INTERNATIONAL STANDARD

ISO/IEC 14496-22

Fourth edition
2019-01

AMENDMENT 2
2023-01

Information technology — Coding of audio-visual objects —

Part 22: **Open Font Format**

AMENDMENT 2: Extending colour font functionality and other updates

Technologies de l'information — Codage des objets audiovisuels —

Partie 22: Format de police de caractères ouvert

AMENDEMENT 2: Extension de la fonctionnalité des polices de couleur et autres mises à jour

Reference number
ISO/IEC 14496-22:2019/Amd. 2:2023(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 14496 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Information technology — Coding of audio-visual objects —

Part 22: **Open Font Format**

AMENDMENT 2: Extending colour font functionality and other updates

4.3

Add the following row to the table defining data types before the row that specifies Offset32:

Offset24	24-bit offset to a table, same as uint24, NULL offset = 0x000000

5.7.1

Replace the entire content of the subclause with the following:

The DSIG table contains the digital signature of the OFF font. Signature formats are widely documented and rely on a key pair architecture. Software developers, or publishers posting material on the Internet, create signatures using a private key. Operating systems or applications authenticate the signature using a public key.

The W3C and major software and operating system developers have specified security standards that describe signature formats, specify secure collections of web objects, and recommend authentication architecture. OFF fonts with signatures will support these standards.

OFF fonts offer many security features:

— Operating systems and browsing applications can identify the source and integrity of font files before using them,

— Font developers can specify embedding restrictions in OFF fonts, and these restrictions cannot be altered in a font signed by the developer.

The enforcement of signatures is an administrative policy that may be supported by the host environment in which fonts are used. Systems may restrict use of unsigned fonts, or may allow policy to be controlled by a system administrator.

Anyone can obtain identity certificates and encryption keys from a certifying agency, such as Verisign or GTE's Cybertrust, free or at a very low cost.

The DSIG table is organized as follows. The first portion of the table is the header.

DSIG Header

Type	**Name**	**Description**
uint32	version	Version number of the DSIG table (0x00000001)

Type	Name	Description
uint16	numSignatures	Number of signatures in the table
uint16	flags	Shall be set to 0x0001
SignatureRecord	signatureRecords[numSignatures]	Array of signature records

The version of the DSIG table is expressed as a uint32, beginning at 0. The version of the DSIG table currently used is version 1 (0x00000001).

Permission bit 0 allows a party signing the font to prevent any other parties from also signing the font (counter-signatures). If this bit is set to zero (0) the font may have a signature applied over the existing digital signature(s). A party who wants to ensure that their signature is the last signature can set this bit.

The DSIG header has an array of signature records that specify the format and offset of signature blocks.

SignatureRecord

Type	Name	Description
uint32	format	Format of the signature
uint32	length	Length of signature in bytes
Offset32	signatureBlockOffset	Offset to the signature block from the beginning of the table

Signatures are contained in one or more signature blocks. Signature blocks may have various formats; currently one format is defined. The format identifier specifies both the format of the signature block, as well as the hashing algorithm used to create and authenticate the signature.

Signature Block Format 1

Type	Name	Description
uint16	reserved1	Reserved for future use; set to zero.
uint16	reserved2	Reserved for future use; set to zero.
uint32	signatureLength	Length (in bytes) of the PKCS#7 packet in the signature field.
uint8	signature[signatureLength]	PKCS#7 packet

For more information about PKCS#7 signatures see [10].

For more information about counter-signatures, see [11].

Format 1: For whole fonts, with either TrueType outlines and/or CFF data

PKCS#7 or PKCS#9. The signed content digest is created as follows:

1) If there is an existing DSIG table in the font:

   a) Remove the DSIG table from font.

   b) Remove the DSIG table entry from the Table Directory.

   c) Adjust table offsets as necessary.

   d) Recalculate the checksumAdjustment in the ‘head’ table.

2) Hash the revised font data using a secure one-way hash (such as MD5) to create the content digest.

3) Create the PKCS#7 signature block using the content digest.

4) Create a new DSIG table containing the signature block.

5) Add the DSIG table to the font, adjusting table offsets as necessary.

6) Add a DSIG table entry to the Table Directory.

7) Recalculate the checksumAdjustment in the 'head' table.

Validation of a signature in a font is done by repeating steps 1 – 4 in an in-memory copy of the font file. Note that changing the checksumAdjustment in the last step does not break the signature because verification is done on an in-memory copy with these changes.

Prior to signing a font file, ensure that all the following attributes are true:

— The magic number in the 'head' table is correct.

— Given the numTables value in the Table Directory, the other values in the Table Directory are consistent.

— The table records in the Table Directory are ordered alphabetically by the table tags, and there are no duplicate tags.

— The offset of each table is a multiple of 4. (That is, tables are long word aligned.)

— The first actual table in the file comes immediately after the directory of tables.

— If the tables are sorted by offset, then for all tables i (where index 0 means the table with the smallest offset), Offset[i] + Length[i] <= Offset[i+1] and Offset[i] + Length[i] >= Offset[i+1] - 3. In other words, the tables do not overlap, and there are at most 3 bytes of padding between tables.

— The pad bytes between tables are all zeros.

— The offset of the last table in the file plus its length is not greater than the size of the file.

— The checksums of all tables are correct.

— The 'head' table's checksumAdjustment field is correct.

Signatures for Font Collections

The DSIG table for a Font Collection (TTC) shall be the last table in the TTC file. The offset to the table is put in the TTCHeader (version 2). Signatures of TTC files are expected to be Format 1 signatures.

The signature of a TTC file applies to the entire file, not to the individual fonts contained within the TTC. Signing the TTC file ensures that other contents are not added to the TTC.

Individual fonts included in a font collection should not be individually signed as the process of making the TTC could invalidate the signature on the font.

When DSIG table is created for a collection file, the steps given above are used, with these revisions:

— In step 1: if there is an existing DSIG table referenced in a version 2.0 TTC header, the DSIG table is removed, and the DSIG fields in the header is set to NULL. No recalculation of a checksumAdjustment is required.

— In steps 6 and 7: the DSIG table is added to the file, not to any individual font within the collection. A version 2.0 TTC header is required, with the DSIG fields in the header set to reference the DSIG table.

— Step 8 is not applicable.

See the TTC Header description (subclause 4.6.3) for related information.

5.7.11

Replace the content of subclause 5.7.11 with the following:

The COLR table adds support for multi-colored glyphs in a manner that integrates with the rasterizers of existing text engines and that is designed to be easy to support with current OpenType font files.

The COLR table defines color presentations for glyphs. The color presentation of a glyph is specified as a graphic composition using other glyphs, such as a layered arrangement of glyphs, each with a different color. The term “color glyph” is used informally to refer to such a graphic composition defined in the COLR table; and the term “base glyph” is used to refer to a glyph for which a color glyph is provided. Processing of the COLR table is done on glyph sequences after text layout processing is completed and prior to final presentation of glyphs. Typically, a base glyph is a glyph that may occur in a sequence that results from the text layout process.

For example, the Unicode character U+1F600 is the grinning face emoji. Suppose in an emoji font the ‘cmap’ table maps U+1F600 to glyph ID 718. Assuming no glyph substitutions, glyph ID 718 would be considered the base glyph. Suppose the COLR table has data describing a color presentation for this using a layered arrangement of other glyphs with different colors assigned: that description and its presentation result would be considered the corresponding color glyph.

Two versions of the COLR table are defined.

Version 0 allows for a simple composition of colored elements: a linear sequence of glyphs that are stacked vertically as layers in bottom-up z-order. Each layer combines a glyph outline from the ‘glyf’, CFF or CFF2 table (referenced by glyph ID) with a solid color fill. These capabilities are sufficient to define color glyphs such as those illustrated in Figure 5.6.

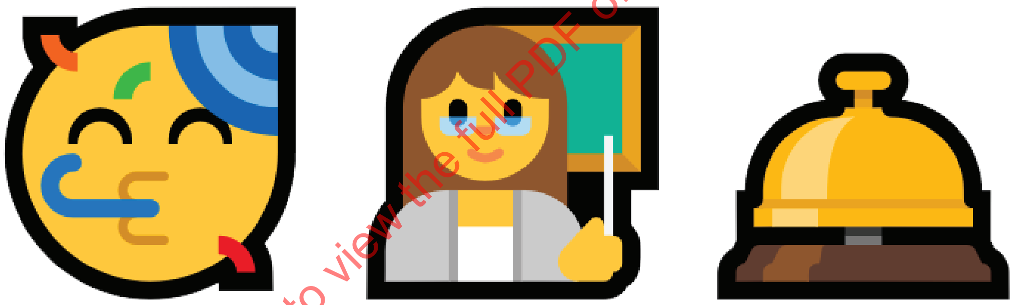


Figure 5.6 — Examples of the graphic capabilities of COLR version 0

Version 1 supports additional graphic capabilities. In addition to solid colors, gradient fills can be used, as well as more complex fills using other graphic operations, including affine transformations and various blending modes. Version 1 capabilities allow for color glyphs such as those illustrated in Figure 5.7:

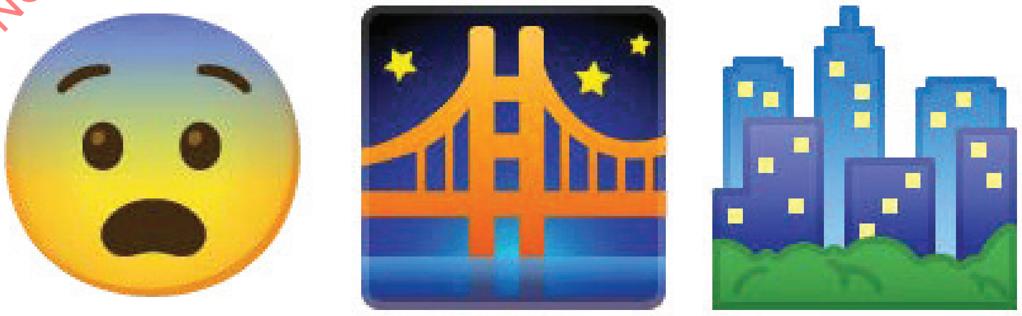


Figure 5.7 — Examples of the graphic capabilities of COLR version 1

Version 1 also extends capabilities in variable fonts. A COLR version 0 table can be used in variable fonts with glyph outlines being variable, but no other aspect of the color composition being variable. In version 1, all of the new constructs for which it could be relevant have been designed to be variable; for example, the placement of color stops in a gradient, or the alpha values applied to colors. The graphic capabilities supported in version 0 and in version 1 are described in more detail below.

The COLR table is used in combination with the CPAL table (5.7.12): all color values are specified as entries in color palettes defined in the CPAL table. If the COLR table is present in a font but no CPAL table exists, then the COLR table is ignored.

5.7.11.1 Graphic compositions

The graphic compositions in a color glyph definition use a set of 2D graphic concepts and constructs:

— Shapes (or *geometries*)

— Fills (or *shadings*)

— Layering—a *z-order*—of elements

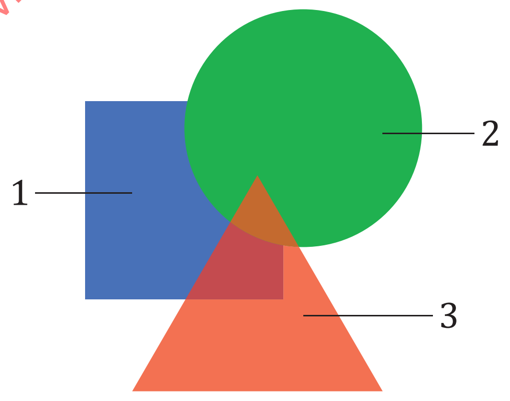
— Composition and blending modes—different ways that the content of a layer is combined with the content of layers above or below it

— Affine transformations

For both version 0 and version 1, shapes are obtained from glyph outlines in the 'glyf', 'CFF' or CFF2 table, referenced by glyph ID. Colors used in fills are obtained from the CPAL table.

The simplest color glyphs use just a few of the concepts above: shapes, solid color fills, and layering. This is the set of capabilities provided by version 0 of the COLR table. In version 0, a base glyph record specifies the color glyph for a given base glyph as a sequence of layers. Each layer is specified in a layer record and has a shape (a glyph ID) and a solid color fill (a CPAL palette entry). The filled shapes in the layer stack are composed using only alpha blending.

Figure 5.8 illustrates the version 0 capabilities: three shapes are in a layered stack: a blue square in the bottom layer, an opaque green circle in the next layer, and a red triangle with some transparency in the top layer.



Key

1	layer 0 (bottom)
2	layer 1
3	layer 2 (top)

Figure 5.8 — Basic graphic capabilities of COLR version 0

The basic concepts also apply to color glyphs defined using the version 1 formats: shapes have fills and can be arranged in layers. But the additional formats of version 1 support much richer capabilities. In a version 1 color glyph, graphic constructs and capabilities are represented primarily in *Paint* tables, which are linked together in a *directed, acyclic graph*. Several different Paint formats are defined, each describing a particular type of graphic operation:

— A PaintColrLayers table provides a layering structure used for creating a color glyph from layered elements. A PaintColrLayers table can be used at the root of the graph, providing a base layering structure for the entire color glyph definition. A PaintColrLayers table can also be nested within the graph, providing a set of layers to define some graphic sub-component within the color glyph.

— The PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, and PaintVarSweepGradient tables provide basic fills, using color entries from the CPAL table.

— The PaintGlyph table provides glyph outlines as the basic shapes.

— The PaintTransform and PaintVarTransform tables are used to apply an affine transformation matrix to a sub-graph of paint tables, and the graphic operations they represent. Several Paint formats are also provided for specific transformation types: translate, scale, rotate, or skew, with additional variants of these formats for variations and other options.

— The PaintComposite table supports alternate compositing and blending modes for two sub-graphs.

— The PaintColrGlyph table allows a color glyph definition, referenced by a base glyph ID, to be re-used as a sub-graph within multiple color glyphs.

NOTE Some paint formats come in *Paint** and *PaintVar** pairs. In these cases, the latter format supports variations in variable fonts, while the former provides a more compact representation for the same graphic capability but without variation capability.

In a simple color glyph description, a PaintGlyph table might be linked to a PaintSolid table, for example, representing a glyph outline filled using a basic solid color fill. But the PaintGlyph table could instead be linked to a much more complex sub-graph of Paint tables, representing a shape that gets filled using the more-complex set of operations described by the sub-graph of Paint tables.

The graphic capabilities are described in more detail in 5.7.11.1.1 – 5.7.11.1.9. The formats used for each are specified in 5.7.11.2.

5.7.11.1.1 Colors and solid color fills

All colors are specified as a base zero index into CPAL (5.7.12) palette entries. A font can define alternate palettes in its CPAL table; it is up to the application to determine which palette is used. A palette entry index value of 0xFFFF is a special case indicating that the text foreground color (defined by the application) should be used, and shall not be treated as an actual index into the CPAL ColorRecord array.

The CPAL color data includes alpha information, as well as RGB values. In the COLR version 0 formats, a color reference is made in a LayerRecord as a palette entry index alone. In the formats added for COLR version 1, color references include a palette entry index and a separate alpha value within the COLR structure for a solid color fill or gradient color stop (described below). Separation of alpha from palette entries in version 1 allows use of transparency in a color glyph definition independent of the choice of palette. The alpha value in the COLR structure is multiplied into the alpha value given in the CPAL color entry.

Two color index record formats are defined: ColorIndex, and VarColorIndex. The latter can be used in variable fonts to make the alpha value variable.

In version 1, a solid color fill is specified using a PaintVarSolid or PaintSolid table, with or without variation support, respectively. See 5.7.11.2.6.2 for format details.

See 5.7.11.1.3 for details on how fills are applied to a shape.

5.7.11.1.2 Gradients

5.7.11.1.2.1 General

COLR version 1 supports three types of gradients: linear gradients, radial gradients, and sweep gradients. For each type, non-variable and variable formats are defined. Each type of gradient is specified using a color line.

5.7.11.1.2.2 Color Lines

A color line is a function that maps real numbers to color values to define a one-dimensional gradation of colors, to be used in the definition of linear, radial, or sweep gradients. A color line is defined as a set of one or more color stops, each of which maps a particular real number to a specific color.

On its own, a color line has no positioning, orientation or size within a design grid. The definition of a linear, radial, or sweep gradient will reference a color line and map it onto the design grid by specifying positions in the design grid that correspond to the real values 0 and 1 in the color line. The specification for linear, radial and sweep gradients also include rules for where to draw interpolated colors of the color line, following from the placement of 0 and 1.

A color stop is defined by a real number, the *stop offset*, and a color. A color line shall have at least one color stop. (Stop offsets are represented using F2DOT14 values, therefore color stops can only be specified within the range [-2, 2). See 5.7.11.2.5 for format details.) If only one color stop is specified, that color is used for the entire color line; at least two color stops are needed to create color gradation.

Color gradation is defined over the interval from the color stop with the minimum offset, through the successive color stops, to the color stop with the maximum offset. Between numerically-adjacent color stops, color values are linearly interpolated. See *Interpolation of Colors* in 5.7.12 for requirements on how colors are interpolated.

Color values outside the defined interval are determined by the color line's *extend mode*, described below. In this way, colors are defined for all stop offset values, from negative infinity to positive infinity.

For example, a gradient color line could be defined with two color stops at 0.2 and 1.5. Colors for offsets between 0.2 and 1.5 are interpolated. Colors for offsets above 1.5 and below 0.2 are determined by the color line's *extend mode*.

If there are multiple color stops defined for the same stop offset, the first one is used for computing color values on the color line below that stop offset, and the last one is used for computing color values at or above that stop offset. All other color stops for that stop offset are ignored.

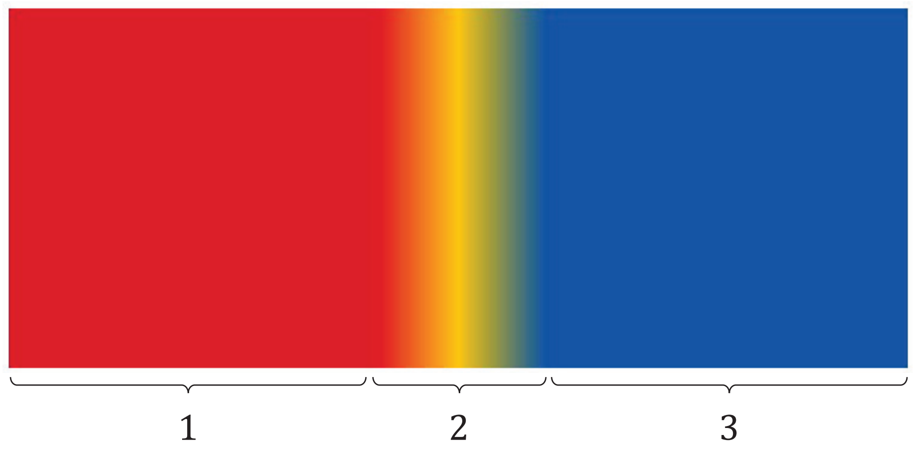
The color patterns outside the defined interval are determined by the color line's extend mode. Three extend modes are supported:

— **Pad:** outside the defined interval, the color of the closest color stop is used. Using a sequence of letters as an analogy, given a sequence "ABC", it is extended to "...*AA* ABC *CC*...".

— **Repeat:** The color line is repeated over repeated multiples of the defined interval. For example, if color stops are specified for a defined interval of [0.2, 1.5], then the pattern is repeated above the defined interval for intervals (1.5, 2.8], (2.8, 4.1], etc.; and also repeated below the defined interval for intervals [-1.1, 0.2), [-2.4, -1.1), etc. In each repeated interval, the first color is that of the farthest defined color stop. By analogy, given a sequence "ABC", it is extended to "...*ABC* ABC *ABC*...".

— **Reflect:** The color line is repeated over repeated intervals, as for the repeat mode. However, in each repeated interval, the ordering of color stops is the reverse of the adjacent interval. By analogy, given a sequence "ABC", it is extended to "...*ABC CBA* ABC *CBA ABC*...".

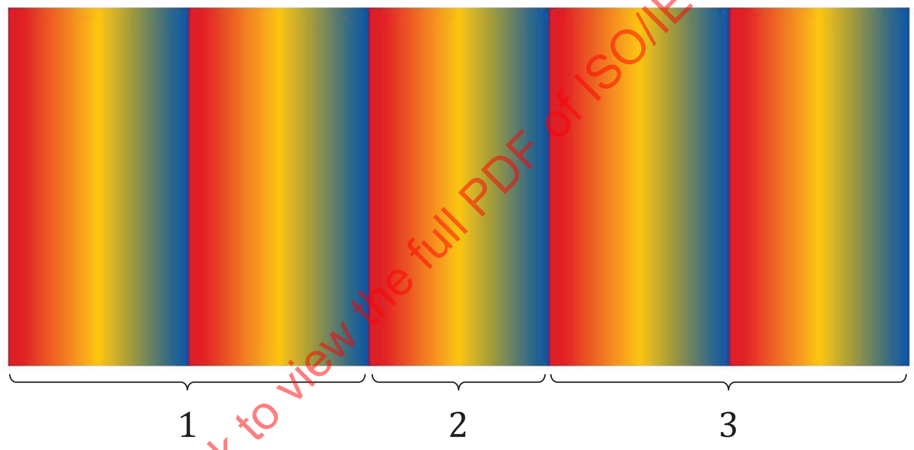
Figures 5.9–5.11 illustrate the different color line extend modes. The figures show the color line extended over a limited interval, but the extension is unbounded in either direction.



Key

1	pad with starting color
2	defined interval
3	pad with ending color

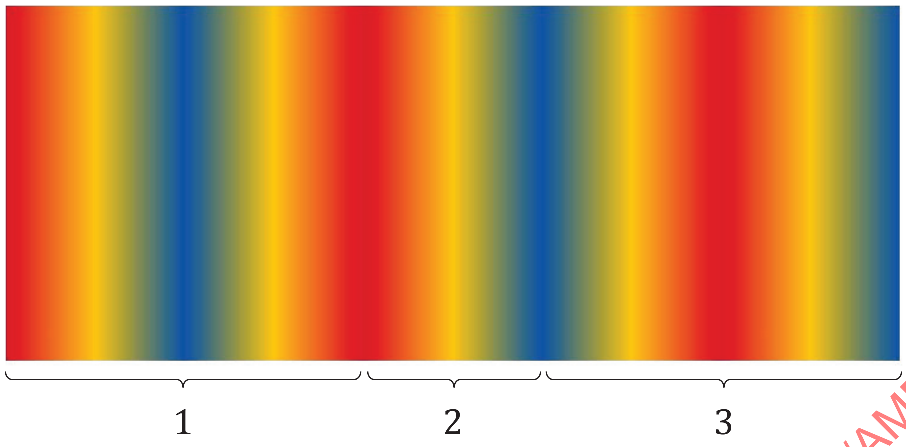
Figure 5.9 — Color gradation extended using pad mode



Key

1	repeated intervals
2	defined interval
3	repeated intervals

Figure 5.10 — Color gradation extended using repeat mode



Key

1	reflected intervals
2	defined interval
3	reflected intervals

Figure 5.11 — Color gradation extended using reflect mode

NOTE 1 The extend modes are the same as the spread Method attribute used for linear and radial gradients in the Scalable Vector Graphics (SVG) 1.1 (2nd Edition) specification.

When combining a color line with the geometry of a particular gradient definition, one might want to achieve a certain number of repetitions of the gradient pattern over a particular geometric range. Assuming that geometric range will correspond to placement of stop offsets 0 and 1, the following steps can be used:

— In order to get a certain number of repetitions of the gradient pattern (without reflection), divide 1 by the number of desired repetitions, use the result as the maximum stop offset for specified color stops, and set the extend mode to *repeat*.

— In order to get a certain number of repetitions of the reflected gradient pattern, divide 1 by two times the number of desired repetitions, use the result as the maximum stop offset for specified color stops, and set the extend mode to *reflect*.

NOTE 2 Special considerations apply to color line extend modes for sweep gradients. See 5.7.11.1.2.5 for details.

Color lines are specified using color line tables, which contain arrays of color stop records. Two color line table and two color stop record formats are defined:

— ColorLine table and ColorStop record

— VarColorLine table and VarColorStop record

The VarColorLine and VarColorStop formats can be used in variable fonts and allow for stop offsets and color alpha values to be variable. The ColorLine and ColorStop formats provide a more compact representation when variation is not required. See 5.7.11.2.5 for format details.

