialPrivateDICT field, the value of that field is subtracted from the index and the result shall be used as the offset into the CFSH PrivateDICT Index.
5. Otherwise the index for the composite shall be used as the offset into the CFF2 FontDICT Index, and its PrivateDICT shall be used for the composite.

Per-glyph parameters

A hinted glyph has some combination of these parameters (5.3.3.9.2.1):

1. Horizontal and vertical stem regions (hstem(hm), vstem(hm))
2. Hintmasks
3. Counter hinting (cntrmask)

Relative to a given "top-level" composite glyph to be rendered, an "atomic" (or "bottom-level") component glyph may be subject to multiple sets of transformations and translations, with one set per level of variable compositing. This section describes how to apply those transformations and translations to stem regions, and how to decide whether to proceed with hinting in a given dimension of the component glyph (horizontal or vertical) or to *cancel* hinting in that dimension. Intuitively, hinting in a given dimension should be cancelled when there is rotation that is not a multiple of 180 degrees, or when there is any skew in the opposite dimension.

NOTE 1: In addition to variable composite transforms, a glyph may also be subject to "external" transforms, such as those specified with a Cascading Style Sheets `transform` property. An implementation can either avoid hinting in such circumstances or can compose those transforms into the cumulative transformation of the stem hints, as described below.

NOTE 2: As of the current version of this specification a `cntrmask` only applys at the level of atomic components; there is no means of applying a counter-mask to influence the spacing between elements of different components of a composite.

Whether hinting in a given dimension should be cancelled, and how the stems in that dimension should be adjusted when it is not cancelled, can be determined by performing the cumulative transformation on the three points p_1 (100, 0), p_2 (0, 0) and p_3 (0, 100) and considering the result. If we call the transformed points p_1', p_2', and p_3' respectively, adjustment of the horizontal stems proceeds as follows:

If the line from p_1' to p_2' is not close enough to horizontal for hinting purposes (which may vary by implementation), hinting of horizontal stems is cancelled. Otherwise hinting proceeds in that dimension with a scale factor of

$s = (y_3' - y_2') / 100$

and a translation factor of

$t = y_2'$

If hinting proceeds, the horizontal stem deltas are unpacked into positional bottom, top pairs and each pair is processed as follows:

1. If the pair does not represent an edge stem, each value is multiplied by *s* and translated by *t*. If *s* is negative the top and bottom values are swapped.
2. If the pair represents a bottom edge stem, the lower edge is multiplied by *s* and translated by *t*. If *s* is positive the pair is re-encoded as a bottom edge stem with the adjusted value as the bottom. If *s* is negative the pair is re-encoded as a top edge stem with the adjusted value as the top.

   For example, if the original bottom position was *b*, the unpacked pair was encoded as having a width of -21 (i.e. the first number unpacked to *b* + 21 (the "upper edge") and the second to *b* (the "lower edge")). If *s* is positive the adjusted lower edge will

be *b**s* + *t* and the adjusted upper edge will be *b**s* + *t* + 21. (If necessary this can be re-encoded as a bottom edge hint with the first value *b**s* + *t* + 21 and a width of -21.)

If *s* is negative then the hint is adjusted to be a top hint, with *b**s* + *t* as the upper edge and *b**s* + *t* - 20 as the lower edge, which can be re-encoded as an initial value of *b**s* + *t* and a width of -20.

3. If the pair represents a top edge stem, the upper edge is multiplied by *s* and translated by *t*. If *s* is positive the pair is re-encoded as a top edge stem with the adjusted value as the top. If *s* is negative the pair is re-encoded as a bottom edge stem with the adjusted value as the bottom.

If *s* is positive the adjusted stem pairs simply replace the respective original pairs. If *s* is negative the orders of both the stem pairs and the corresponding bits in each of the `hintmasks` and `cntrmasks` for the glyph are reversed, and those stem pairs and masks are used to render the component.

Adjustment or cancellation of vertical stems is analogous to the horizontal case.