5.7.11.1.2.3 Linear gradients

A linear gradient provides gradation of colors along a straight line. The gradient is defined by three points, p_0, p_1 and p_2, plus a color line. The color line is positioned in the design grid with stop offset 0 aligned to p_0 and stop offset 1.0 aligned to p_1. (The line passing through p_0 and p_1 will be referred to as line p_0p_1.) Colors at each position on line p_0p_1 are interpolated using the color line. For each position along line p_0p_1, the color at that position is projected on either side of the line.

The additional point, p_2, is used to rotate the gradient orientation in the space on either side of the line p_0p_1. The line passing through points p_0 and p_2 (line p_0p_2) determines the direction in which colors are projected on either side of the color line. That is, for each position on line p_0p_1, the line that passes through that position on line p_0p_1 and that is parallel to line p_0p_2 will have the color for that position on line p_0p_1.

NOTE 1 For convenience, point p_2 can be referred to as the *rotation point*, and the vector from p_0 to p_2 can be referred to as the *rotation vector*. However, neither the magnitude of the vector nor the direction (from p_0 to p_2, versus from p_2 to p_0) has significance.

If either point p_1 or p_2 is the same as point p_0, the gradient is ill-formed and shall not be rendered.

If line p_0p_2 is parallel to line p_0p_1 (or near-parallel for an implementation-determined definition), then the gradient is ill-formed and shall not be rendered.

NOTE 2 An implementation can derive a single vector, from p_0 to a point p_3, by computing the orthogonal projection of the vector from p_0 to p_1 onto a line perpendicular to line p_0p_2 and passing through p_0 to obtain point p_3. The linear gradient defined using p_0, p_1 and p_2 as described above is functionally equivalent to a linear gradient defined by aligning stop offset 0 to p_0 and aligning stop offset 1.0 to p_3, with each color projecting on either side of that line in a perpendicular direction. This specification uses three points, p_0, p_1 and p_2, as that provides greater flexibility in controlling the placement and rotation of the gradient, as well as variations thereof.

Figures 5.12 to 5.14 illustrate linear gradients using the three different color line extend modes. Each figure illustrates linear gradients with two different rotation vectors. In each case, three color stops are specified: red at 0.0, yellow at 0.5, and blue at 1.0.

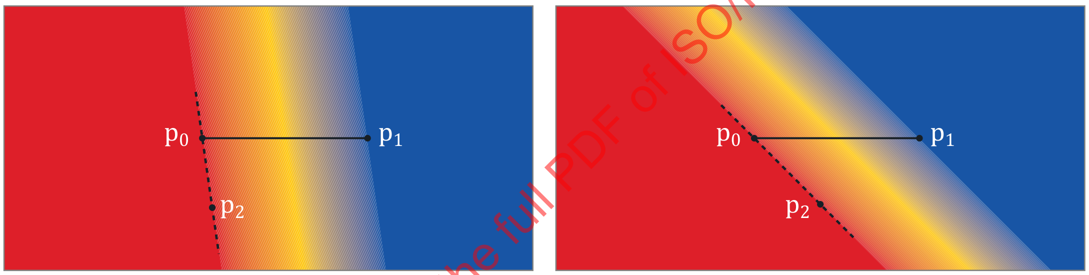


Figure 5.12 — Linear gradients with different rotations using the pad extend mode

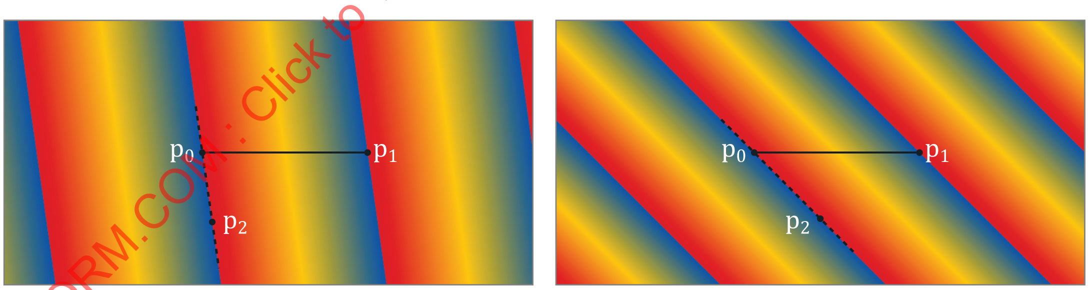


Figure 5.13 — Linear gradients with different rotations using the repeat extend mode

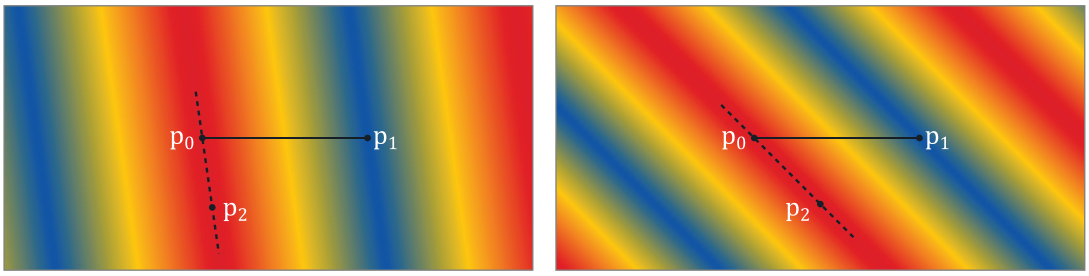


Figure 5.14 — Linear gradients with different rotations using the reflect extend mode

NOTE 3 When a linear gradient is combined with a transformation (see 5.7.11.1.5), the appearance will be the same as if the gradient were defined using the transformed positions of points p_0, p_1 and p_2.

Linear gradients are specified using a PaintVarLinearGradient or PaintLinearGradient table, with or without variation support, respectively. See 5.7.11.2.6.3 for format details.

See 5.7.11.1.3 for details on how fills are applied to a shape.

5.7.11.1.2.4 Radial gradients

A radial gradient provides gradation of colors along a cylinder defined by two circles. The gradient is defined by circles with center c_0 and radius r_0, and with center c_1 and radius r_1, plus a color line. The color line aligns with the two circles by associating stop offset 0 with the first circle (with center c_0) and aligning stop offset 1.0 with the second circle (with center c_1).

NOTE 1 The term “radial gradient” is used in some contexts for more limited capabilities. In some contexts, the type of gradient defined here is referred to as a “two point conical” gradient.

The drawing algorithm for radial gradients follows the HTML WHATWG Canvas specification for createRadialGradient() [32], but adapted with alternate color line extend modes, as described in 5.7.11.1.2.2. Radial gradients shall be rendered with results that match the results produced by the following steps.

With circle center points c_0 and c_1 defined as $c_0 = (x_0, y_0)$ and $c_1 = (x_1, y_1)$:

1) If $c_0 = c_1$ and $r_0 = r_1$ then paint nothing and return.

2) For real values of ω: Let $x(\omega) = (x_1\text{-}x_0)\omega + x_0$ Let $y(\omega) = (y_1\text{-}y_0)\omega + y_0$ Let $r(\omega) = (r_1\text{-}r_0)\omega + r_0$ Let the color at ω be the color at position ω on the color line.

3) For all values of ω where $r(\omega) > 0$, starting with the value of ω nearest to positive infinity and ending with the value of ω nearest to negative infinity, draw the circular line with radius $r(\omega)$ centered at position $(x(\omega), y(\omega))$, with the color at ω, but only painting on the parts of the bitmap that have not yet been painted on in this step of the algorithm for earlier values of ω.

The algorithm provides results in various cases as follows:

— When the circles are identical, then nothing is painted.

— When both radii are 0 ($r_0 = r_1 = 0$), then $r(\omega)$ is always 0 and nothing is painted.

— If the centers of the circles are distinct, the radii of the circles are different, and neither circle is entirely contained within the radius of the other circle, then the resulting shape resembles a cone that is open to one side. The surface outside the cone is not painted (see Figures 5.15 to 5.17).

— If the centers of the circles are distinct but the radii are the same, and neither circle is contained within the other, then the result will be a strip, similar to the flattened projection of a circular cylinder. The surface outside the strip is not painted (see Figures 5.18 to 5.20).

— If the radii of the circles are different but one circle is entirely contained within the radius of the other circle, the gradient will radiate in all directions from the inner circle, and the entire surface will be painted (see Figures 5.24 to 5.26).

Figures 5.15 to 5.17 illustrate radial gradients using the three different color line extend modes. The color line is defined with stops for the interval [0, 1]: red at 0.0, yellow at 0.5, and blue at 1.0. Note that the circles that define the gradient are not stroked as part of the gradient itself. Stroked circles have been overlaid in the figure to illustrate the color line and the region that is painted in relation to the two circles.

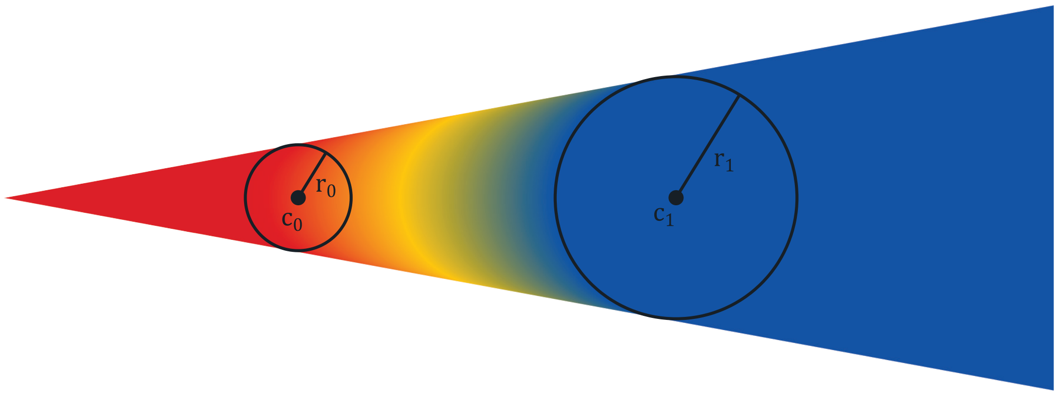


Figure 5.15 — Radial gradient using pad extend mode.

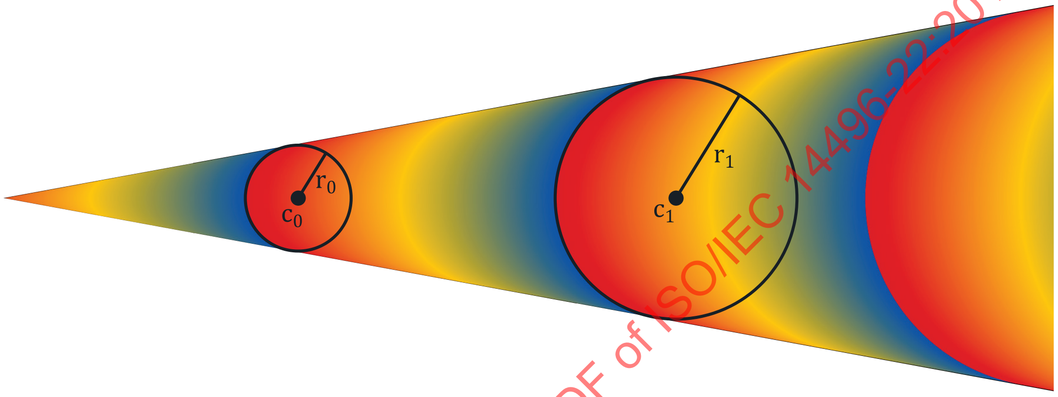


Figure 5.16 — Radial gradient using repeat extend mode.

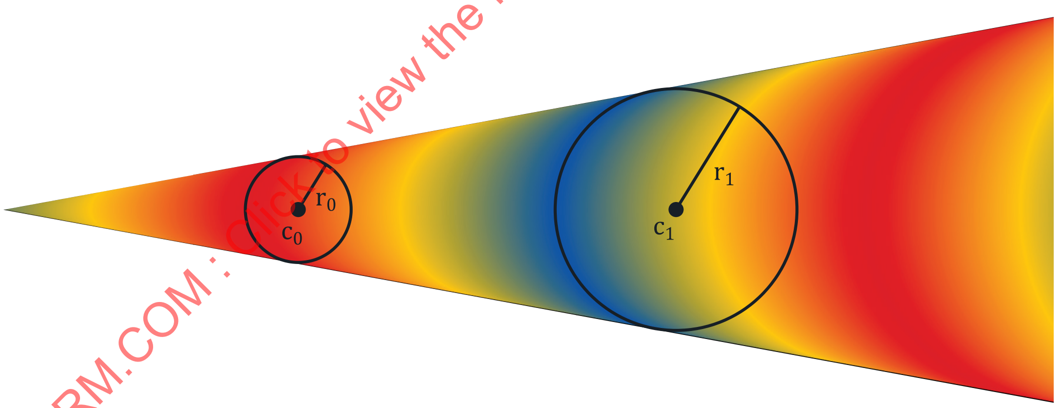


Figure 5.17 — Radial gradient using reflect extend mode.

Figures 5.18 to 5.20 illustrate the case in which the circles have distinct centers but the same radii, and neither circle is contained within the other, giving the appearance of a strip. The color stops are as in the previous figures.

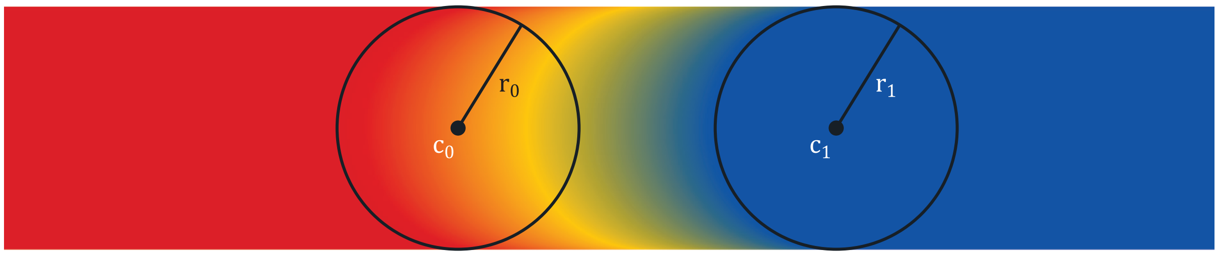


Figure 5.18 — Radial gradient with same-size circles appearing as a strip, using pad extend mode.

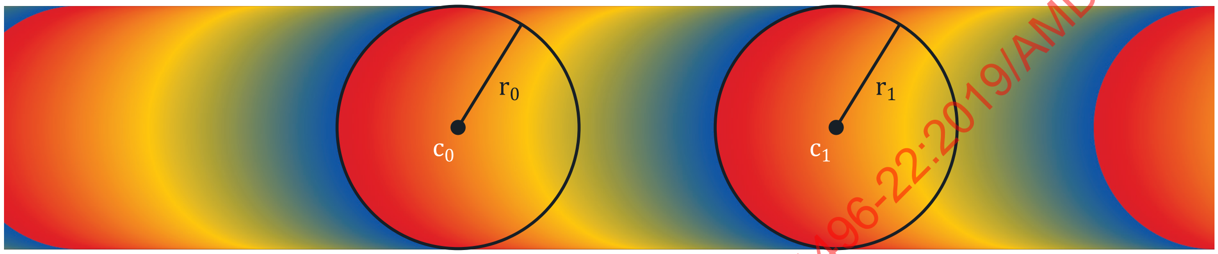


Figure 5.19 — Radial gradient with same-size circles appearing as a strip, using repeat extend mode.

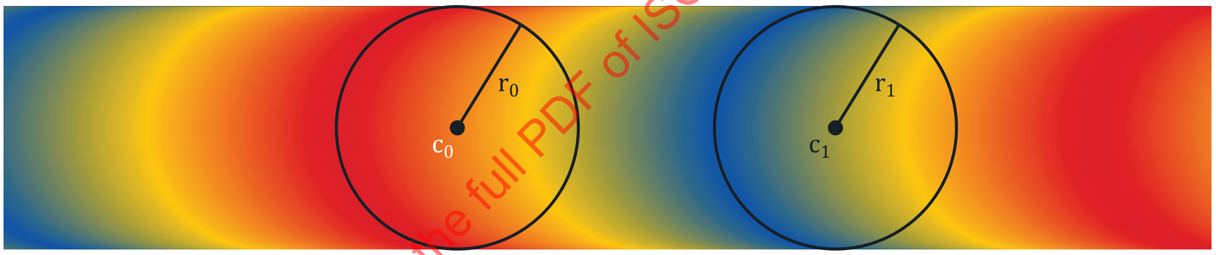


Figure 5.20 — Radial gradient with same-size circles appearing as a strip, using reflect extend mode.

Because the rendering algorithm progresses ω in a particular direction, from positive infinity to negative infinity, and because pixels are not re-painted as ω progresses, the appearance will be affected by which circle is considered circle 0 and which is circle 1. This is illustrated in Figures 5.21 – 5.23. The gradient in Figure 5.21 is the same as that in Figure 5.15, using the pad extend mode. In this gradient, circle 0 is the small circle, on the left. In Figure 5.22, the start and end circles are reversed: circle 0 is the large circle, on the right. The color line is kept the same, and so the red end starts at circle 0, now on the right. In Figure 5.23, the order of stops in the color line is also reversed to put red on the left. The key difference to notice between the gradients in these Figures is the way that colors are painted in the interior: when the two circles are not overlapping, the arcs of constant color bend in the same direction as the near side of circle 1.

NOTE 2 This difference does not exist if one circle is entirely contained within the other: in that case, the arcs of constant color are complete circles.

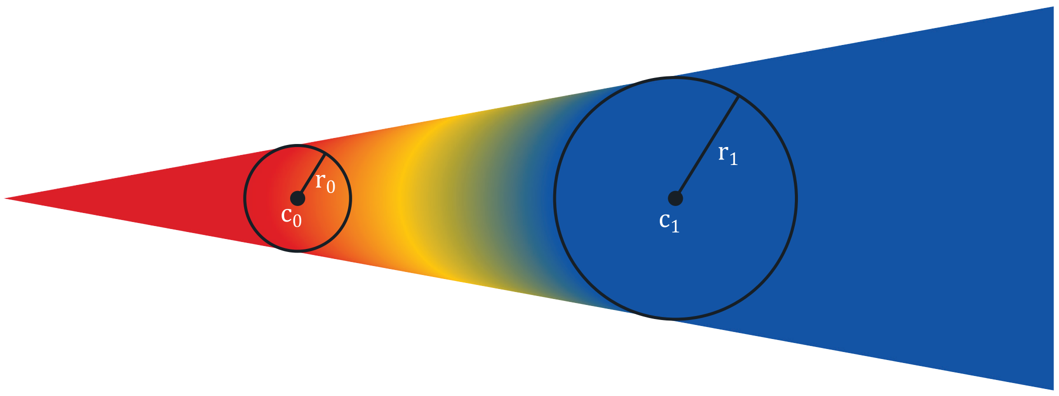


Figure 5.21 — Cone-shaped radial gradient with circle 0 on the left.

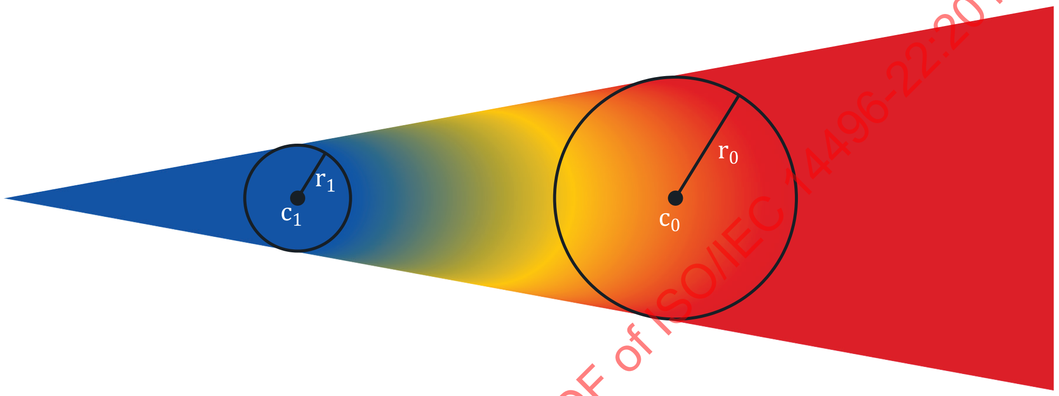


Figure 5.22 — Cone-shaped radial gradient with start and end circles swapped.

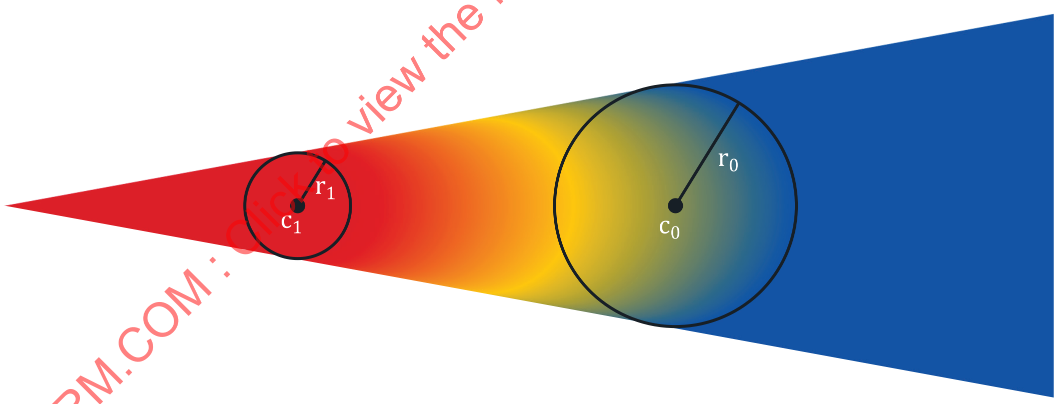


Figure 5.23 — Cone-shaped radial gradient with start and end circles swapped and color line reversed.

When one circle is contained within the other, the extension of the gradient beyond the larger circle will fill the entire surface. Colors in the areas inside the inner circle and outside the outer circle are determined by the extend mode. Figures 5.24 – 5.26 illustrate this for the different extend modes.

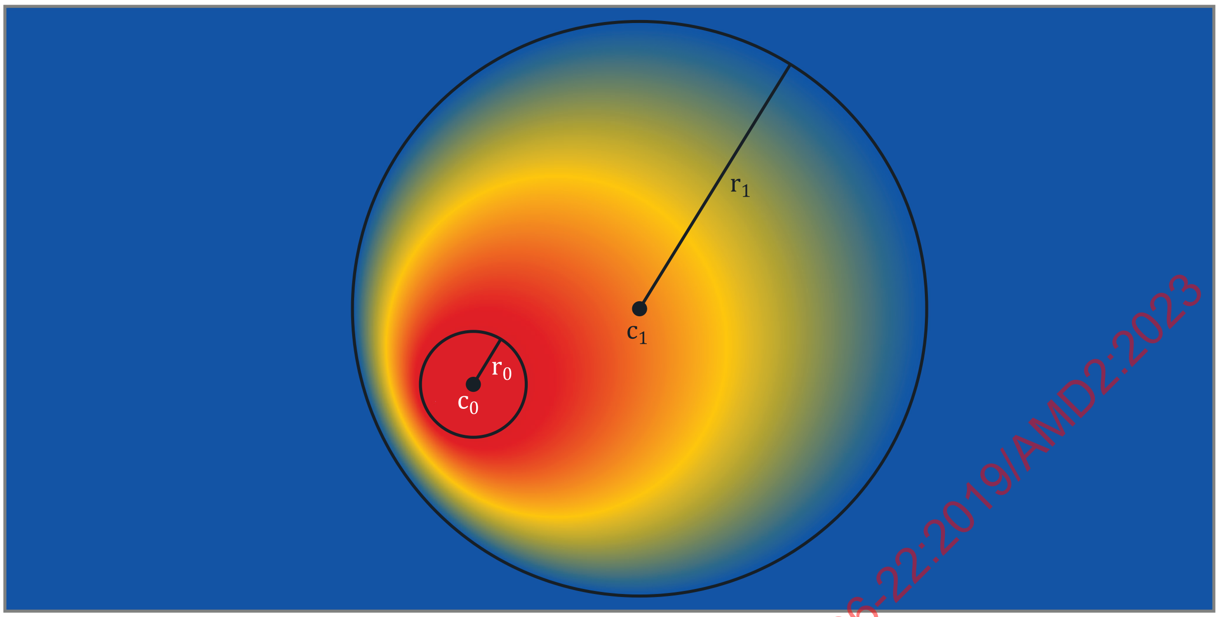


Figure 5.24 — Radial gradient with one circle contained within the other, pad extend mode.

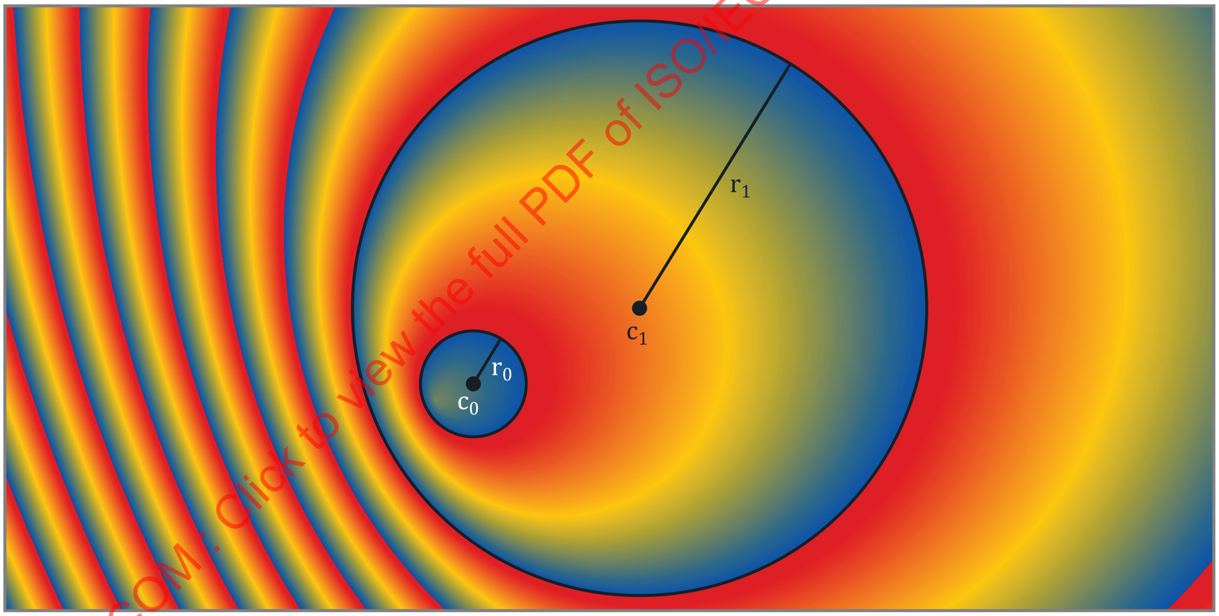


Figure 5.25 — Radial gradient with one circle contained within the other, repeat extend mode.

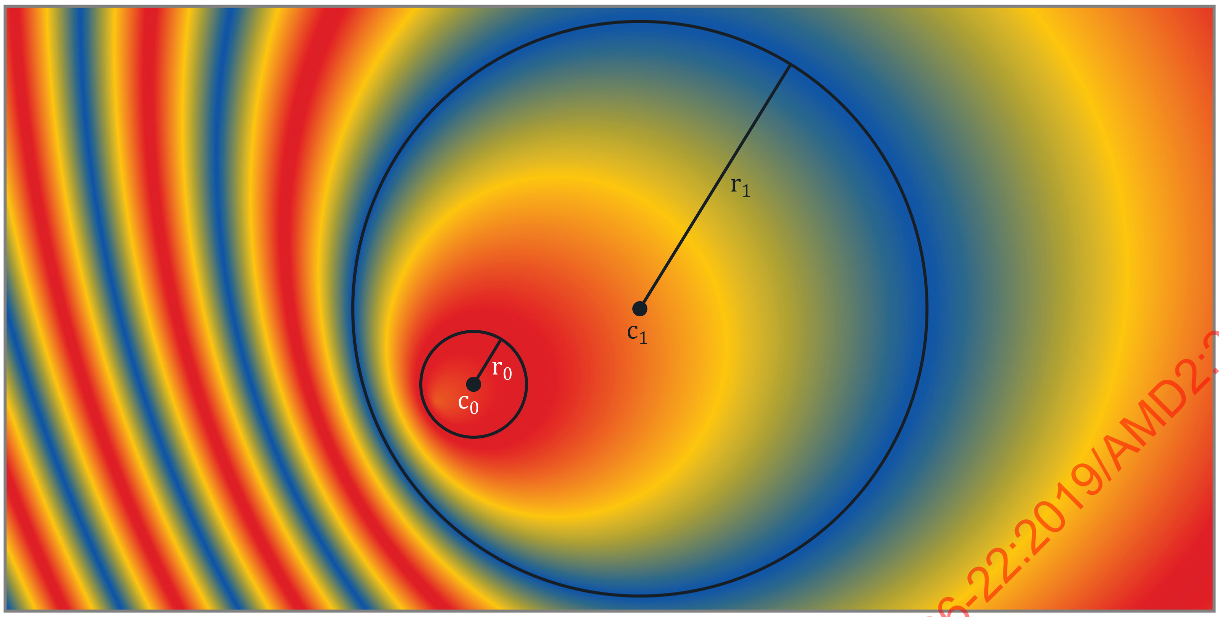


Figure 5.26 — Radial gradient with one circle contained within the other, reflect extend mode.

NOTE 3 When a radial gradient is combined with a transformation (see 5.7.11.1.5), the appearance will be the same as if the geometry of the two circles were transformed and step 3 of the algorithm were performed by interpolating the shapes derived from the two transformed circles. For the condition r(ω) > 0, the pre-transformation values of r(ω) can be used.

NOTE 4 A scale transformation can flatten shapes to resemble lines. If a radial gradient is nested in the child sub-graph of a transformation that flattens the circles so that they are nearly lines, the centers could still be separated by some distance. In that case, the radial gradient would appear as a strip or a cone filled with a linear gradient.

If a radial gradient is nested in the sub-graph of a transformation that flattens the circles so that they form a single line (or nearly a line, for an implementation-determined definition), with both centers on that line, then the resulting gradient is degenerate and shall not be rendered.

NOTE 5 As seen in the figures above, the gradient fills the space when one circle is contained within the other, but not when neither circle is contained within the other. In a variable font, if the placement or radii of the circles vary, then a sharp transition can occur if the variation results in one circle being contained within the other for some instances but not for other instances. This transition will occur when the inner circle just touches the outer circle (i.e., they have exactly one point in common). In this case, the gradient will fill exactly one half of the space. This is illustrated in Figure 5.27 using the pad extend mode.

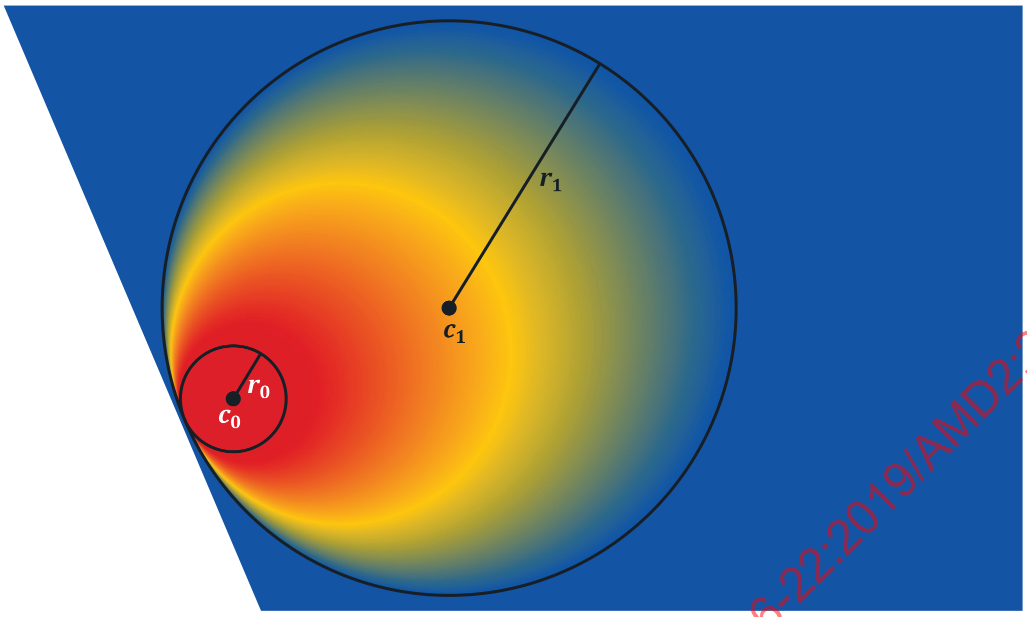


Figure 5.27 — Radial gradient with inner circle just touching the outer circle, pad extend mode

When the repeat or reflect extend modes are used, having the two circles in very close proximity results in very high spatial-frequency transitions that can lead to Moiré patterns or other display artifacts. This is illustrated in Figure 5.28, which shows the display result, for one particular rendering context, of a radial gradient defined using nearly-identical circles and the reflect extend mode.

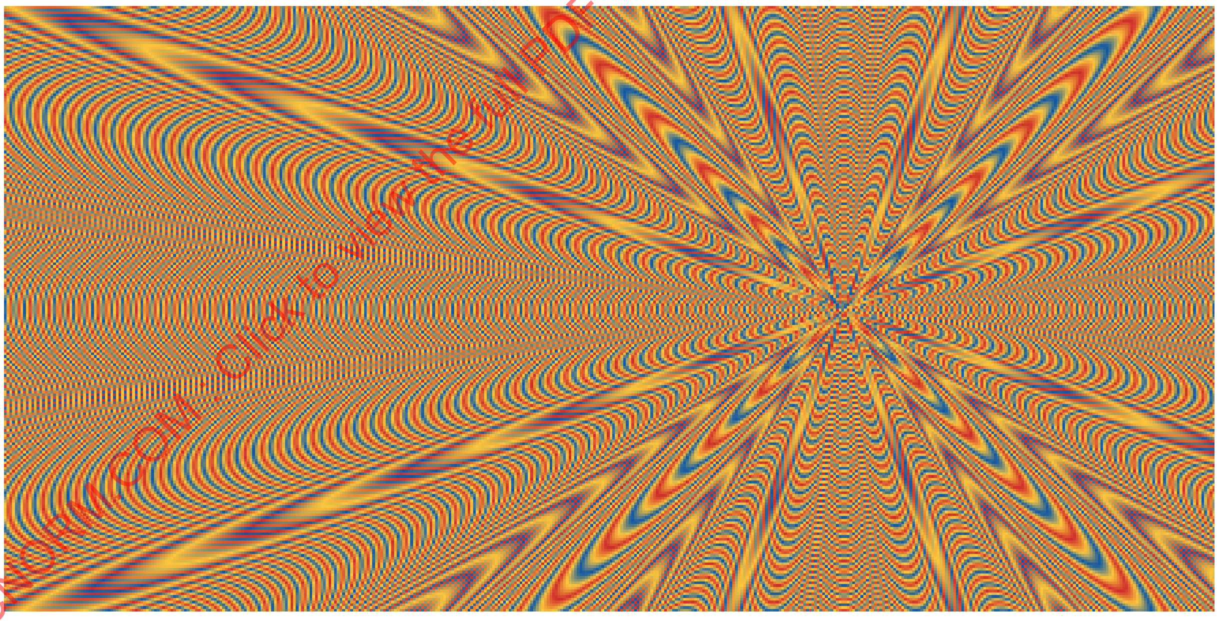


Figure 5.28 — Radial gradient defined using nearly-identical circles, showing interference patterns

The artifacts seen can be affected by a combination of several factors, such as image scaling, sub-pixel rendering, display technology, and limitations in software implementation or display capabilities. For this reason, the appearance can be very different in different situations. Font designers should exercise caution if the circles are in close proximity (either in a static design or for some variable font instances), and should not rely on these display artifacts to obtain a particular pattern.

Radial gradients are specified using a PaintVarRadialGradient or PaintRadialGradient table, with or without variation support, respectively. See 5.7.11.2.6.4 for format details.

See 5.7.11.1.3 for details on how fills are applied to a shape.

5.7.11.1.2.5 Sweep gradients

A sweep gradient provides a gradation of colors that sweep around a center point. For a given color on a color line, that color projects as a ray from the center point in a given direction. This is illustrated in Figure 5.29.

NOTE 1 The following figures illustrate sweep gradients clipped to a circular region. Sweep gradients are not bounded, however, and fill the entire space.

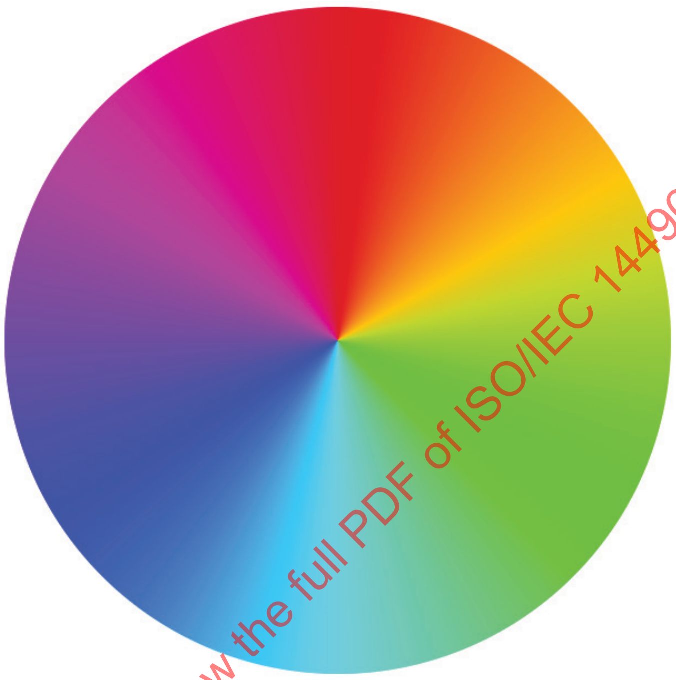


Figure 5.29 — Sweep gradient

NOTE 2 In some contexts, this type of gradient is referred to as a "conic" gradient, or as an "angular" gradient.

A sweep gradient is defined by a center point, starting and ending angles, and a color line. The angles are expressed in counter-clockwise degrees from the direction of the positive x-axis on the design grid.

The color line is aligned to a circular arc around the center point, with arbitrary radius, with stop offset 0 aligned with the starting angle, and stop offset 1 aligned with the ending angle. The color line progresses from the start angle to the end angle in the counter-clockwise direction; for example, if the start and end angles are both 0°, then stop offset 0.1 is at 36° counter-clockwise from the direction of the positive x-axis. For each position along the circular arc, from start to end in the counter-clockwise direction, a ray from the center outward is painted with the color of the color line at the point where the ray passes through the arc.

The color line may be defined using color stops outside the range [0, 1], and color stops outside the range [0, 1] can be used to interpolate color values within the range [0, 1], but only color values for the range [0, 1] are painted. If the specified color stops cover the entire [0, 1] range (or beyond), then the extend mode is not relevant and may be ignored. If the specified color stops do not cover the entire [0, 1] range, the extend mode is used to determine color values for the remainder of that range. For example, if a color line is specified with two color stops, red at stop offset 0.3 and yellow at stop offset 0.6, and the pad extend mode is specified, then the extend mode is used to derive color values from 0.0 to 0.3 (red), and from 0.6 to 1.0 (yellow).

Because a sweep gradient is defined using start and end angles, the gradient does not need to cover a full 360° sweep around the center. This is illustrated in Figure 5.30:

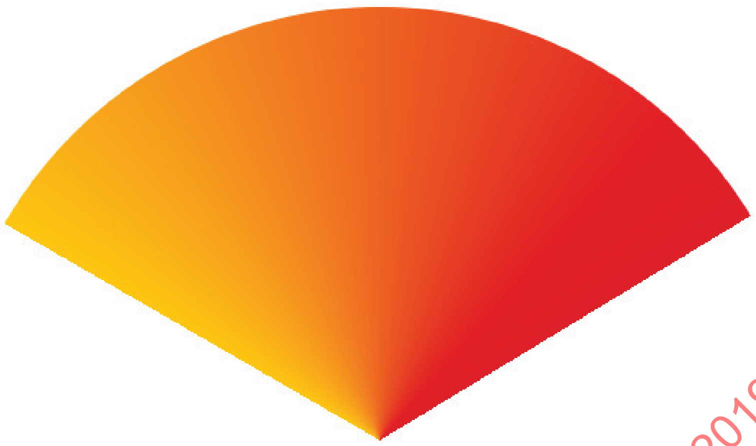


Figure 5.30 — A sweep gradient with start angle of 30° and an end angle of 150°

Start and end angle values can be outside the range [0, 360), and are converted to values within that range by applying a modulus operation. For example, an angle -60° is treated the same as 300. As a consequence, the [0, 1] range of the color line covers at most one full rotation around the center, never more.

If the starting and ending angle are the same, a sharp color transition can occur if the colors at stop offsets 0 and 1 are different. This is illustrated in Figure 5.31, showing a gradient from red to yellow that starts and stops at 0°.

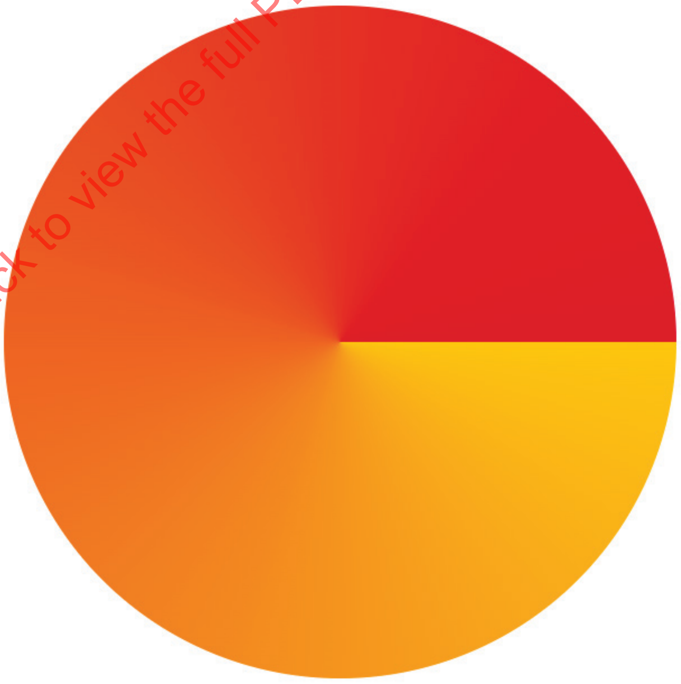


Figure 5.31 — A sweep gradient with a sharp transition at the start/end angle 0°

To avoid such a sharp transition, the stop offsets 0 and 1 on the color line need to have the same color value. Figure 5.32 illustrates a sweep gradient that transitions from red at stop offset 0, to yellow at stop offset 0.5, and back to red at stop offset 1.0.

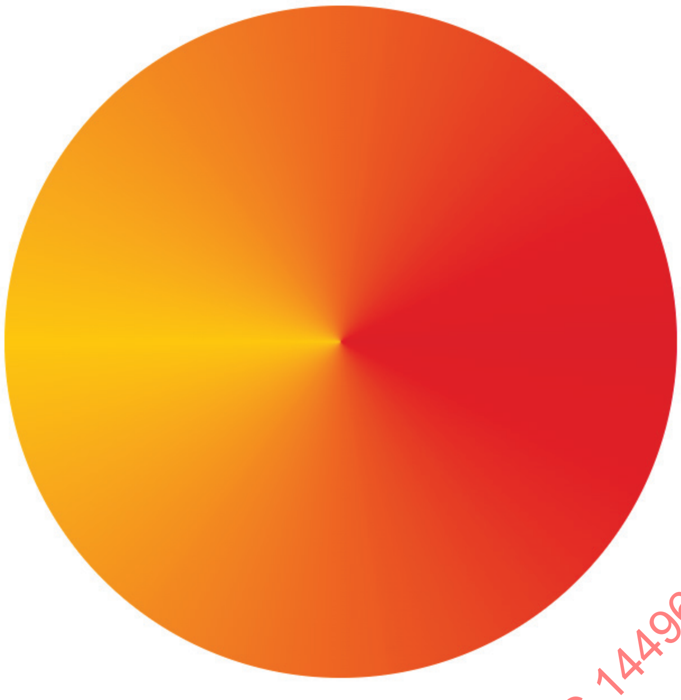


Figure 5.32 — A sweep gradient with a smooth transition at the start/end angle 0°

NOTE 3 When a sweep gradient is combined with a transformation (see 5.7.11.1.5), the appearance will be the same as if a circular arc of some non-zero radius were computed from the start and end angles; the center point and arc transformed; the color line aligned to the transformed arc; and then a gradient derived from the result, with rays from the transformed center point passing through the transformed color arc. When aligning the color line to the transformed arc, stop offset 0 would be aligned to the transformed point derived from the start angle, with stop offset 1 aligned to the transformed point derived from the end angle. Thus, a transform can result in the color line progressing in a clockwise rather than counter-clockwise direction.

Sweep gradients are specified using a PaintVarSweepGradient or PaintSweepGradient table, with or without variation support, respectively. See 5.7.11.2.6.5 for format details.

See 5.7.11.1.3 for details on how fills are applied to a shape.

5.7.11.1.3 Filling shapes

All basic shapes used in a color glyph are obtained from glyph outlines, referenced using a glyph ID. In a color glyph description, a PaintGlyph table is used to represent a basic shape.

NOTE Shapes can also be derived using PaintGlyph tables in combination with other tables, such as PaintTransform (see 5.7.11.1.5) or PaintComposite (see 5.7.11.1.6).

The PaintGlyph table has a field for the glyph ID, plus an offset to a child paint table that is used as the fill for the shape. The glyph outline is not rendered; only the fill is rendered.

Any of the basic fill formats (PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, PaintVarSweepGradient) can be used as the child paint table. This is illustrated in Figure 5.33: a PaintGlyph table has a glyph ID for an outline in the shape of a triangle, and it links to a child PaintLinearGradient table. The combination is used to represent a triangle filled with the linear gradient.

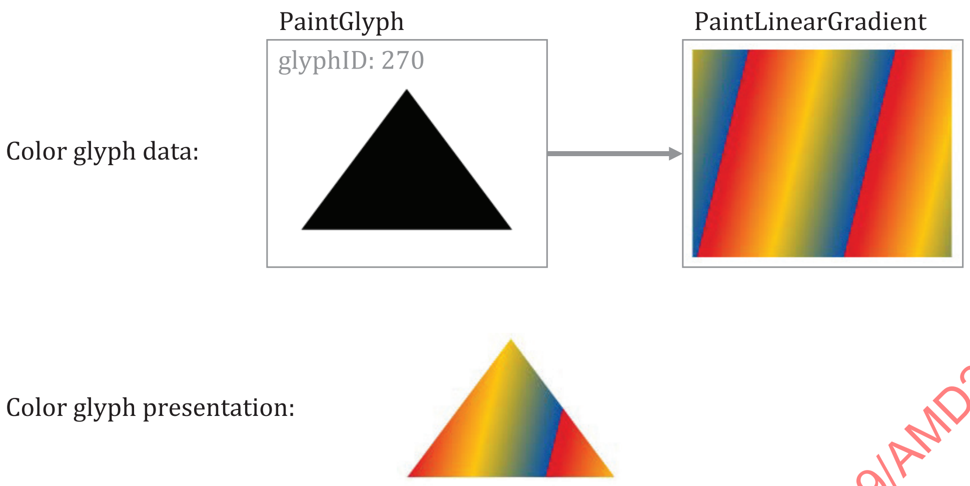


Figure 5.33 — PaintGlyph and PaintLinearGradient tables used to fill a shape with a linear gradient.

The child of a PaintGlyph table is not, however, limited to one of the basic fill formats. Rather, the child can be the root of a sub-graph that describes some graphic composition that is used as a fill. Another way to describe the relationship between a PaintGlyph table and its child sub-graph is that the glyph outline specified by the PaintGlyph table defines a bounds, or *clip region*, that is applied to the fill composition defined by the child sub-graph.

To illustrate this, the example in Figure 5.33 is extended in Figure 5.34 so that a PaintGlyph table links to a second PaintGlyph that links to a PaintLinearGradient: the parent PaintGlyph will clip the filled shape described by the child sub-graph.

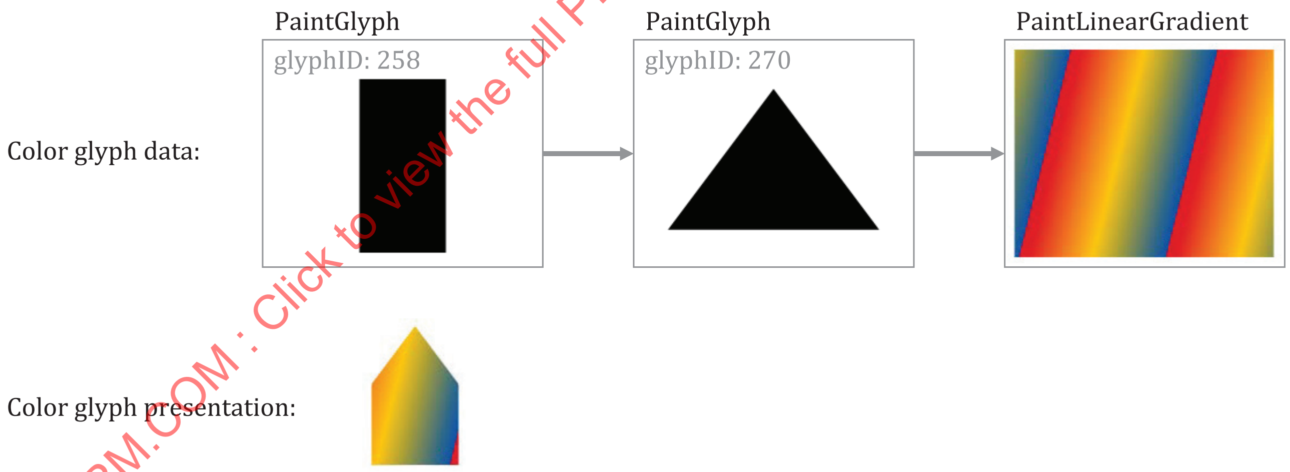


Figure 5.34 — A PaintGlyph table defines a clip region for the composition defined by its child sub-graph.

A PaintGlyph table on its own does not add content: if there is no child paint table, then the graph is not well formed. See 5.7.11.1.9 for details regarding well-formedness and validity of the graph.

5.7.11.1.4 Layering

Layering of visual elements was introduced above, in the introduction to 5.7.11.1. Both version 0 and version 1 support use of multiple layers, though in different ways.

For version 0, layers are fundamental: they are the sole way in which separate elements are composed into a color glyph. An array of LayerRecords is created, with each LayerRecord specifying a glyph ID and a CPAL entry (a shape and solid color fill). Each color glyph definition is a slice from that array (that is, a contiguous sub-sequence), specified in a BaseGlyphRecord for a particular base glyph. Within a given slice, the first record specifies the content of the bottom layer, and each subsequent record specifies content that overlays the preceding content (increasing z-order). A single array is used for defining all color glyphs. The LayerRecord slices for two base glyphs may overlap, though often will not overlap.

Figure 5.35 illustrates layers using version 0 formats.

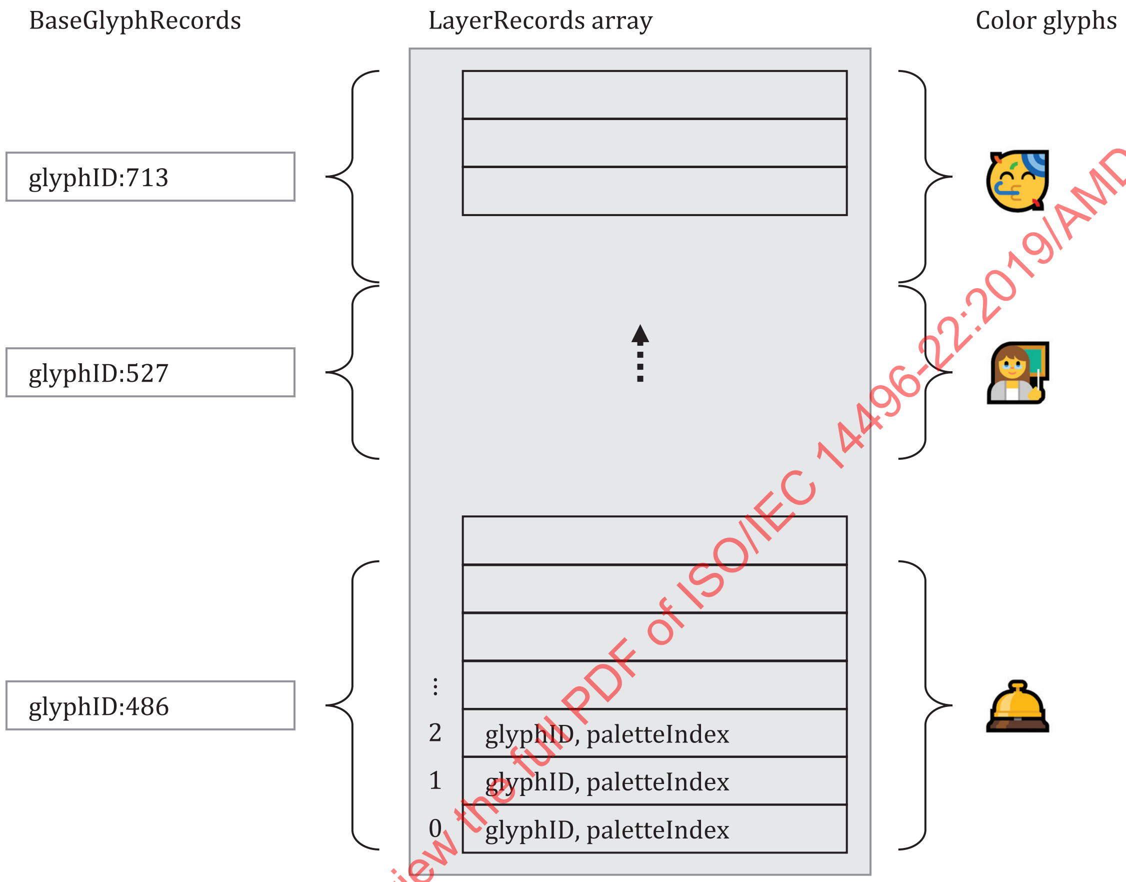


Figure 5.35 — Version 0: Color glyphs are defined by slices of a layer records array.

When using version 1 formats, use of multiple layers is supported but is optional. For example, a simple glyph description need not use any layering, as illustrated in Figure 5.36:

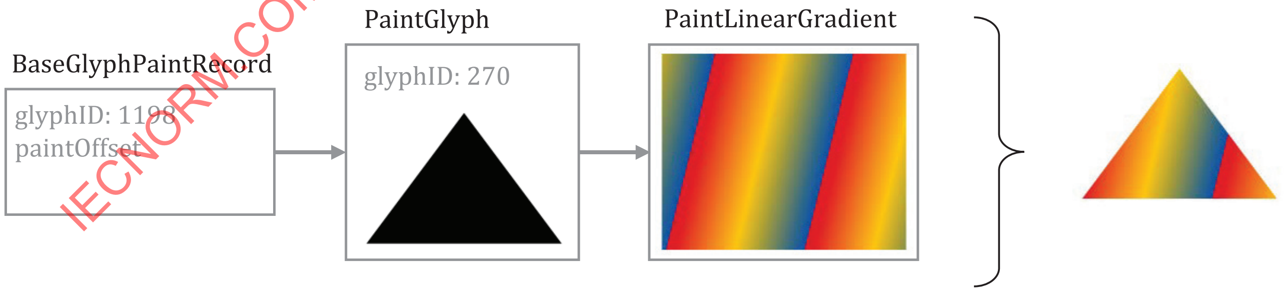


Figure 5.36 — Complete color glyph definition without use of layers.

The version 1 formats define a color glyph as a directed, acyclic graph of paint tables, and the concept of layering corresponds roughly to the number of distinct leaf nodes in the graph (see 5.7.11.1.9). The basic fill formats (PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, PaintVarSweepGradient) do not have child paint tables and so can only be leaf nodes in the graph. Some paint tables, such as the

PaintGlyph table, have only a single child, so can be used within a layer but do not provide any means of adding additional layers. Increasing the number of layers requires paint tables that have two or more children, creating a fork in the graph.

The version 1 formats include two paint formats that have two or more children, and so can increase the number of layers in the graph:

— The PaintComposite table allows two sub-graphs to be composed together using different compositing or blending modes.

— The PaintColrLayers table supports defining a sequence of several layers.

NOTE 1 The PaintColrGlyph table provides a means of incorporating the graph of one color glyph as a sub-graph in the definition of another color glyph. In this way, PaintColrGlyph provides an indirect means of introducing additional layers into a color glyph definition: forks in the resulting graph do not come from the PaintColrGlyph table itself, but can come from PaintColrLayers or PaintComposite tables that are nested in the incorporated sub-graph. See 5.7.11.1.7.4 for a description of the PaintColrGlyph table.

While the PaintComposite table only combines two sub-graphs, other PaintComposite tables can be nested to provide additional layers. The primary purpose of PaintComposite is to support compositing or blending modes other than simple alpha blending. The PaintComposite table is covered in more detail in 5.7.11.1.6. The remainder of this clause will focus on the PaintColrLayers table.

The PaintColrLayers table is used to define a bottom-up z-order sequence of layers. Similar to version 0, it defines a layer set as a slice in an array, but in this case the array is an array of offsets to paint tables, contained in a LayerList table. Each referenced paint table is the root of a sub-graph of paint tables that specifies a graphic composition to be used as a layer. Within a given slice, the first offset provides the content for the bottom layer, and each subsequent offset provides content that overlays the preceding content. Definition of a layer set—a slice within the layer list—is given in a PaintColrLayers table.

Figure 5.37 illustrates the organizational relationship between PaintColrLayers tables, the LayerList, and referenced paint tables that are roots of sub-graphs.

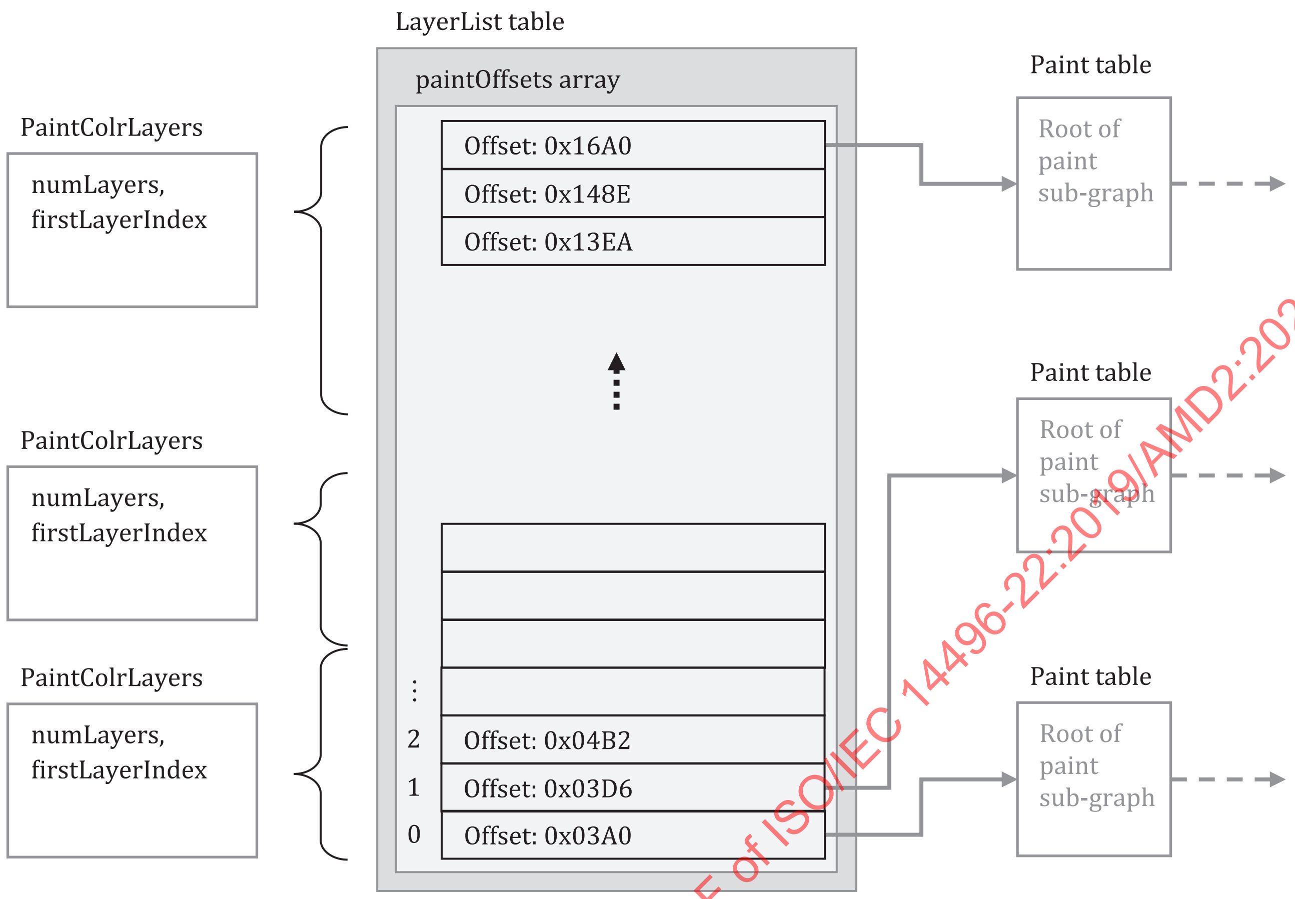


Figure 5.37 — Version 1: PaintColrLayers tables specify slices within the LayerList, providing a layering of content defined in sub-graphs.

NOTE 2 Paint table offsets in the LayerList table are only used in conjunction with PaintColrLayers tables. If a paint table does not need to be referenced via a PaintColrLayers table, its offset does not need to be included in the LayerList array.

A PaintColrLayers table can be used as the root of a color glyph definition, providing a base layering structure for the color glyph. In this usage, the PaintColrLayers table is referenced by a BaseGlyphPaintRecord, which specifies the root of the graph of a color glyph definition for a given base glyph. This is illustrated in Figure 5.38.

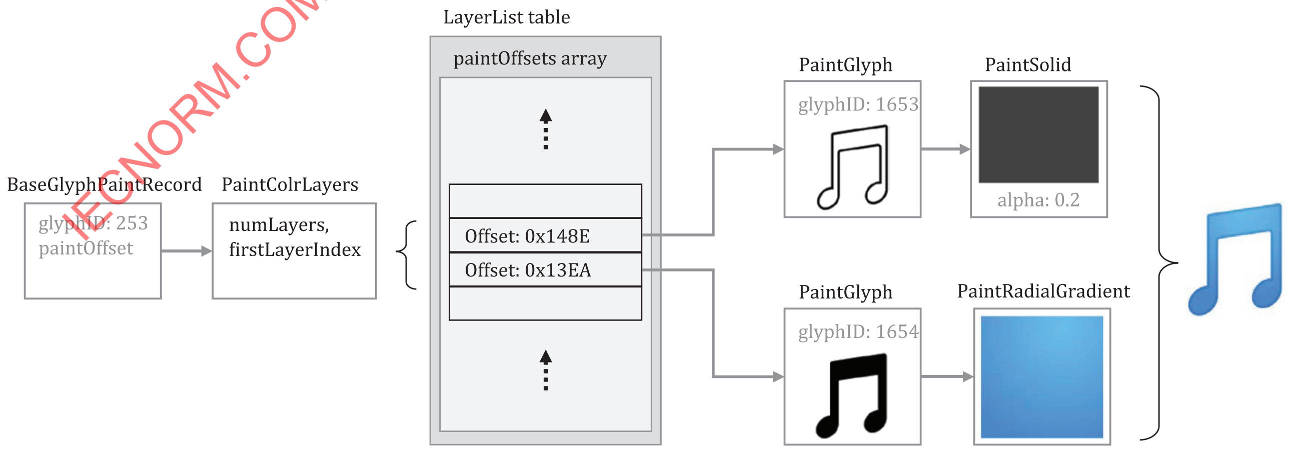


Figure 5.38 — PaintColrLayers table used as the root of a color glyph definition.

A PaintColrLayers table can also be nested more deeply within the graph, providing a layer structure to define some component within a larger color glyph definition (see 5.7.11.1.7.3 for more information). The ability to nest a PaintColrLayers table within a graph creates the potential to introduce a cycle within the graph, which would be invalid (see 5.7.11.1.9).

5.7.11.1.5 Transformations

A 2 × 3 transformation matrix can be used within a color glyph description to apply an affine transformation to a sub-graph. Affine transformations supported by a matrix can be a combination of scale, skew, mirror, rotate, or translate. The transformation is applied to all nested paints in the child sub-graph.

A transformation matrix is specified using a PaintVarTransform or PaintTransform table, with or without variation support, respectively. See 5.7.11.2.6.8 for format details.

The effect of a transformation is illustrated in Figure 5.39: a PaintTransform table is used to specify a rotation, and both the glyph outline and gradient in the sub-graph are rotated.

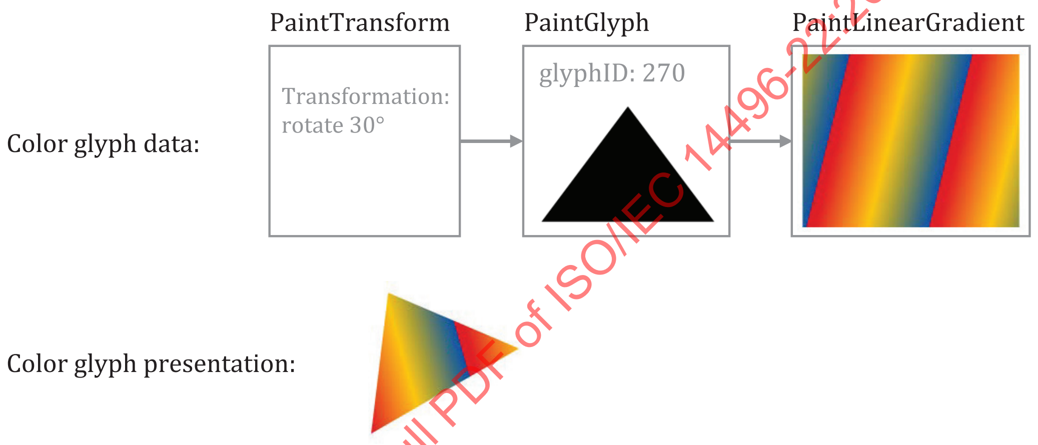


Figure 5.39 — A rotation transformation rotates the fill content defined by the child sub-graph.

If the sub-graph of a transformation table contains another nested transformation table, then the second transformation also applies to its child sub-graph. For the sub-sub-graph, the two transformations are combined. To illustrate this, the example in Figure 5.40 is extended in Figure 5.40 by inserting a mirroring transformation between the PaintGlyph and PaintLinearGradient tables: the glyph outline is rotated as before, but the gradient is mirrored in its (pre-rotation) y-axis as well as being rotated. Notice that both visible elements—the shape and the gradient fill—are affected by the rotation, but only the gradient is affected by the mirroring.

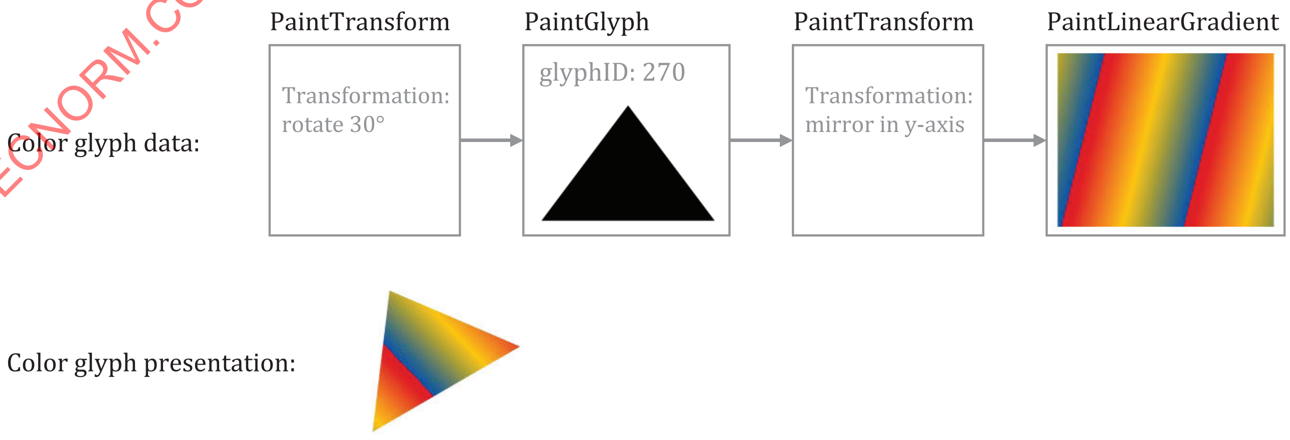


Figure 5.40 — Combined effects of a transformation nested within the child sub-graph of another transformation.

While the PaintTransform and PaintVarTransform tables support several types of transforms, additional paint formats are defined to support specific transformations:

— PaintTranslate and PaintVarTranslate support translation only, without or with variation support, respectively. See 5.7.11.2.6.9 for format details.

— PaintScale and PaintVarScale support scaling only, without or with variation support. These two formats scale relative to the origin, and allow for different scale factors in X and Y directions. PaintScaleAroundCenter, PaintVarScaleAroundCenter, PaintScaleUniform, PaintVarScaleUniform, PaintScaleUniformAroundCenter, and PaintVarScaleUniformAroundCenter support scaling relative to a different center, scaling uniformly in both X and Y directions, or both. See 5.7.11.2.6.10 for format details.

— PaintRotate and PaintVarRotate support rotation only, without or with variation support. These two formats rotate around the origin; the PaintRotateAroundCenter and PaintVarRotateAroundCenter formats support rotation around a different center. See 5.7.11.2.6.11 for format details.

— PaintSkew and PaintVarSkew support skew only, without or with variation support. These two formats skew using the origin as a center for the skew rotation; the PaintSkewAroundCenter and PaintVarSkewAroundCenter formats support skews using a different center. See 5.7.11.2.6.12 for format details.

NOTE 1 Horizontal mirroring is done by scaling using a scale factor in the x direction of -1. Vertical mirroring is done by scaling with a -1 scale factor in the y direction.

When only one of these specific types of transformation is required, these formats provide a more compact representation than the PaintTransform or PaintVarTransform formats. Another significant difference of the rotation and skew formats is that the rotations and skews are specified as angles, in counter-clockwise degrees.

NOTE 2 Specifying the rotation or skew as an angle can have a significant benefit in variable fonts if an angle of skew or rotation needs to vary, since it is easier to implement variation of angles when specified directly rather than as matrix elements. This is because the matrix elements for a rotation or skew are the sine, cosine or tangent of the rotation angle, which do not change in linear proportion to the angle. To achieve a linear variation of rotation using matrix elements would require approximating the variation using multiple delta sets.

The rotations and skews specified using PaintRotate, PaintSkew and their variants can also be represented as a matrix using a PaintTransform or PaintVarTransform table. The behavior for the PaintRotate or PaintSkew formats and their variants shall be the same as if the rotation or skew were represented using an equivalent matrix. See 5.7.11.2.6.11 for details regarding the matrix equivalent for a rotation expressed as an angle; and see 5.7.11.2.6.12 for similar details in relation to skews.

5.7.11.1.6 Compositing and blending

When a color glyph has overlapping content in two layers, the pixels in the two layers must be combined in some way. If the content in the top layer has full opacity, then normally the pixels from that layer are shown, occluding overlapping pixels from lower layers. If the top layer has some transparency (some portion has alpha less than 1.0), then blending of colors for overlapping pixels occurs by default. The default interaction between layers uses simple alpha compositing, as described in W3C Compositing and Blending Level 1 specification [33].

A PaintComposite table can be used to get other compositing or blending effects. The PaintComposite table combines content defined by two sub-graphs: a *source* sub-graph; and a destination, or *backdrop*, sub-graph. First, the paint operations for the backdrop sub-graph are executed, then the drawing operations for the source sub-graph are executed and combined with backdrop using a specified compositing or blending mode. The available modes are given in the CompositeModes enumeration (see 5.7.11.2.6.13). The effect and processing rule of each mode are specified in W3C Compositing and Blending Level 1 specification [33].

The available modes fall into two general types: compositing modes, also referred to as “Porter-Duff” modes; and blending modes. In rough terms, the Porter-Duff modes determine how much effect pixels from the source and the backdrop each contribute in the result, while blending modes determine how color values for pixels from the source and backdrop are combined. These are illustrated with examples in Figures 5.41 and 5.42: in each case, red and blue rectangles are the source and backdrop content.

Figure 5.41 shows the effect of a Porter-Duff mode, *XOR*, which has the effect that only non-overlapping pixels contribute to the result.

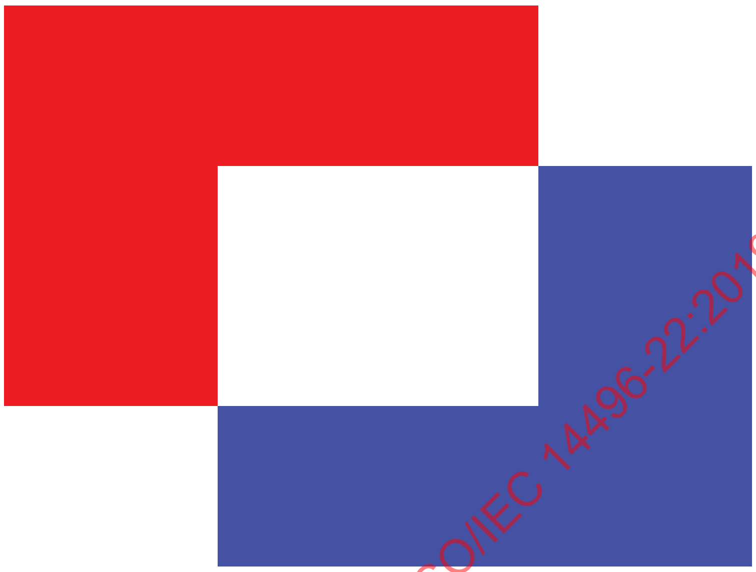


Figure 5.41 — Two content elements combined using the Porter-Duff *XOR* mode.

Figure 5.42 shows the effect of a *lighten* blending mode, which has the effect that the R, G, and B color components for each pixel in the result is the greater of the R, G, and B values from corresponding pixels in the source and backdrop.

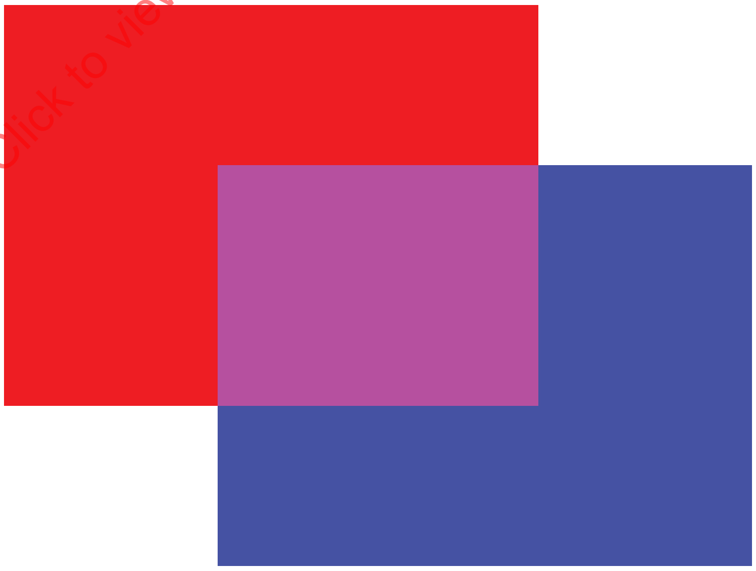


Figure 5.42 — Two content elements combined using the *lighten* blending mode.

For complete details on each of the Porter-Duff and blending modes, see W3C Compositing and Blending Level 1 specification [33].

Figure 5.43 illustrates how the PaintComposite table is used in combination with content sub-graphs to implement an alternate compositing effect. The source sub-graph defines a green capital A; the backdrop sub-graph defines a black circle. The compositing mode used is *Source Out*, which has the effect that the source content punches out a hole in the backdrop. (For this mode, the fill color of the source is irrelevant; a black or yellow "A" would have the same effect.) A red rectangle is included as a lower layer to show that the backdrop has been punched out by the source, making that portion of the lower layer visible.

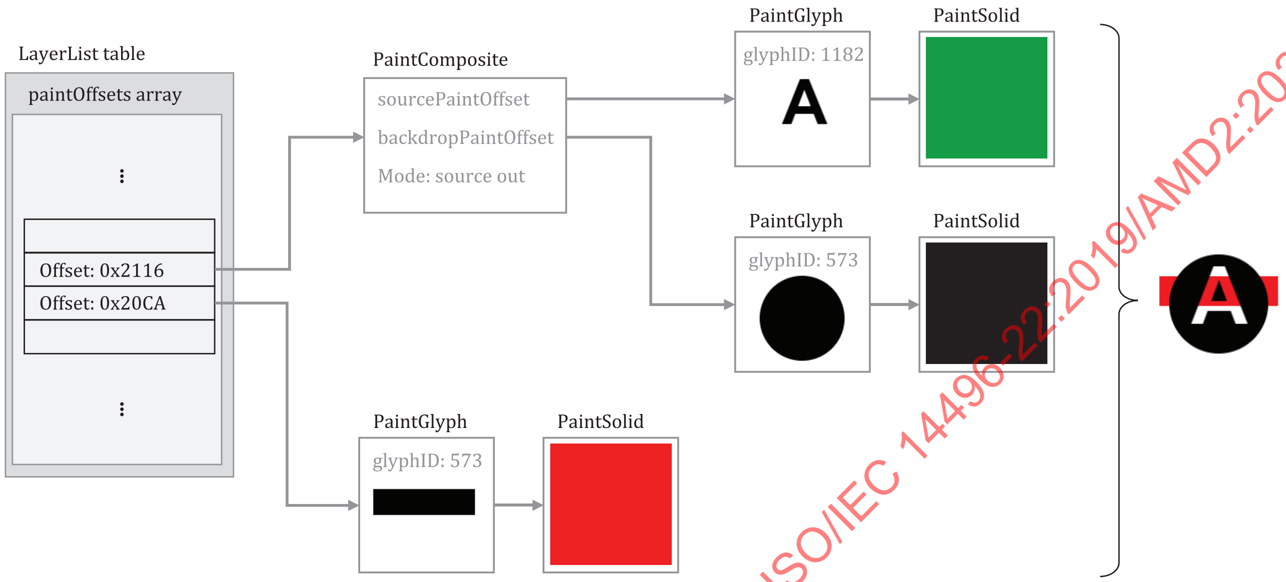


Figure 5.43 — A color glyph using a PaintComposite table to punch out a shape from the fill of a circle.

NOTE In Figure 5.43, the "A" is filled with green to illustrate that the color of the fill has no affect for the *Source Out* composite mode. Because that is the case, the black or red PaintSolid could have been re-used instead of adding a separate PaintSolid table. See 5.7.11.1.7.2 for more information on re-use of paint tables for such situations.

Scalable Vector Graphics (SVG) supports alpha channel masking using the <mask> element. The same effects can be implemented in COLR version 1 using a PaintComposite table by setting a pattern of alpha values in the source sub-graph and selecting the *Source In* composite mode. This is illustrated in Figure 5.44.

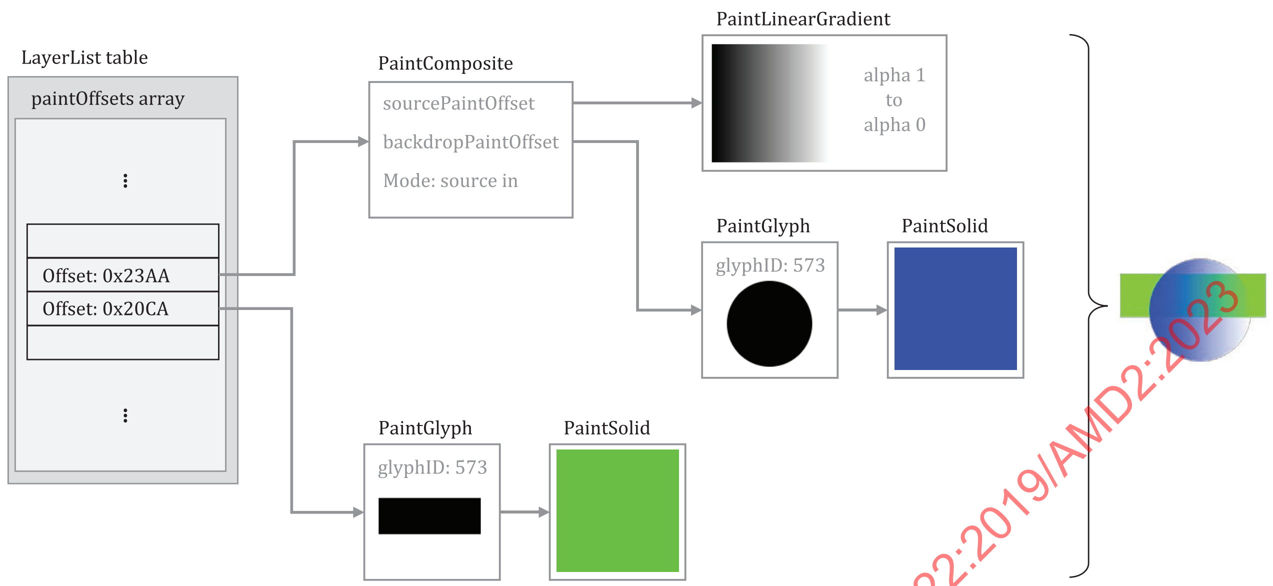


Figure 5.44 — An alpha mask implemented using a PaintComposite table and the *Source In* mode.

5.7.11.1.7 Re-usable components

5.7.11.1.7.1 Overview

Within a color font, many color glyphs might share components in common. For example, in emoji fonts, many different "smilies" or clock faces share a common background. This can be seen in Figure 5.45, which shows color glyphs for three emoji clock faces.

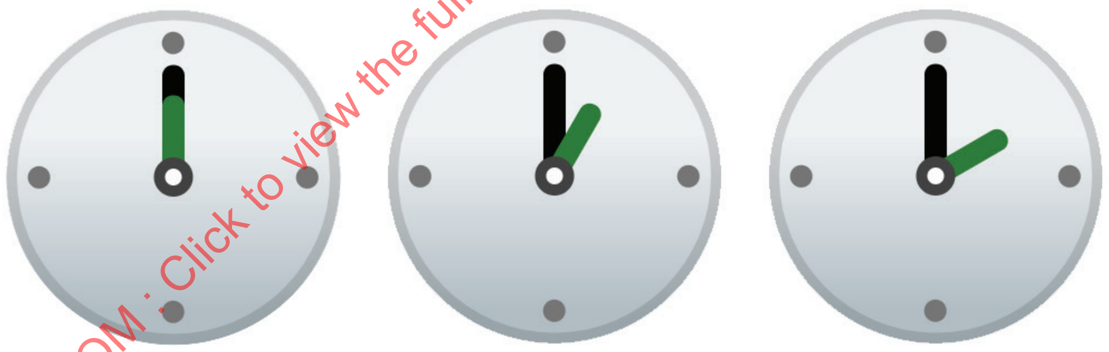


Figure 5.45 — Emoji clock faces for 12 o'clock, 1 o'clock and 2 o'clock.

Several components are shared between these color glyphs: the entire face, with a gradient background and dots at the 3, 6, 9 and 12 positions; the minute hand pointing to the 12 position; and the circles in the center. Also, note that the four dots have the same shape and fill, and differ only in their position. In addition, the hour hands have the same shape and fill, and differ only in their orientation.

There are several ways in which elements of a color glyph description can be re-used:

— Reference to shared subtables

— Use of a PaintColrLayers table

— Use of a PaintColrGlyph table

The PaintColrLayers and PaintColrGlyph table formats create a potential for introducing cycles within the graph of a color glyph, which would be invalid (see 5.7.11.1.9).

5.7.11.1.7.2 Re-use by referencing shared subtables

Several of the paint table formats link to a child paint table using a forward offset within the file:

— PaintGlyph

— PaintComposite

— PaintTransform, PaintVarTransform

— PaintTranslate, PaintVarTranslate

— PaintScale, PaintVarScale, and the other variant scaling paint formats

— PaintRotate, PaintVarRotate, PaintRotateAroundCenter, PaintVarRotateAroundCenter

— PaintSkew, PaintVarSkew, PaintSkewAroundCenter, PaintVarSkewAroundCenter

A child subtable can be shared by several tables of these formats. For example, several PaintGlyph tables might link to the same PaintSolid table, or to the same node for a sub-graph describing a more complex fill. The only limitation is that child paint tables are referenced using a forward offset from the start of the referencing table, so a re-used paint table can only occur later in the file than any of the paint tables that use it.

The clock faces shown in Figure 5.45 provide an example of how PaintRotate tables can be combined with re-use of a sub-graph. As noted above, the hour hands have the same shape and fill, but have a different orientation. The glyph outline could point to the 12 position, then in color glyph descriptions for other times, PaintRotate tables could link to the same glyph/fill sub-graph, re-using that component but rotated as needed.

This is illustrated in Figures 5.46 and 5.47. Figure 5.46 shows a sub-graph defining the hour hand, with upright orientation, using a PaintGlyph and a PaintSolid table. Example file offsets for the tables are indicated.

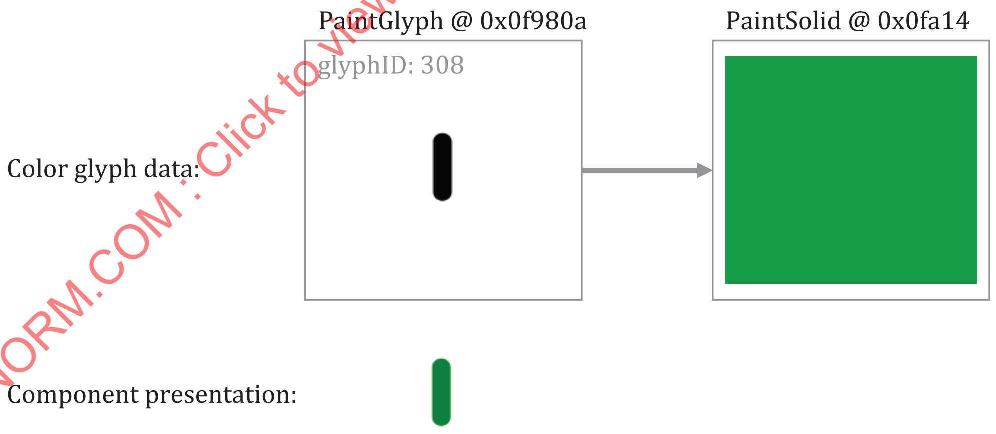


Figure 5.46 — A PaintGlyph and PaintSolid table are used to define the clock hour hand pointing to 12.

Figure 5.47 shows this sub-graph of paint tables being re-used, in some cases linked from PaintRotate tables that rotate the hour hand to point to different clock positions as needed. All of the paint tables that reference this sub-graph occur earlier in the file.

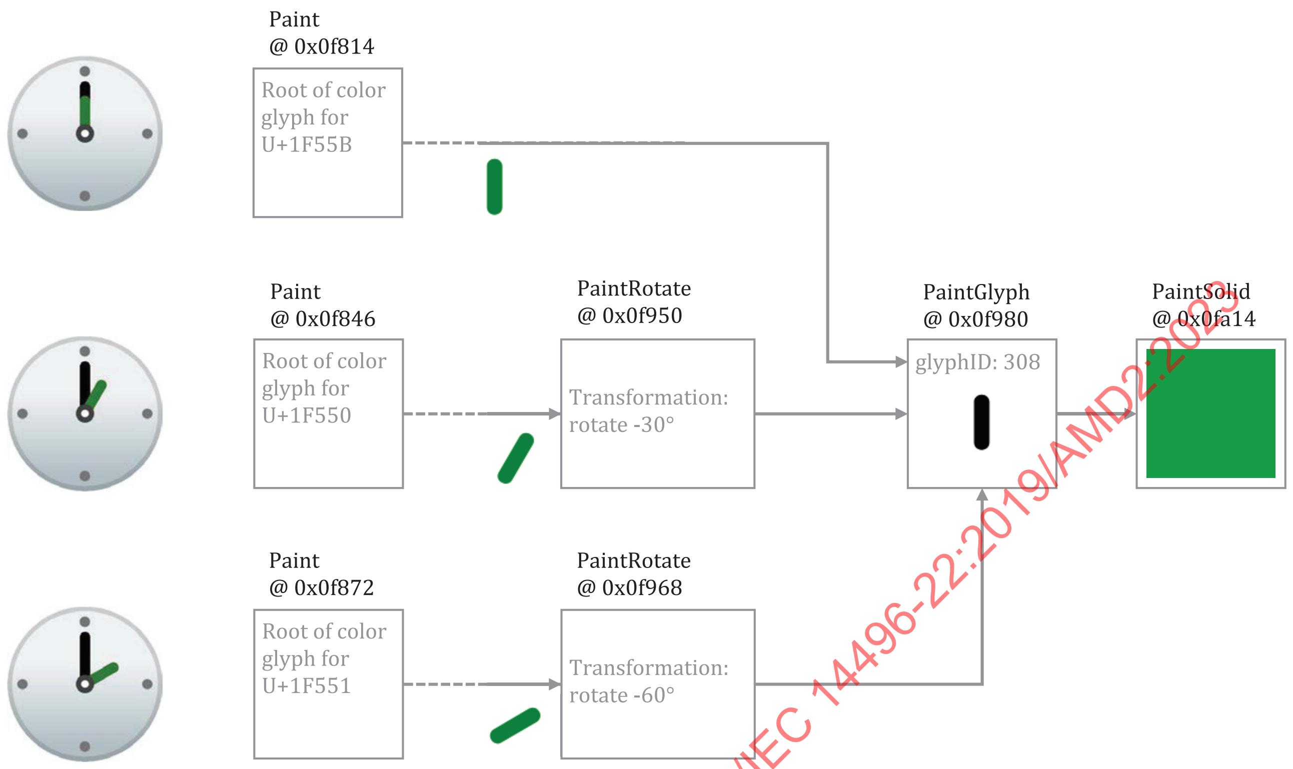

Paint
@ 0x0f814
Root of color glyph for U+1F55B
Paint
@ 0x0f846
Root of color glyph for U+1F550
PaintRotate
@ 0x0f950
Transformation: rotate -30°
PaintGlyph
@ 0x0f980
glyphID: 308
PaintSolid
@ 0x0fa14
Paint
@ 0x0f872
Root of color glyph for U+1F551
PaintRotate
@ 0x0f968
Transformation: rotate -60°

Figure 5.47 — The sub-graph for the hour hand is re-used with PaintRotate tables to point to different hours.

5.7.11.1.7.3 Re-use using PaintColrLayers

As described above (see 5.7.11.1.4), a PaintColrLayers table defines a set of paint sub-graphs arranged in bottom-up z-order layers, and an example was given of a PaintColrLayers table used as the root of a color glyph definition. A PaintColrLayers table can also be nested more deeply within the graph of a color glyph. One purpose for doing this is to reference a re-usable component defined as a contiguous set of layers in the LayerList table.

This is readily explained using the clock faces as an example. As described above, each clock face shares several elements in common. Some of these form a contiguous set of layers. Suppose four sub-graphs for shared clock face elements are given in the LayerList as contiguous layers, as shown in Figure 5.48. (For brevity, the visual result for each sub-graph is shown, but not the paint details.)

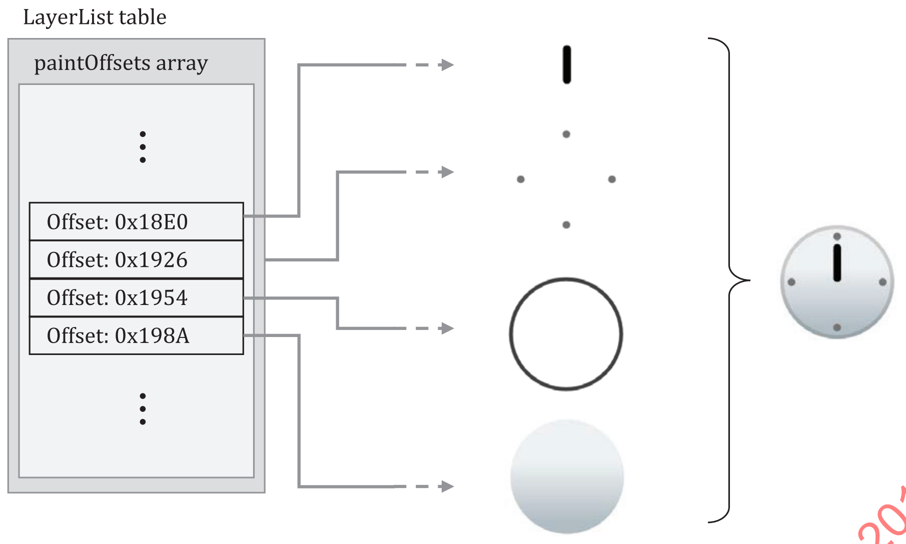


Figure 5.48 — Common clock face elements given as a slice within the LayerList table.

A PaintColrLayers table can reference any contiguous slice of layers in the LayerList table. Thus, the set of layers shown in Figure 5.48 can be referenced by PaintColrLayers tables anywhere in the graph of any color glyph. In this way, this set of layers can be re-used in multiple clock face color glyph definitions.

This is illustrated in Figure 5.49: The color glyph definition for the one o'clock emoji has a PaintColrLayers table as its root, referencing a slice of three layers in the LayerList table. The upper two layers are the hour hand, which is specific to this color glyph; and the cap over the pivot for the minute and hour hands, which is common to other clock emoji but in a layer that is not contiguous with other common layers. The bottom layer of these three layers is the composition for all the remaining common layers. It is represented using a nested PaintColrLayers table that references the slice within the LayerList for the common clock face elements shown in Figure 5.48.

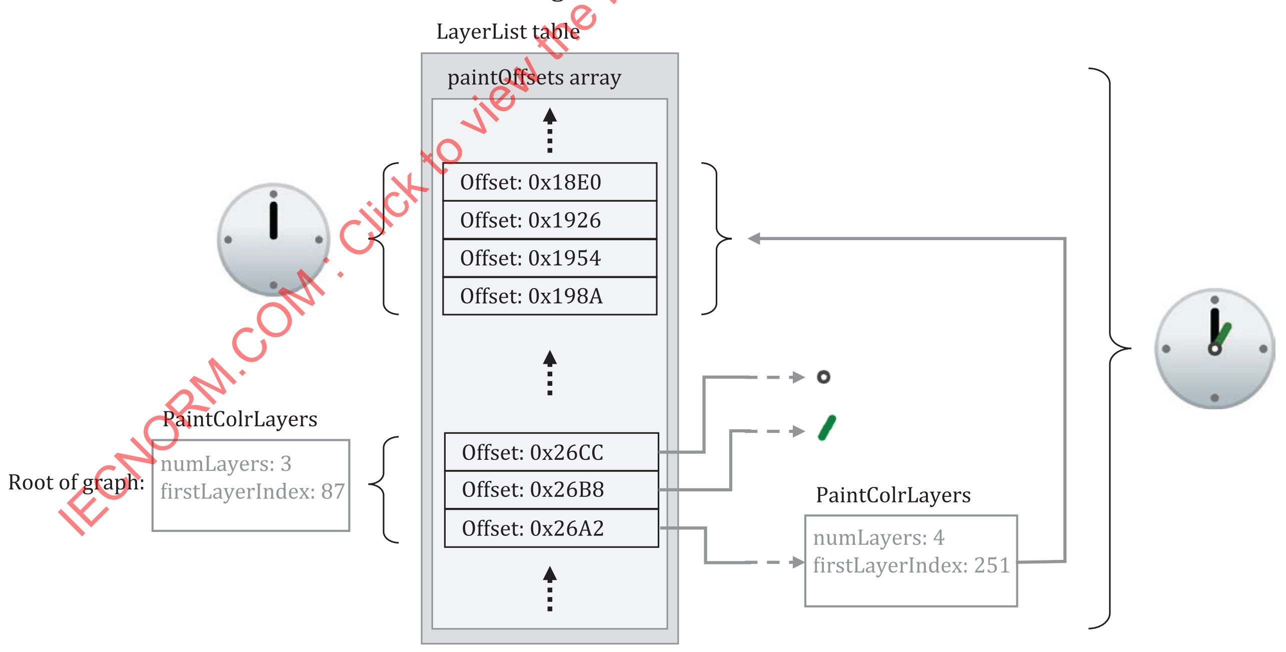


Figure 5.49 — A PaintColrLayers table is used to reference a set of layers that define a shared clock face composition.

The color glyphs for other clock face emoji could be structured in exactly the same way, using a nested PaintColrLayers table to re-use the layer composition of the common clock face elements.

5.7.11.1.7.4 Re-use using PaintColrGlyph

A third way to re-use components in color glyph definitions is to use a nested PaintColrGlyph table. This format references a base glyph ID, which is used to access a corresponding BaseGlyphPaintRecord. That record will provide the offset of a paint table that is the root of a graph for a color glyph definition. That graph can potentially be used as an independent color glyph, but it can also define a shared composition that gets re-used in multiple color glyphs. Each time the shared composition is to be re-used, it is referenced by its base glyph ID using a PaintColrGlyph table. The graph of the referenced color glyph is thereby incorporated into the graph of the PaintColrGlyph table as its child sub-graph.

When a PaintColrGlyph table is used, a BaseGlyphPaintRecord with the specified glyph ID is expected. If no BaseGlyphPaintRecord with that glyph ID is found, the color glyph is not well formed. See 5.7.11.1.9 for details regarding well-formedness and validity of the graph.

The example from 5.7.11.1.7.3 is modified to illustrate use of a PaintColrGlyph table. In Figure 5.50, a PaintColrLayers table references a slice within the LayerList that defines the shared component. Now, however, this PaintColrLayers table is treated as the root of a color glyph definition for base glyph ID 63163. The color glyph for the one o'clock emoji is defined with three layers, as before, but now the bottom layer uses a PaintColrGlyph table that references the color glyph definition for glyph ID 63163.

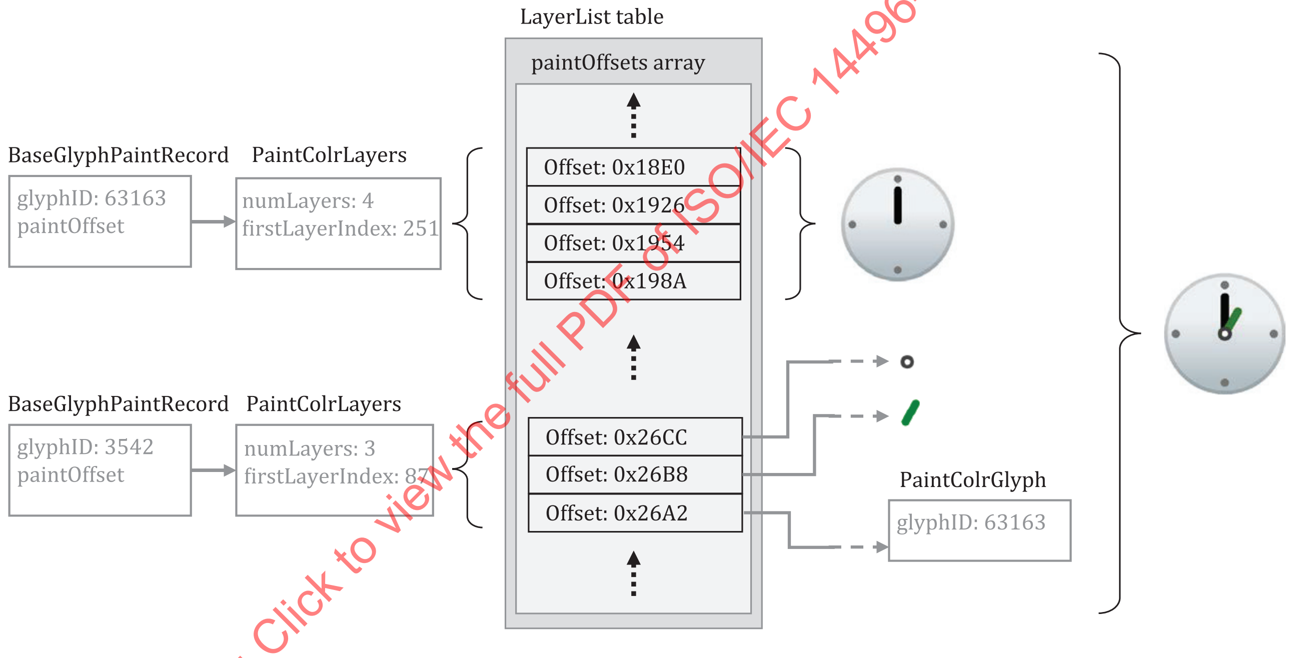


Figure 5.50 — A PaintColrGlyph table is used to reference the shared clock face composition via a glyph ID.

While the PaintColrGlyph and PaintColrLayers tables are similar in being able to reference a layer set as a re-usable component, they could be handled differently in implementations. In particular, an implementation could process and cache the result of the color glyph description for a given base glyph ID. In that case, subsequent references to that base glyph ID using a PaintColrGlyph table would not require the corresponding graph of paint tables to be re-processed. As a result, using a PaintColrGlyph for re-used graphic components could provide performance benefits.

5.7.11.1.8 Glyph metrics and boundedness

5.7.11.1.8.1 Metrics for color glyphs using version 0 formats

For color glyphs using version 0 formats, the advance width of glyphs used for each layer shall be the same as the advance width of the base glyph. If the font has vertical metrics, the glyphs used for each layer shall also have the same advance height and vertical Y origin as the base glyph.

5.7.11.1.8.2 Metrics and boundedness of color glyphs using version 1 formats

For color glyphs using version 1 formats, the advance width of the base glyph shall be used as the advance width for the color glyph. If the font has vertical metrics, the advance height and vertical Y origin of the base glyph shall be used for the color glyph. The advance width and height of glyphs referenced by PaintGlyph tables are not required to be the same as that of the base glyph and are ignored.

A valid color glyph definition shall define a bounded region—that is, it shall paint within a region for which a finite bounding box could be defined. A clip box can be specified to set overall bounds for a color glyph (see below). Otherwise, boundedness is determined by the graph of paint tables that describe the color glyph content. The different paint formats have different boundedness characteristics:

— PaintGlyph is inherently bounded.

— PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, and PaintVarSweepGradient are inherently unbounded.

— PaintColrLayers is bounded *if and only if* all referenced sub-graphs are bounded.

— PaintColrGlyph is bounded *if and only if* the color glyph definition for the referenced base glyph ID is bounded.

— Paint formats for transformations (PaintTransform, PaintVarTransform, PaintTranslate, PaintScale, etc.) are bounded *if and only if* the referenced sub-graph is bounded.

— PaintComposite is either bounded or unbounded, according to the composite mode used and the boundedness of the referenced sub-graphs. See 5.7.11.2.6.13 for details.

A ClipBox table (5.7.11.2.4) may be associated with a color glyph to define overall bounds for the color glyph. The clip box may vary in a variable font. If a clip box is provided for a color glyph, the color glyph is bounded, and no inspection of the Paint graph is required to determine boundedness. If no clip box is defined for a color glyph, however, applications shall confirm that the color glyph definition is bounded, and shall not render the color glyph if the defining graph is not bounded.

NOTE 1 If present, the clip box for a color glyph can be used to allocate a drawing surface without needing to traverse the graph of the color glyph definition.

NOTE 2 If no ClipBox table is present but a bounding box is required by the implementation, it can be computed for a given color glyph by traversing the graph of Paint tables that defines that color glyph.

To ensure that rendering implementations do not clip any part of a color glyph, the clip box needs to be large enough to encompass the entire color glyph composition. In a variable font, glyph outlines can vary, but transformations in a color glyph description can also vary, affecting the portions of the design grid to be painted. For example, a filled rectangle that is wide but not tall for one variation instance can be variably rotated to be tall but not wide for other instances. The clip box either should be large enough to encompass the color glyph for all instances, or should itself vary such that each instance of the clip box encompasses the instance color glyph.

5.7.11.1.9 Color glyphs as a directed acyclic graph

When using version 1 formats, a color glyph is defined by a directed, acyclic graph of linked paint tables. For each BaseGlyphPaintRecord, the paint table referenced by that record is the root of a graph defining a color glyph composition.

The graph for a given color glyph is made up of all paint tables reachable from the BaseGlyphPaintRecord. The BaseGlyphPaintRecord and several paint table formats use direct links; that is, they include a forward offset to a paint subtable. Two paint formats make indirect links:

— A PaintColrLayers table references a slice of offsets within the LayerList. The paint tables referenced by those offsets are considered to be linked within the graph as children of the PaintColrLayers table.

— A PaintColrGlyph table references a base glyph ID, for which a corresponding BaseGlyphPaintRecord is expected. That record points to the root of a graph that is a complete color glyph definition on its own. But when referenced in this way by a PaintColrGlyph table, that entire graph is considered to be a child sub-graph of the PaintColrGlyph table, and a continuation of the graph of which the PaintColrGlyph table is a part.

The graph for a color glyph is a combination of paint tables using any of the paint table formats. The simplest color glyph definition would consist of a PaintGlyph table linked to a basic fill table (PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, PaintVarSweepGradient). But the graph can be arbitrarily complex, with an arbitrary depth of paint nodes (to the limits inherent in the formats).

The graph can define a visual element in a single layer, or many elements in many layers. The concept of layers, as distinct visual elements stacked in a z-order, is not precisely defined in relation to the complexity of the graph. Each separate visual element requires a leaf node, but nodes in the graph, including leaf nodes, can be re-used (see 5.7.11.1.7). Also, each separate visual element requires a fork in the graph, and a separate root-to-leaf path, but not all paths necessarily result in a distinct visual element. For example, a gradient mask effect can be created with a gradient with gradation of alpha values, and then using that as the source of a PaintComposite table with the *Source In* compositing mode. In that case, the leaf has a visual affect but does not result in a distinct visual element. This was illustrated in Figure 5.44, repeated here as Figure 5.51: the PaintLinearGradient is a leaf node in the graph and creates a masking effect but does not add a distinct visual element.

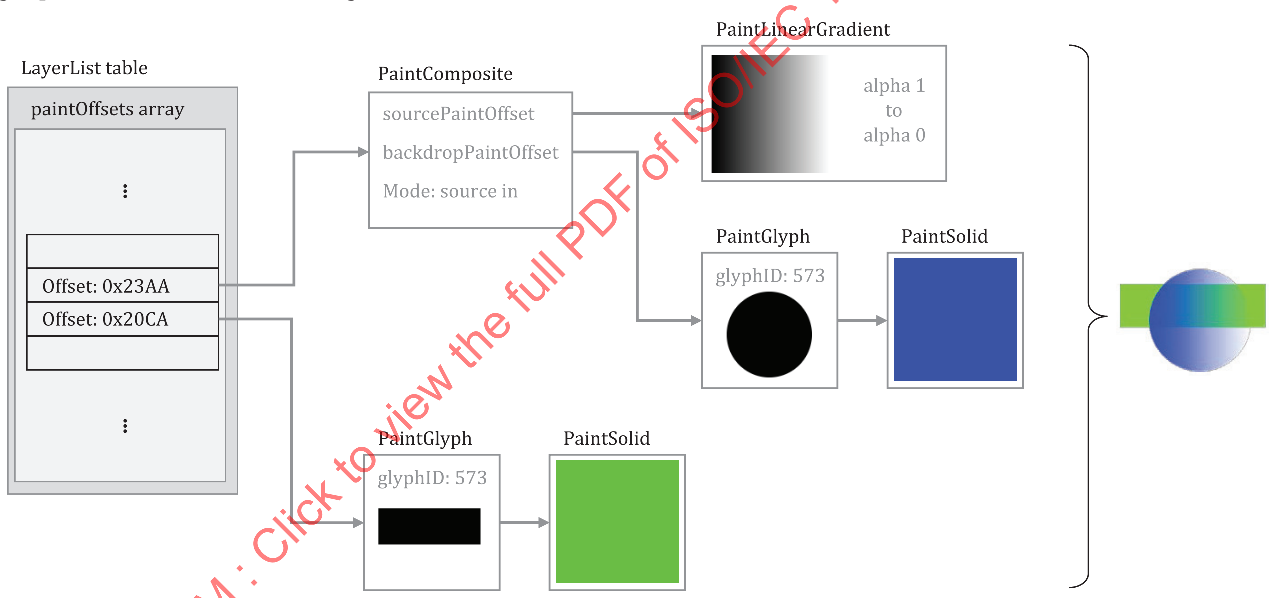


Figure 5.51 — Graph with a leaf node that isn't a distinct visual element.

Thus, the generalization that can be made regarding the relationship between the number of layers and the nature of the graph is that the number of distinct root-to-leaf paths will be greater than or equal to the number of layers.

The following are necessary for the graph to be well-formed and valid:

— All subtable links shall satisfy the following criteria:

  — Forward offsets are within the COLR table bounds.

  — If a PaintColrLayers table is present, then a LayerList is also present, and the referenced slice is within the length of the LayerList.

  — If a PaintColrGlyph table is present, there is a BaseGlyphPaintRecord for the referenced base glyph ID.

— The graph shall be acyclic.

NOTE 1 These constraints imply that all leaf nodes will be one of PaintSolid, PaintVarSolid, PaintLinearGradient, PaintVarLinearGradient, PaintRadialGradient, PaintVarRadialGradient, PaintSweepGradient, or PaintVarSweepGradient.

For the graph to be acyclic, no paint table shall have any child or descendent paint table that is also its parent or ancestor within the graph. In particular, because the PaintColrLayers and PaintColrGlyph tables use indirect child references rather than forward offsets, they provide a possibility for introducing cycles. Applications should track paint tables within a path in the graph, checking whether any paint table was already encountered within that path. The following pseudo-code algorithm can be used:

```
// called initially with the root paint and an empty set pathPaints
 function paintIsAcyclic(paint, pathPaints)
     if paint is in pathPaints
       return false // cycle detected
     add paint to pathPaints
     for each childPaint referenced by paint as a child subtable
       call paintIsAcyclic(childPaint, pathPaints)
     remove paint from pathPaints
```

For the graph to be valid, it shall also be visually bounded, as described in 5.7.11.1.8.2.

NOTE 2 Implementations can combine testing for cycles and other well-formedness or validity requirements together with other processing for rendering the color glyph.

If the graph contains a cycle or is otherwise not well formed or valid, the paint table at which the error occurs should be ignored, that sub-graph should not be rendered, and that node in the graph should be considered to be visually bounded. The application should attempt to render the remainder of the graph, if well-formed and valid.

Future minor version updates of the COLR table could introduce new paint formats. If a paint table with an unrecognized format is encountered, it and its sub-graph should similarly be ignored, the node should be considered to be visually bounded, and the application should attempt to render the remainder of the graph.

If an application is not able to recover from errors while traversing the graph, it may ignore the color glyph entirely. If the base glyph ID has an outline, that may be rendered as a non-color glyph instead.

5.7.11.2 COLR table formats

5.7.11.2.1 Overview

Various table and record formats are defined for COLR version 0 and version 1. Several values contained within the version 1 formats are variable.

For items that vary in a variable font, the variation data is contained in an ItemVariationStore table (7.2.3). To associate each variable item with the corresponding variation data, a DeltaSetIndexMap table (7.2.3.1) is used. Within a given table that has variable items, a base/sequence scheme is used to index into the mapping data. See 5.7.11.4 for details.

Future minor version updates of the COLR table could introduce new formats that extend the capabilities for color glyph descriptions using version 1 formats. Unrecognized formats should be ignored. See 5.7.11.1.9 for more information.

All table offsets are from the start of the parent table in which the offset is given, unless otherwise indicated.

The COLR table begins with a header. Two versions have been defined. Offsets in the header are from the start of the table.

5.7.11.2.2 COLR header

5.7.11.2.2.1 COLR version 0

COLR version 0:

Type	Name	Description
uint16	version	Table version number—set to 0.
uint16	numBaseGlyphRecords	Number of BaseGlyph records.
Offset32	baseGlyphRecordsOffset	Offset to baseGlyphRecords array.
Offset32	layerRecordsOffset	Offset to layerRecords array.
uint16	numLayerRecords	Number of Layer records.

NOTE For fonts that use COLR version 0, some early implementations of the COLR table require glyph ID 1 to be the .null glyph.

5.7.11.2.2.2 COLR version 1

COLR version 1:

Type	Name	Description
uint16	version	Table version number—set to 1.
uint16	numBaseGlyphRecords	Number of BaseGlyph records; may be 0 in a version 1 table.
Offset32	baseGlyphRecordsOffset	Offset to baseGlyphRecords array (may be NULL).
Offset32	layerRecordsOffset	Offset to layerRecords array (may be NULL).
uint16	numLayerRecords	Number of Layer records; may be 0 in a version 1 table.
Offset32	baseGlyphListOffset	Offset to BaseGlyphList table.
Offset32	layerListOffset	Offset to LayerList table (may be NULL).
Offset32	clipListOffset	Offset to ClipList table (may be NULL).
Offset32	varIndexMapOffset	Offset to DeltaSetIndexMap table (may be NULL).
Offset32	itemVariationStoreOffset	Offset to ItemVariationStore (may be NULL).

The BaseGlyphList and its subtables are only used in COLR version 1.

The LayerList is only used in conjunction with the BaseGlyphList and, specifically, with PaintColrLayers tables (5.7.11.2.6.1); it is not required if no color glyphs use a PaintColrLayers table. If not used, set layerListOffset to NULL.

The ClipList is only used in conjunction with the BaseGlyphList. If not used, set clipListOffset to NULL.

The ItemVariationStore (7.2.3) is used in conjunction with a BaseGlyphList and its subtables, but only in variable fonts. If it is not used, set itemVariationStoreOffset to NULL.

The DeltaSetIndexMap table is described in 7.2.3.1. Within the COLR table, either format 0 or format 1 of the DeltaSetIndexMap can be used. A DeltaSetIndexMap is used in conjunction with the ItemVariationStore in a variable font. The DeltaSetIndexMap is optional: if an ItemVariationStore is present but a DeltaSetIndexMap is not included (varIndexMapOffset is NULL), then an implicit mapping is used. See 5.7.11.4 for details.

5.7.11.2.2.3 Mixing version 0 and version 1 formats

A font that uses COLR version 1 and that includes a BaseGlyphList can also include BaseGlyph and Layer records for compatibility with applications that only support COLR version 0.

Color glyphs that can be implemented in COLR version 0 using BaseGlyph and Layer records can also be implemented using the version 1 BaseGlyphList and subtables. Thus, a font that uses the version 1 formats does not need to use the version 0 BaseGlyph and Layer records. However, a font may use the version 1 structures for some base glyphs and the version 0 structures for other base glyphs. A font may also include a version 1 color glyph definition for a given base glyph ID that is equivalent to a version 0 definition, though this should never be needed.

A font may define a color glyph for a given base glyph ID using version 0 formats, and also define a different color glyph for the same base glyph ID using version 1 formats. Applications that support COLR version 1 should give preference to the version 1 color glyph.

For applications that support COLR version 1, the application should search for a base glyph ID first in the BaseGlyphList. Then, if not found, search in the baseGlyphRecords array, if present.

5.7.11.2.3 BaseGlyph and Layer records

BaseGlyph and Layer records are required for COLR version 0, but optional for version 1 (see 5.7.11.2.2.3).

A BaseGlyph record is used to map a base glyph to a sequence of layer records that define the corresponding color glyph. The BaseGlyph record includes a base glyph index, an index into the layerRecords array, and the number of layers.

BaseGlyph record:

Type	Name	Description
uint16	glyphID	Glyph ID of the base glyph.
uint16	firstLayerIndex	Index (base 0) into the layerRecords array.
uint16	numLayers	Number of color layers associated with this glyph.

The glyph ID shall be less than the numGlyphs value in the 'maxp' table (5.2.6).

The BaseGlyph records shall be sorted in increasing glyphID order. It is assumed that a binary search can be used to find a matching BaseGlyph record for a specific glyphID.

The color glyph for a given base glyph is defined by the consecutive records in the layerRecords array for the specified number of layers, starting with the record indicated by firstLayerIndex. The first record in this sequence is the bottom layer in the z-order, and each subsequent layer is stacked on top of the previous layer.

The layer record sequences for two different base glyphs may overlap, with some layer records used in multiple color glyph definitions.

The Layer record specifies the glyph used as the graphic element for a layer and the solid color fill.

Layer record:

Type	Name	Description
uint16	glyphID	Glyph ID of the glyph used for a given layer.
uint16	paletteIndex	Index (base 0) for a palette entry in the CPAL table.

The glyphID in a Layer record shall be less than the numGlyphs value in the 'maxp' table. That is, it shall be a valid glyph with outline data in the 'glyf' (5.3.4), 'CFF' (5.4.2) or CFF2 (5.4.3) table. See 5.7.11.1.8.2 for requirements regarding glyph metrics of referenced glyphs.

The paletteIndex value shall be less than the numPaletteEntries value in the CPAL table (5.7.12). A paletteIndex value of 0xFFFF is a special case, indicating that the text foreground color (as determined by the application) is to be used.

5.7.11.2.4 BaseGlyphList, LayerList and ClipList

The BaseGlyphList table is, conceptually, similar to the baseGlyphRecords array in COLR version 0, providing records that map a base glyph to a color glyph definition. The color glyph definitions that each refer to are significantly different, however—see 5.7.11.1.

BaseGlyphList table:

Type	Name	Description
uint32	numBaseGlyphPaintRecords	
BaseGlyphPaintRecord	baseGlyphPaintRecords[numBaseGlyphPaintRecords]	

BaseGlyphPaintRecord:

Type	Name	Description
uint16	glyphID	Glyph ID of the base glyph.
Offset32	paintOffset	Offset to a Paint table.

The glyphID value shall be less than the numGlyphs value in the 'maxp' table (5.2.6).

The records in the baseGlyphPaintRecords array shall be sorted in increasing glyphID order. It is intended that a binary search can be used to find a matching BaseGlyphPaintRecord for a specific glyphID.

The paint table referenced by the BaseGlyphPaintRecord is the root of the graph for a color glyph definition.

NOTE 1 Often the paint table that is the root of the graph for the color glyph definition will be a PaintColrLayers table, though this is not required. See 5.7.11.1.9 for more information regarding the graph of a color glyph, and 5.7.11.1.4 for background information regarding the PaintColrLayers table.

A LayerList table is used in conjunction with PaintColrLayers tables to represent layer structures. A single LayerList is defined and can be used by multiple PaintColrLayers tables, each of which references a slice of the layer list.

LayerList table:

Type	Name	Description
uint32	numLayers	
Offset32	paintOffsets[numLayers]	Offsets to Paint tables.

The sequence of offsets to paint tables corresponds to a bottom-up z-order layering of the graphic compositions defined by the sub-graph of each referenced paint table graph. For a given slice of the list, the sub-graph of the first paint table defines the element at the bottom of the z-order, and the sub-graph of each subsequent paint table defines an element that is layered on top of the previous element. As each element is a composition defined in a sub-graph, one of these elements may itself be multi-layered. In that case, the layers of this element are stacked above all previous layers, and layers of following elements are stacked above the top layer of this element.

Offsets for paint tables not referenced by any PaintColrLayers table should not be included in the paintOffsets array.

A ClipList table is used to provide precomputed clip boxes for color glyphs. It contains an array of Clip records, each of which associates a range of base glyph IDs with a ClipBox table. The ClipBox table provides a precomputed clip box for the associated color glyphs. Clip boxes are optional: a font may provide clip boxes for some color glyphs but not others.

ClipList table:

Type	Name	Description
uint8	format	Set to 1.
uint32	numClips	Number of Clip records.
Clip	clips[numClips]	Clip records. Sorted by startGlyphID.

Clip record:

Type	Name	Description
uint16	startGlyphID	First glyph ID in the range.
uint16	endGlyphID	Last glyph ID in the range.
Offset32	clipBoxOffset	Offset to a ClipBox table.

Within a ClipList table, the glyph ID ranges of Clip records shall not overlap.

Two Clipbox table formats are defined: format 1 for clip boxes without variation, and format 2 allowing for clip boxes that can vary in a variable font.

ClipBoxFormat1 table, static clip box:

Type	Name	Description
uint8	format	Set to 1.
FWORD	xMin	Minimum x of clip box.
FWORD	yMin	Minimum y of clip box.
FWORD	xMax	Maximum x of clip box.
FWORD	yMax	Maximum y of clip box.

ClipBoxFormat2 table, variable clip box:

Type	Name	Description
uint8	format	Set to 2.
FWORD	xMin	Minimum x of clip box. For variation, use varIndexBase + 0.
FWORD	yMin	Minimum y of clip box. For variation, use varIndexBase + 1.
FWORD	xMax	Maximum x of clip box. For variation, use varIndexBase + 2.
FWORD	yMax	Maximum y of clip box. For variation, use varIndexBase + 3.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

Any content drawn outside the clip box shall not render.

The clip box is not required to be a tight bounding box around the content. As it may be used by implementations to allocate resources, however, it should not be unnecessarily large.

NOTE 2 At runtime, when computing a variable ClipBox, compute the min/max coordinates using floating point values and then round to integer values such that the clip box expands. That is, round xMin and yMin towards negative infinity and round xMax and yMax towards positive infinity.

For variable data, a base/sequence scheme is used to index into variation mapping data. See 5.7.11.4 for details.

5.7.11.2.5 Color references, ColorStop and ColorLine

Colors are used in solid color fills for graphic elements, or as *stops* in a color line used to define a gradient. Colors are defined by reference to palette entries in the CPAL table (5.7.12). While CPAL entries include an alpha component, formats for COLR version 1 that reference palette entries also includes a separate alpha specification to allow different graphic elements to use the same color but with different alpha values, and to allow for variation of the alpha in variable fonts.

A paletteIndex value of 0xFFFF is a special case, indicating that the text foreground color (as determined by the application) is to be used.

The alpha value is always set explicitly. Values for alpha outside the range [0., 1.] (inclusive) are reserved; values outside this range shall be clipped. A value of zero means no opacity (fully transparent); 1.0 means fully opaque (no transparency). The alpha indicated in this record is multiplied with the alpha component of the CPAL entry (converted to float—divide by 255). Note that the resulting alpha value can be combined with and does not supersede alpha or opacity attributes set in higher-level, application-defined contexts.

See 5.7.11.1.1 for more information regarding color references and solid color fills. Solid color fills are defined using a PaintSolid or PaintVarSolid table, described below—see 5.7.11.2.6.2.

Gradients are defined using a color line. A color line is a mapping of real numbers to color values, defined using color stops. See 5.7.11.1.2.2 for an overview and additional details.

Two color-stop record formats are defined: one that allows for variation of stop offset position or of alpha, and one that does not. The format supporting variations uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

ColorStop record:

Type	Name	Description
F2DOT14	stopOffset	Position on a color line.
uint16	paletteIndex	Index for a CPAL palette entry.
F2DOT14	alpha	Alpha value.

VarColorStop record:

Type	Name	Description
F2DOT14	stopOffset	Position on a color line. For variation, use varIndex-Base + 0.
uint16	paletteIndex	Index for a CPAL palette entry.
F2DOT14	alpha	Alpha value. For variation, use varIndexBase + 1.
uint32	varIndexBase	Base index into DeltaSetIn-dexMap.

A color line is defined by an array of color stop records plus an extend mode. Two color-line table formats are defined: one that allows for variation of color stop offsets positions or of alpha values, and one that does not. Different paint table formats for gradients use one or the other of the color line formats.

ColorLine table:

Type	Name	Description
uint8	extend	An Extend enum value.
uint16	numStops	Number of ColorStop records.
ColorStop	colorStops[numStops]	

VarColorLine table:

Type	Name	Description
uint8	extend	An Extend enum value.
uint16	numStops	Number of ColorStop records.
VarColorStop	colorStops[numStops]	Allows for variations.

Applications shall apply the colorStops entries in increasing stopOffset order. Within a variable font, the stopOffset values can vary, and the relative orderings of color stop records along the color line can change as a result of variation. With a variable font, the colorStops entries shall be ordered after the instance values for the stop offsets have been derived.

A color line defines stops for only certain positions along the line, but the color line extends infinitely in either direction. The extend field is used to indicate how the color line is extended. The same behavior is used for extension in both directions. The extend field uses the following enumeration:

Extend enumeration:

Value	Name	Description
0	EXTEND_PAD	Use nearest color stop.
1	EXTEND_REPEAT	Repeat from farthest color stop.
2	EXTEND_REFLECT	Mirror color line from nearest end.

The extend mode behaviors are described in detail in 5.7.11.1.2.2. If a ColorLine in a font has an unrecognized extend value, applications should use EXTEND_PAD by default.

5.7.11.2.6 Paint tables

Paint tables are used for COLR version 1 color glyph definitions. Thirty-two paint table formats are defined (formats 1 to 32). Some formats come in non-variable and variable pairs, but otherwise, each provides different graphic capability for defining the composition for a color glyph. The graphic capability of each format and the manner in which they are combined to represent a color glyph has been described above—see 5.7.11.1.

Each paint table format has a one-byte format field as the first field. When parsing font data, the format field can be read first to determine the format of the table.

5.7.11.2.6.1 Format 1: PaintColrLayers

Format 1 is used to define a vector of layers. The layers are a slice of layers from the LayerList table. The first layer is the bottom of the z-order, and subsequent layers are composited on top using the COMPOSITE_SRC_OVER composition mode (see 5.7.11.2.6.13).

For general information on the PaintColrLayers table, see 5.7.11.1.4. For information about its use for shared, re-usable components, see 5.7.11.1.7.3.

PaintColrLayers table (format 1):

Type	Name	Description
uint8	format	Set to 1.
uint8	numLayers	Number of offsets to paint tables to read from LayerList.
uint32	firstLayerIndex	Index (base 0) into the LayerList.

NOTE An 8-bit value is used for numLayers to minimize size for common scenarios. If more than 256 layers are needed, then two or more PaintColrLayers tables can be combined in a tree using a PaintComposite table or another PaintColrLayers table to combine them.

5.7.11.2.6.2 Formats 2 and 3: PaintSolid, PaintVarSolid

Formats 2 and 3 are used to specify a solid color fill. Format 3 allows for variation of alpha in a variable font; format 2 provides a more compact representation when variation is not required. Format 3 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

For general information about specifying color values, see 5.7.11.1.1. For information about applying a fill to a shape, see 5.7.11.1.3.

PaintSolid table (format 2):

Type	Name	Description
uint8	format	Set to 2.
uint16	paletteIndex	Index for a CPAL palette entry.
F2DOT14	alpha	Alpha value.

PaintVarSolid table (format 3):

Type	Name	Description
uint8	format	Set to 3.
uint16	paletteIndex	Index for a CPAL palette entry.
F2DOT14	alpha	Alpha value. For variation, use varIndexBase + 0.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

5.7.11.2.6.3 Formats 4 and 5: PaintLinearGradient, PaintVarLinearGradient

Formats 4 and 5 are used to specify a linear gradient fill. Format 4 allows for variation of color stop positions or of alpha in a variable font; format 5 provides a more compact representation when variation is not required. Format 5 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

For general information about linear gradients, see 5.7.11.1.2.3. For information about applying a fill to a shape, see 5.7.11.1.3.

The PaintLinearGradient and PaintVarLinearGradient tables have a ColorLine and VarColorLine subtable, respectively. For the ColorLine and VarColorLine table formats, see 5.7.11.2.5. For background information on the color line, see 5.7.11.1.2.2.

PaintLinearGradient table (format 4):

Type	Name	Description
uint8	format	Set to 4.
Offset24	colorLineOffset	Offset to ColorLine table.
FWORD	x0	Start point (p_0) x coordinate.
FWORD	y0	Start point (p_0) y coordinate.
FWORD	x1	End point (p_1) x coordinate.
FWORD	y1	End point (p_1) y coordinate.
FWORD	x2	Rotation point (p_2) x coordinate.
FWORD	y2	Rotation point (p_2) y coordinate.

PaintVarLinearGradient table (format 5):

Type	Name	Description
uint8	format	Set to 5.

Type	Name	Description
Offset24	colorLineOffset	Offset to VarColorLine table.
FWORD	x0	Start point (p_0) x coordinate. For variation, use varIndexBase + 0.
FWORD	y0	Start point (p_0) y coordinate. For variation, use varIndexBase + 1.
FWORD	x1	End point (p_1) x coordinate. For variation, use varIndexBase + 2.
FWORD	y1	End point (p_1) y coordinate. For variation, use varIndexBase + 3.
FWORD	x2	Rotation point (p_2) x coordinate. For variation, use varIndexBase + 4.
FWORD	y2	Rotation point (p_2) y coordinate. For variation, use varIndexBase + 5.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarLinearGradient format uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

5.7.11.2.6.4 Formats 6 and 7: PaintRadialGradient, PaintVarRadialGradient

Formats 6 and 7 are used to specify a radial gradient fill. Format 7 allows for variation of color stop positions or of alpha in a variable font; format 6 provides a more compact representation when variation is not required. Format 7 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

For general information about radial gradients supported in COLR version 1, see 5.7.11.1.2.4. For information about applying a fill to a shape, see 5.7.11.1.3.

The PaintRadialGradient and PaintVarRadialGradient tables have a ColorLine and VarColorLine subtable, respectively. For the ColorLine and VarColorLine table formats, see in 5.7.11.2.5. For background information on the color line, see 5.7.11.1.2.2.

PaintRadialGradient table (format 6):

Type	Name	Description
uint8	format	Set to 6.
Offset24	colorLineOffset	Offset to ColorLine table.
FWORD	x0	Start circle center x coordinate.
FWORD	y0	Start circle center y coordinate.
UFWORD	radius0	Start circle radius.
FWORD	x1	End circle center x coordinate.
FWORD	y1	End circle center y coordinate.
UFWORD	radius1	End circle radius.

PaintVarRadialGradient table (format 7):

Type	Name	Description
uint8	format	Set to 7.
Offset24	colorLineOffset	Offset to VarColorLine table.
FWORD	x0	Start circle center x coordinate. For variation, use varIndexBase + 0.
FWORD	y0	Start circle center y coordinate. For variation, use varIndexBase + 1.

Type	Name	Description
UFWORD	radius0	Start circle radius. For variation, use varIndexBase + 2.
FWORD	x1	End circle center x coordinate. For variation, use varIndexBase + 3.
FWORD	y1	End circle center y coordinate. For variation, use varIndexBase + 4.
UFWORD	radius1	End circle radius. For variation, use varIndexBase + 5.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarRadialGradient format uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

5.7.11.2.6.5 Formats 8 and 9: PaintSweepGradient, PaintVarSweepGradient

Formats 8 and 9 are used to specify a sweep gradient fill. Format 9 allows for variation of color stop positions or of alpha in a variable font; format 8 provides a more compact representation when variation is not required. Format 9 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

For general information about sweep gradients, see 5.7.11.1.2.5. For information about applying a fill to a shape, see 5.7.11.1.3.

The PaintSweepGradient and PaintVarSweepGradient table have a ColorLine and VarColorLine subtable, respectively. For the ColorLine and VarColorLine table formats, see 5.7.11.2.5. For background information on the color line, see 5.7.11.1.2.2.

PaintSweepGradient table (format 8):

Type	Name	Description
uint8	format	Set to 8.
Offset24	colorLineOffset	Offset to ColorLine table.
FWORD	centerX	Center x coordinate.
FWORD	centerY	Center y coordinate.
F2DOT14	startAngle	Start of the angular range of the gradient, 180° in counter-clockwise degrees per 1.0 of value.
F2DOT14	endAngle	End of the angular range of the gradient, 180° in counter-clockwise degrees per 1.0 of value.

PaintVarSweepGradient table (format 9):

Type	Name	Description
uint8	format	Set to 9.
Offset24	colorLineOffset	Offset to VarColorLine table.
FWORD	centerX	Center x coordinate. For variation, use varIndexBase + 0.
FWORD	centerY	Center y coordinate. For variation, use varIndexBase + 1.
F2DOT14	startAngle	Start of the angular range of the gradient, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 2.
F2DOT14	endAngle	End of the angular range of the gradient, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 3.

Type	Name	Description
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarSweepGradient format uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

Angles are expressed in counter-clockwise degrees from the direction of the positive x-axis in the design grid.

5.7.11.2.6.6 Format 10: PaintGlyph

Format 10 is used to specify a glyph outline to use as a shape to be filled or, equivalently, a clip region. The outline sets a clip region that constrains the content of a separate paint subtable and the sub-graph linked from that subtable.

For information about applying a fill to a shape, see 5.7.11.1.3.

PaintGlyph table (format 10):

Type	Name	Description
uint8	format	Set to 10.
Offset24	paintOffset	Offset to a Paint table.
uint16	glyphID	Glyph ID for the source outline.

The glyphID value shall be less than the numGlyphs value in the 'maxp' table (5.2.6). That is, it shall be a valid glyph with outline data in the 'glyf' (5.3.4), 'CFF' (5.4.2) or CFF2 (5.4.3) table. Only that outline data is used. In particular, if this glyph ID has a description in the COLR table (glyphID appears in a COLR BaseGlyph record or the BaseGlyphList), that COLR data is not relevant for purposes of the PaintGlyph table.

5.7.11.2.6.7 Format 11: PaintColrGlyph

Format 11 is used to allow a color glyph definition from the BaseGlyphList to be a re-usable component that can be incorporated into multiple color glyph definitions. See 5.7.11.1.7.4 for more information.

PaintColrGlyph table (format 11):

Type	Name	Description
uint8	format	Set to 11.
uint16	glyphID	Glyph ID for a BaseGlyphList base glyph.

The glyphID value shall be a glyphID found in a BaseGlyphPaintRecord within the BaseGlyphList. The BaseGlyphPaintRecord provides an offset to a paint table; that paint table and the graph linked from it are incorporated as a child sub-graph of the PaintColrGlyph table within the current color glyph definition.

5.7.11.2.6.8 Formats 12 and 13: PaintTransform, PaintVarTransform

Formats 12 and 13 are used to apply an affine transformation to a sub-graph. The paint table that is the root of the sub-graph is linked as a child.

Format 13 allows for variation of the transformation in a variable font; format 12 provides a more compact representation when variation is not required. Format 13 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

For general information regarding transformations in a color glyph definition, see 5.7.11.1.5.

PaintTransform table (format 12):

Type	Name	Description
uint8	format	Set to 12.
Offset24	paintOffset	Offset to a Paint subtable.
Offset24	transformOffset	Offset to an Affine2x3 table.

PaintVarTransform table (format 13):

Type	Name	Description
uint8	format	Set to 13.
Offset24	paintOffset	Offset to a Paint subtable.
Offset24	transformOffset	Offset to a VarAffine2x3 table.

The affine transformation is defined by a 2×3 matrix, specified in an Affine2x3 or VarAffine2x3 record. The 2×3 matrix supports scale, skew, reflection, rotation, and translation transformations. The VarAffine2x3 table supports mapping into variation data, allowing the transform definition to be variable in a variable font.

Affine2x3 table:

Type	Name	Description
Fixed	xx	x-component of transformed x-basis vector.
Fixed	yx	y-component of transformed x-basis vector.
Fixed	xy	x-component of transformed y-basis vector.
Fixed	yy	y-component of transformed y-basis vector.
Fixed	dx	Translation in x direction.
Fixed	dy	Translation in y direction.

VarAffine2x3 table:

Type	Name	Description
Fixed	xx	x-component of transformed x-basis vector. For variation, use varIndexBase + 0.
Fixed	yx	y-component of transformed x-basis vector. For variation, use varIndexBase + 1.
Fixed	xy	x-component of transformed y-basis vector. For variation, use varIndexBase + 2.
Fixed	yy	y-component of transformed y-basis vector. For variation, use varIndexBase + 3.
Fixed	dx	Translation in x direction. For variation, use varIndexBase + 4.
Fixed	dy	Translation in y direction. For variation, use varIndexBase + 5.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The VarAffine2x3 format uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

For a pre-transformation position *(x, y)*, the post-transformation position *(x′, y′)* is calculated as follows:

x′ = xx * x + xy * y + dx
y′ = yx * x + yy * y + dy

NOTE It is helpful to understand linear transformations by their effect on *x-* and *y-basis* vectors $\hat{\imath}$ = (1, 0) and $\hat{\jmath}$ = (0, 1). The transform described by the Affine2x3 or VarAffine2x3 table maps the basis vectors to $\hat{\imath}'$ = (*xx*, *yx*) and $\hat{\jmath}'$ = (*xy*, *yy*), and translates the origin to (*dx*, *dy*).

When the transformed composition from the referenced paint table (and its sub-graph) is composed into the destination (represented by the parent of this table), the source design grid origin is aligned to the destination design grid origin. The transform can translate the source such that a pre-transform position (0,0) is moved elsewhere. The *post-transform* origin, (0,0), is aligned to the destination origin.

5.7.11.2.6.9 Formats 14 and 15: PaintTranslate, PaintVarTranslate

Formats 14 and 15 are used to apply a translation to a sub-graph. The paint table that is the root of the sub-graph is linked as a child.

Format 15 allows for variation of the translation in a variable font; format 14 provides a more compact representation when variation is not required. Format 15 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

These tables use reduced precision for compactness. Where higher precision is required use PaintTransform/PaintVarTransform.

For general information regarding transformations in a color glyph definition, see 5.7.11.1.5.

PaintTranslate table (format 14):

Type	Name	Description
uint8	format	Set to 14.
Offset24	paintOffset	Offset to a Paint subtable.
FWORD	dx	Translation in x direction.
FWORD	dy	Translation in y direction.

PaintVarTranslate table (format 15):

Type	Name	Description
uint8	format	Set to 15.
Offset24	paintOffset	Offset to a Paint subtable.
FWORD	dx	Translation in x direction. For variation, use varIndexBase + 0.
FWORD	dy	Translation in y direction. For variation, use varIndexBase + 1.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarTranslate format uses a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

NOTE Pure translation can also be represented using the PaintTransform or PaintVarTransform table by setting *xx* = 1, *yy* = 1, *xy* and *yx* = 0, and setting *dx* and *dy* to the translation values. The PaintTranslate or PaintVarTranslate table provides a more compact representation when only translation is required.

The translation will result in the pre-transform position (0,0) being moved elsewhere. See 5.7.11.2.6.8 regarding alignment of the transformed content with the destination.

5.7.11.2.6.10 Formats 16 to 23: PaintScale and variant scaling formats

Formats 16 to 23 are used to scale a sub-graph. The paint table that is the root of the sub-graph is linked as a child. Several variant formats are provided:

— Formats 16 and 17: scale in x or y directions relative to the origin. Format 17 allows for variation of the x and y scale factors in a variable font; format 16 provides a more compact representation when variation is not required.

— Formats 18 and 19: scale in x or y directions relative to a specified center. Format 19 allows for variation of the x and y scale factors or of the center position; format 18 provides a more compact representation when variation is not required.

— Formats 20 and 21: scale uniformly in x and y directions relative to the origin. Format 21 allows for variation of the scale factor in a variable font; format 20 provides a more compact representation when variation is not required.

— Formats 22 and 23: scale uniformly in x and y directions relative to a specified center. Format 23 allows for variation of the scale factor or of the center position; format 22 provides a more compact representation when variation is not required.

Formats 17, 19, 21 and 23 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

These tables use reduced precision for compactness. Where higher precision is required use PaintTransform/PaintVarTransform.

For general information regarding transformations in a color glyph definition, see 5.7.11.1.5.

PaintScale table (format 16):

Type	**Name**	**Description**
uint8	format	Set to 16.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scaleX	Scale factor in x direction.
F2DOT14	scaleY	Scale factor in y direction.

PaintVarScale table (format 17):

Type	**Name**	**Description**
uint8	format	Set to 17.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scaleX	Scale factor in x direction. For variation, use varIndexBase + 0.
F2DOT14	scaleY	Scale factor in y direction. For variation, use varIndexBase + 1.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

PaintScaleAroundCenter table (format 18):

Type	**Name**	**Description**
uint8	format	Set to 18.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scaleX	Scale factor in x direction.
F2DOT14	scaleY	Scale factor in y direction.
FWORD	centerX	x coordinate for the center of scaling.
FWORD	centerY	y coordinate for the center of scaling.

PaintVarScaleAroundCenter table (format 19):

Type	**Name**	**Description**
uint8	format	Set to 19.
Offset24	paintOffset	Offset to a Paint subtable.

Type	Name	Description
F2DOT14	scaleX	Scale factor in x direction. For variation, use varIndexBase + 0.
F2DOT14	scaleY	Scale factor in y direction. For variation, use varIndexBase + 1.
FWORD	centerX	x coordinate for the center of scaling. For variation, use varIndexBase + 2.
FWORD	centerY	y coordinate for the center of scaling. For variation, use varIndexBase + 3.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

PaintScaleUniform table (format 20):

Type	Name	Description
uint8	format	Set to 20.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scale	Scale factor in x and y directions.

PaintVarScaleUniform table (format 21):

Type	Name	Description
uint8	format	Set to 21.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scaleX	Scale factor in x and y directions. For variation, use varIndexBase + 0.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

PaintScaleUniformAroundCenter table (format 22):

Type	Name	Description
uint8	format	Set to 22.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scale	Scale factor in x and y directions.
FWORD	centerX	x coordinate for the center of scaling.
FWORD	centerY	y coordinate for the center of scaling.

PaintVarScaleUniformAroundCenter table (format 23):

Type	Name	Description
uint8	format	Set to 23.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	scale	Scale factor in x and y directions. For variation, use varIndexBase + 0.
FWORD	centerX	x coordinate for the center of scaling. For variation, use varIndexBase + 1.
FWORD	centerY	y coordinate for the center of scaling. For variation, use varIndexBase + 2.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarScale, PaintVarScaleAroundCenter, PaintVarScaleUniform, and PaintVarScaleUniformAroundCenter formats use a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

NOTE Pure scaling can also be represented using the PaintTransform or PaintVarTransform table. For scaling about the origin, this could be done by setting xx and yy to x and y scale factors, and setting xy, yx, dx and dy = 0. The PaintScale table and variants provide more compact representation when only scaling is required.

5.7.11.2.6.11 Formats 24 to 27: PaintRotate, PaintVarRotate, PaintRotateAroundCenter, PaintVarRotateAroundCenter

Formats 24 to 27 are used to apply a rotation to a sub-graph. The paint table that is the root of the sub-graph is linked as a child. The amount of rotation is expressed directly as an angle, using a floating point value where 1.0 represents an angle of 180°.

Formats 24 and 25 apply rotations using the origin as the center of rotation. Format 25 allows for variation of the rotation in a variable font; format 24 provides a more compact representation when variation is not required.

Formats 26 and 27 apply rotations around a specified center of rotation. Format 27 allows for variation of the rotation or of the position of the center of rotation in a variable font; format 26 provides a more compact representation when variation is not required.

Formats 25 and 27 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

These tables use reduced precision for compactness. Where higher precision is required use PaintTransform/PaintVarTransform.

For general information regarding transformations in a color glyph definition, see 5.7.11.1.5.

PaintRotate table (format 24):

Type	**Name**	**Description**
uint8	format	Set to 24.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	angle	Rotation angle, 180° in counter-clockwise degrees per 1.0 of value.

PaintVarRotate table (format 25):

Type	**Name**	**Description**
uint8	format	Set to 25.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	angle	Rotation angle, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 0.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

PaintRotateAroundCenter table (format 26):

Type	**Name**	**Description**
uint8	format	Set to 26.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	angle	Rotation angle, 180° in counter-clockwise degrees per 1.0 of value.
FWORD	centerX	x coordinate for the center of rotation.
FWORD	centerY	y coordinate for the center of rotation.

PaintVarRotateAroundCenter table (format 27):

Type	Name	Description
uint8	format	Set to 27.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	angle	Rotation angle, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 0.
FWORD	centerX	x coordinate for the center of rotation. For variation, use varIndexBase + 1.
FWORD	centerY	y coordinate for the center of rotation. For variation, use varIndexBase + 2.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarRotate and PaintVarRotateAroundCenter formats use a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

NOTE 1 Pure rotation about a point can also be represented using the PaintTransform or PaintVarTransform table. For rotation about the origin, this could be done by setting matrix values as follows for angle θ:

— $xx = \cos(\theta)$

— $yx = \sin(\theta)$

— $xy = -\sin(\theta)$

— $yy = \cos(\theta)$

— $dx = dy = 0$

The important difference of the PaintRotate table and its variants is in allowing an angle to be specified directly in degrees, rather than as changes to basis vectors. In variable fonts, if a rotation angle needs to vary, it is easier to get smooth variation if an angle is specified directly than when using trigonometric functions to derive matrix elements.

NOTE 2 The rotation angle is represented using an F2DOT14 value, which supports values in the range [-2, 2). Since each 1.0 unit represents a change of 180°, rotation angles of [-360, 360) can be represented directly. Variations of the rotation angle are not limited to that range, however.

NOTE 3 If representation of rotation directly as an angle is preferred but higher precision is required to specify a center of rotation, a chained sequence of transforms can be used. For example, a PaintTransform can be used to align the origin to the desired center of rotation, then PaintRotate can be used for the desired rotation, and a second PaintTransform can be used to reset the origin.

When combining the transform effect of a PaintRotate table (or variants) with other transforms, the result shall be the same as if the rotation were represented using an equivalent matrix or sequence of matrices.

A rotation can result in the pre-transform position (0, 0) being moved elsewhere. See 5.7.11.2.6.8 regarding alignment of the transformed content with the destination.

5.7.11.2.6.12 Format 28 to 31: PaintSkew, PaintVarSkew, PaintSkewAroundCenter, PaintVarSkewAroundCenter

Formats 28 to 31 are used to apply a skew to a sub-graph. The paint table that is the root of the sub-graph is linked as a child. The amounts of skew in the x or y direction are expressed directly as angles, using floating point values where 1.0 represents an angle of 180°.

Formats 28 and 29 apply skews using the origin as the center of rotation for the skew. Format 29 allows for variation of the rotation in a variable font; format 28 provides a more compact representation when variation is not required.

Formats 30 and 31 apply skews around a specified center of rotation. Format 31 allows for variation of the rotation or of the position of the center of rotation in a variable font; format 30 provides a more compact representation when variation is not required.

Formats 29 and 31 shall not be used in non-variable fonts or if the COLR table does not have an ItemVariationStore subtable.

These tables use reduced precision for compactness. Where higher precision is required use PaintTransform/PaintVarTransform.

For general information regarding transformations in a color glyph definition, see 5.7.11.1.5.

PaintSkew table (format 28):

Type	Name	Description
uint8	format	Set to 28.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	xSkewAngle	Angle of skew in the direction of the x-axis, 180° in counter-clockwise degrees per 1.0 of value.
F2DOT14	ySkewAngle	Angle of skew in the direction of the y-axis, 180° in counter-clockwise degrees per 1.0 of value.

PaintVarSkew table (format 29):

Type	Name	Description
uint8	format	Set to 29.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	xSkewAngle	Angle of skew in the direction of the x-axis, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 0.
F2DOT14	ySkewAngle	Angle of skew in the direction of the y-axis, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 1.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

PaintSkewAroundCenter table (format 30):

Type	Name	Description
uint8	format	Set to 30.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	xSkewAngle	Angle of skew in the direction of the x-axis, 180° in counter-clockwise degrees per 1.0 of value.
F2DOT14	ySkewAngle	Angle of skew in the direction of the y-axis, 180° in counter-clockwise degrees per 1.0 of value.
FWORD	centerX	x coordinate for the center of rotation.
FWORD	centerY	y coordinate for the center of rotation.

PaintVarSkewAroundCenter table (format 31):

Type	Name	Description
uint8	format	Set to 31.
Offset24	paintOffset	Offset to a Paint subtable.
F2DOT14	xSkewAngle	Angle of skew in the direction of the x-axis, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 0.
F2DOT14	ySkewAngle	Angle of skew in the direction of the y-axis, 180° in counter-clockwise degrees per 1.0 of value. For variation, use varIndexBase + 1.

Type	Name	Description
FWORD	centerX	x coordinate for the center of rotation. For variation, use varIndexBase + 2.
FWORD	centerY	y coordinate for the center of rotation. For variation, use varIndexBase + 3.
uint32	varIndexBase	Base index into DeltaSetIndexMap.

The PaintVarSkew and PaintVarSkewAroundCenter formats use a base/sequence scheme to index into mapping data; see 5.7.11.4 for details.

NOTE 1 Pure skews about a point can also be represented using the PaintTransform or PaintVarTransform table. For skews about the origin, this could be done by setting matrix values as follows for x skew angle φ and y skew angle ψ:

— $xx = yy = 1$

— $yx = \tan(\psi)$

— $xy = -\tan(\varphi)$

— $dx = dy = 0$

The important difference of the PaintSkew table and its variants is in being able to specify skew as an angle, rather than as changes to basis vectors. In variable fonts, if a skew angle needs to vary, it is easier to get smooth variation if an angle is specified directly than when using trigonometric functions to derive matrix elements.

NOTE 2 The skew angles are represented using F2DOT14 values, which support values in the range [-2, 2). Since each 1.0 unit represents a change of 180°, skew angles of [-360, 360) can be represented directly. Variations of the skew angle are not limited to that range, however.

NOTE 3 If representation of skew directly as an angle is preferred but higher precision is required to specify a center of rotation, a chained sequence of transforms can be used. For example, a PaintTransform can be used to align the origin to the desired center of rotation, then PaintSkew can be used for the desired skew rotation, and a second PaintTransform can be used to reset the origin.

When combining the transform effect of a PaintSkew table (or variants) with other transforms, the result shall be the same as if the skew were represented using an equivalent matrix or sequence of matrices.

A skew can result in the pre-transform position (0, 0) being moved elsewhere. See 5.7.11.2.6.8 regarding alignment of the transformed content with the destination.

5.7.11.2.6.13 Format 32: PaintComposite

Format 32 is used to combine two layered compositions, referred to as *source* and *backdrop*, using different compositing or blending modes. The available compositing and blending modes are defined in an enumeration. For general information and examples, see 5.7.11.1.6.

NOTE The backdrop is also referred to as the “destination”.

PaintComposite table (format 32):

Type	Name	Description
uint8	format	Set to 32.
Offset24	sourcePaintOffset	Offset to a source Paint table.
uint8	compositeMode	A CompositeMode enumeration value.
Offset24	backdropPaintOffset	Offset to a backdrop Paint table.

The compositeMode value shall be one of the values defined in the CompositeMode enumeration, which are taken from the W3C Compositing and Blending Level 1 specification [33]. Details on each mode, including specifications of the required calculations using pixel color and alpha values, are provided in that specification. If an unrecognized value is encountered, COMPOSITE_CLEAR shall be used.

CompositeMode enumeration:

Value	**Name**	**Description**
	Porter-Duff modes	
0	COMPOSITE_CLEAR	Clear
1	COMPOSITE_SRC	Source ("Copy" in [33])
2	COMPOSITE_DEST	Destination
3	COMPOSITE_SRC_OVER	Source Over
4	COMPOSITE_DEST_OVER	Destination Over
5	COMPOSITE_SRC_IN	Source In
6	COMPOSITE_DEST_IN	Destination In
7	COMPOSITE_SRC_OUT	Source Out
8	COMPOSITE_DEST_OUT	Destination Out
9	COMPOSITE_SRC_ATOP	Source Atop
10	COMPOSITE_DEST_ATOP	Destination Atop
11	COMPOSITE_XOR	XOR
12	COMPOSITE_PLUS	Plus (“Lighter” in [33])
	Separable color blend modes:	
13	COMPOSITE_SCREEN	screen
14	COMPOSITE_OVERLAY	overlay
15	COMPOSITE_DARKEN	darken
16	COMPOSITE_LIGHTEN	lighten
17	COMPOSITE_COLOR_DODGE	color-dodge
18	COMPOSITE_COLOR_BURN	color-burn
19	COMPOSITE_HARD_LIGHT	hard-light
20	COMPOSITE_SOFT_LIGHT	soft-light
21	COMPOSITE_DIFFERENCE	difference
22	COMPOSITE_EXCLUSION	exclusion
23	COMPOSITE_MULTIPLY	multiply
	Non-separable color blend modes:	
24	COMPOSITE_HSL_HUE	hue
25	COMPOSITE_HSL_SATURATION	saturation
26	COMPOSITE_HSL_COLOR	color
27	COMPOSITE_HSL_LUMINOSITY	luminosity

The graphic compositions are defined by the source and backdrop paint tables and their respective sub-graphs. Conceptually, they are rendered into bitmaps, and the source is composited or blended into the backdrop using the specified composite mode.

While color values obtained from the CPAL table are represented in sRGB using the non-linear transfer function defined in the sRGB specification, the compositing and blending calculations are done after applying the inverse transfer function to derive linear-light RGB values. For more information regarding the non-linear and linear-light representations for sRGB, see *Interpolation of Colors* in 5.7.12.

As mentioned in 5.7.11.1.8.2, a color glyph definition shall be bounded. A sub-graph that has PaintComposite as its root is either bounded or unbounded, depending on the mode used and the boundedness of the source and backdrop sub-graphs. For each mode, boundedness is determined by the boundedness of the source and backdrop as follows:

— Always bounded:
  — COMPOSITE_CLEAR
— Bounded *if and only if* the source is bounded:
  — COMPOSITE_SRC
  — COMPOSITE_SRC_OUT
— Bounded *if and only if* the backdrop is bounded:
  — COMPOSITE_DEST
  — COMPOSITE_DEST_OUT
— Bounded *if and only if* either the source *or* backdrop is bounded:
  — COMPOSITE_SRC_IN
  — COMPOSITE_DEST_IN
— Bounded *if and only if* both the source *and* backdrop are bounded:
  — All other modes

5.7.11.3 COLR version 1 rendering algorithm

The various graphic concepts represented by COLR version 1 formats were individually described in 5.7.11.1, and the various formats were described in 5.7.11.2. Together, these provide most of the necessary details regarding how a color glyph is rendered. The following provides a comprehensive description of the rendering process, considering the graph as a whole.

The following algorithm can be used to render color glyphs defined using version 1 formats. Applications are not required to implement rendering using this algorithm, but shall produce equivalent results.

NOTE Checks for well-formedness and validity, as described in 5.7.11.1.9, are not repeated here. Actual implementations can integrate such checks with rendering processing.

1) Start with an initial drawing surface. As mentioned in 5.7.11.1.8.2, if a clip box is provided, it can be used to determine the size. Otherwise, the graph can be traversed to compute a required size.

2) Traverse the graph of a color glyph definition, starting with the root paint table referenced by a BaseGlyphPaintRecord, using the following pseudo-code function.

```
// render a paint table and its sub-graph
function renderPaint(paint)

   if format 1: // PaintColrLayers
     for each referenced child paint table, in bottom-up z-order:
       // for ordering, see 5.7.11.1.4, 5.7.11.2.6.1
       call renderPaint() passing the child paint table
       compose the returned graphic onto the surface using simple alpha blending

   if format 2 or 3: // PaintSolid, PaintVarSolid
     paint the specified color onto the surface

   if format 4, 5, 6, 7, 8 or 9:
     // PaintLinearGradient, PaintVarLinearGradient
     // PaintRadialGradient, PaintVarRadialGradient
     // PaintSweepGradient, PaintVarSweepGradient
     paint the gradient onto the surface following the gradient algorithm
```

```
if format 10: // PaintGlyph
  apply the outline of the referenced glyph to the clip region
    // take the intersection of clip regions—see 5.7.11.1.3
  call renderPaint() passing the child paint table
  restore the previous clip region

if format 11: // PaintColrGlyph
  call renderPaint() passing the paint table referenced by the base glyph ID

if format 12 to 31:
  // PaintTransform, PaintVarTransform
   // PaintTranslate, PaintVarTranslate
   // PaintScale*, PaintVarScale*
   // PaintRotate*, PaintVarRotate*
   // PaintSkew*, PaintVarSkew*
  apply the specified transform
     // compose the transform with the current transform state—see 5.7.11.1.5
  call renderPaint() passing the child paint table
  restore the previous transform state

if format 32: // PaintComposite

  // render backdrop sub-graph
  call renderPaint() passing the backdrop child paint table and save the result

  // render source sub-graph
  call renderPaint() passing the source child paint table and save the result

  // compose source and backdrop
  compose the source and backdrop using the specified composite mode

  // compose final result
  compose the result of the above composition onto the surface using simple
  alpha blending
```

5.7.11.4 COLR table and OFF Font Variations

The COLR table can be used in variable fonts. For color glyphs defined using version 0 formats, the glyph outlines can be variable, but no other aspect of the color glyph is variable. For color glyphs defined using version 1 formats, items that can be variable include the glyph outlines plus other aspects of the color glyph definition:

— Alpha values

— Color stop offsets in gradient color lines

— Placement of gradients onto the design grid

— The arguments of transformations (matrix elements, angles, etc.)

Variation data is provided in an Item Variation Store table (7.2.3) contained within the COLR table.

In a variable font, each value within the COLR version 1 formats that is variable needs to be associated with corresponding variation data (delta sets) in the Item Variation Store. This is done using a DeltaSetIndexMap table (defined in 7.3.5.2). The delta-set index mapping table contains an array of entries that provide indices mapping into sets of delta data in the Item Variation Store. Each variable item in the COLR table is given an index (base 0) into the mapping data. For example, if a variable item in the COLR table is given an index value of 5, the sixth entry in the mapping data is used to index into the Item Variation Store.

The indices for variable items in the COLR table are indicated using a base/sequence scheme. Each table or record that contains variable items will use a contiguous sequence of entries in the mapping array, and will include a *varIndexBase* field that indicates the first entry in the mapping array to be used. The variable fields within that table or record use entries in the mapping array, starting with the *varIndexBase* entry, in the order the fields occur in the table or record.

For example, the VarAffine2x3 table (5.7.11.2.6.8) has eight variable fields followed by the varIndexBase field. For the first variable field (*xx*), *varIndexBase + 0* is used as the index into the mapping array; for the second variable field (*yx*), *varIndexBase + 1* is used as the index into the mapping array; and so on.

If the index for a variable item is greater than or equal to the number of entries in the mapping array, the last mapping array entry shall be used.

The sequence of indices derived from a *varIndexBase* value do not wrap on overflow and shall not exceed 0xFFFFFFFF. A *varIndexBase* value of 0xFFFFFFFF is assigned a special meaning indicating that the variable fields in the given table or record do not have variation data.

Similarly, a delta-set index mapping entry with values 0xFFFF/0xFFFF can be used to indicate that an item has no variation data (see 7.2.3.2).

If the COLR table does not contain an Item Variation Store subtable, the *varIndexBase* field of variable tables or records shall be ignored by applications, and should be set to zero.

If the COLR table contains an Item Variation Store but does not contain a mapping table (varIndexMapOffset in the COLR header is NULL), then an implicit identity mapping is used: the sequence of values beginning with *varIndexBase* are treated directly as delta-set indices with 16-bit sub-fields for *outer* (high word) and *inner* (low word) index values. See 7.2.3.3 for more information regarding delta set indices.

For variable fonts that use COLR version 1 formats, special considerations apply to the effect of variation on the bounding box. See 5.7.11.1.8.2 for details.

For general information on OFF font variations, see 7.1.

5.7.12

Replace the first sentence of the first paragraph with the following:

The palette table is a set of one or more palettes, each containing a predefined number of color records.

Replace the second paragraph with the following text:

Palettes are defined by a set of color records. Each color record specifies a color in the sRGB color space using 8-bit BGRA (blue, green, red, alpha) representation. The sRGB color space is specified in IEC 61966-2-1. Details on the specification for the sRGB color space, including the color primaries and “gamma” transfer function, are also provided in CSS Color Module Level 4, section 10.2 [34].

All palettes have the same number of color records, specified by numColorRecords. All color records for all palettes are arranged in a single array, and the color records for any given palette are a contiguous sequence of color records within that array. The first color record of each palette is provided in the colorRecordIndices array.

Add the following paragraphs at the end of the subclause with the heading “Interpolation of colors”:

Interpolation of Colors

The SVG table and version 1 of the COLR table both support color gradient fills. The gradients are defined using color stops to specify color values at specific positions along a color line, with color values for other positions on the color line derived by interpolation.

When interpolating color values, linear interpolation between color stop positions is used. For example, suppose adjacent color stops are specified for positions 0.5 and 0.9 on a color line, and a color value is being calculated for position 0.8. The color value of the first color stop will contribute 75 % of the value

((0.8 - 0.5) / (0.9 - 0.5)), and the color value of the second color stop will contribute 25 % of the value. Interpolated values at each position of the color line are computed in this way for each of the R, G and B color components.

When interpolating color values, specific aspects of the representation of colors as well as handling of alpha need to be considered.

Representations of sRGB color values are expressed as levels of red, green and blue color “primaries” with specific, absolute chromaticity values, which are defined in the sRGB specification. Color-primary levels can potentially be expressed using a linear-light scale that correlates directly to light energy. (On a linear-light scale, for example, a doubling of a color value would correspond to a doubling of display luminance.) For sRGB, however, standard practice is to represent levels using a scale defined by a non-linear transfer function, sometimes referred to as “gamma”. This transfer function is also defined in the sRGB specification (see CSS Color Module Level 4, section 10.2 [34] for details). In the CPAL table, sRGB color values are always specified in terms of the non-linear, sRGB transfer function.

NOTE 1 An advantage of representing colors using a non-linear scale is that it allows more effective use of limited bit depth when color-primary levels are represented as integers: smaller differences in light energy can be represented for lower levels than for higher levels. This is beneficial since the human visual system is more sensitive to differences at low luminance levels than to differences at high luminance levels.

When interpolating colors, different results will be obtained if the interpolation is computed using the non-linear scale for color levels than if using the linear-light scale. For interoperable results, whether the non-linear or linear-light scale is to be used needs to be specified.

For gradient color values in the SVG table, the required interpolation behavior is defined in the SVG 1.1 specification: the ‘color-interpolation’ property can be used in an SVG document to declare whether interpolation is done using the non-linear sRGB scale (the default), or using a linear-light scale by applying the inverse sRGB transfer function.

For gradient color values in the COLR table, interpolation shall be computed using linear-light values (i.e., after applying the inverse sRGB transfer function).

After an interpolated color value is computed, whether or not the non-linear sRGB transfer function needs to be re-applied is determined by the requirements of the implementation context.

For both the COLR and SVG tables, interpolation shall be done with alpha pre-multiplied into each linearized R, G and B component. For alpha specified in a CPAL ColorRecord, the value is converted to a floating value in the range [0, 1.0] by dividing by 255, then multiplied into each R, G and B component. In the COLR table, color references in formats used for version 1 include a separate alpha value; that alpha value (with variation, in a variable font) is multiplied into the R, G and B components as well. Interpolated values are then calculated by linear interpolation using these pre-multiplied, linear-light R, G and B values.

NOTE 2 Alpha components use a linear scale and can be directly interpolated apart from the R, G and B components without any linearization step.

Once interpolation of the pre-multiplied red, green and blue values and of the alpha value is complete, the red, green and blue results are then un-premultiplied by dividing each interpolated value by the corresponding interpolated alpha.

While color values are specified as 8-bit integers, the interpolation computations will require greater precision in each of the linearization, pre-multiply, and interpolation steps. Also, when rendered results are to be presented on an imaging device with known characteristics, visual banding artifacts in a gradient can be minimized by taking full advantage of the color bit depth supported by the device. For instance, if a display supports 10- or 12-bit quantization per color channel, then ideally the ramp of color values in a gradient would use that level of quantization. Other factors from the presentation context may, however, also affect the available capabilities. Therefore, no minimum level of precision is specified as a requirement

6.4.1

Replace the Script tags table with the following:

Script	Script Tag
Adlam	'adlm'
Ahom	'ahom'
Anatolian Hieroglyphs	'hluw'
Arabic	'arab'
Armenian	'armn'
Avestan	'avst'
Balinese	'bali'
Bamum	'bamu'
Bassa Vah	'bass'
Batak	'batk'
Bengali	'beng'
Bengali v.2	'bng2'
Bhaiksuki	'bhks'
Bopomofo	'bopo'
Brahmi	'brah'
Braille	'brai'
Buginese	'bugi'
Buhid	'buhd'
Byzantine Music	'byzm'
Canadian Syllabics	'cans'
Carian	'cari'
Caucasian Albanian	'aghb'
Chakma	'cakm'
Cham	'cham'
Cherokee	'cher'
Chorasmian	'chrs'
CJK Ideographic	'hani'
Coptic	'copt'
Cypriot Syllabary	'cprt'
Cypro-Minoan	‘cpmn’
Cyrillic	'cyrl'
Default	'DFLT'
Deseret	'dsrt'
Devanagari	'deva'
Devanagari v.2	'dev2'
Dives Akuru	'diak'
Dogra	'dogr'
Duployan	'dupl'
Egyptian hieroglyphs	'egyp'
Elbasan	'elba'
Elymaic	'elym'
Ethiopic	'ethi'

Script	Script Tag
Georgian	'geor'
Glagolitic	'glag'
Gothic	'goth'
Grantha	'gran'
Greek	'grek'
Gujarati	'gujr'
Gujarati v.2	'gjr2'
Gunjala Gondi	'gong'
Gurmukhi	'guru'
Gurmukhi v.2	'gur2'
Hangul	'hang'
Hangul Jamo	'jamo'
Hanifi Rohingya	'rohg'
Hanunoo	'hano'
Hatran	'hatr'
Hebrew	'hebr'
Hiragana	'kana'
Imperial Aramaic	'armi'
Inscriptional Pahlavi	'phli'
Inscriptional Parthian	'prti'
Javanese	'java'
Kaithi	'kthi'
Kannada	'knda'
Kannada v.2	'knd2'
Katakana	'kana'
Kayah Li	'kali'
Kharosthi	'khar'
Khitan Small Script	'kits'
Khmer	'khmr'
Khojki	'khoj'
Khudawadi	'sind'
Lao	'lao '
Latin	'latn'
Lepcha	'lepc'
Limbu	'limb'
Linear A	'lina'
Linear B	'linb'
Lisu (Fraser)	'lisu'
Lycian	'lyci'
Lydian	'lydi'
Mahajani	'mahj'
Makasar	'maka'
Malayalam	'mlym'
Malayalam v.2	'mlm2'
Mandaic, Mandaean	'mand'

Script	Script Tag
Manichaean	'mani'
Masaram Gondi	'gonm'
Marchen	'marc'
Mathematical Alphanumeric Symbols	'math'
Medefaidrin (Oberi Okaime, Oberi Ɔkaimɛ)	'medf'
Meitei Mayek (Meithei, Meetei)	'mtei'
Mende Kikakui	'mend'
Meroitic Cursive	'merc'
Meroitic Hieroglyphs	'mero'
Miao	'plrd'
Modi	'modi'
Mongolian	'mong'
Mro	'mroo'
Multani	'mult'
Musical Symbols	'musc'
Myanmar	'mymr'
Myanmar v.2	'mym2'
Nabataean	'nbat'
Nandinagari	'nand'
Newa	'newa'
New Tai Lue	'talu'
Nyiakeng Puachue Hmong	'hmnp'
N'Ko	'nko '
Nüshu	'nshu'
Odia (formerly Oriya)	'orya'
Odia (formerly Oriya) v.2	'ory2'
Ogham	'ogam'
Ol Chiki	'olck'
Old Italic	'ital'
Old Hungarian	'hung'
Old North Arabian	'narb'
Old Permic	'perm'
Old Persian Cuneiform	'xpeo'
Old Sogdian	'sogo'
Old South Arabian	'sarb'
Old Turkic, Orkhon Runic	'orkh'
Old Uyghur	‘ougr’
Osage	'osge'
Osmanya	'osma'
Pahawh Hmong	'hmng'
Palmyrene	'palm'
Pau Cin Hau	'pauc'
Phags-pa	'phag'
Phoenician	'phnx'
Psalter Pahlavi	'phlp'

Script	Script Tag
Rejang	'rjng'
Runic	'runr'
Samaritan	'samr'
Saurashtra	'saur'
Sharada	'shrd'
Shavian	'shaw'
Siddham	'sidd'
Sign Writing	'sgnw'
Sinhala	'sinh'
Sogdian	'sogd'
Sora Sompeng	'sora'
Soyombo	'soyo'
Sumero-Akkadian Cuneiform	'xsux'
Sundanese	'sund'
Syloti Nagri	'sylo'
Syriac	'syrc'
Tagalog	'tglg'
Tagbanwa	'tagb'
Tai Le	'tale'
Tai Tham (Lanna)	'lana'
Tai Viet	'tavt'
Takri	'takr'
Tamil	'taml'
Tamil v.2	'tml2'
Tangsa	‘tnsa’
Tangut	'tang'
Telugu	'telu'
Telugu v.2	'tel2'
Thaana	'thaa'
Thai	'thai'
Tibetan	'tibt'
Tifinagh	'tfng'
Tirhuta	'tirh'
Toto	‘toto’
Ugaritic Cuneiform	'ugar'
Vai	'vai '
Vithkuqi	‘vith’
Wancho	'wcho'
Warang Citi	'wara'
Yezidi	'yezi'
Yi	'yi '
Zanabazar Square (Zanabazarin Dörböljin Useg, Xewtee Dörböljin Bicig, Horizontal Square Script)	'zanb'

6.4.2

Replace the Language systems and tags table with the following:

Language System	**Language System Tag**	**Corresponding ISO 639 ID (if applicable)**
Abaza	'ABA '	abq
Abkhazian	'ABK '	abk
Acholi	'ACH '	ach
Achi	'ACR '	acr
Adyghe	'ADY '	ady
Afrikaans	'AFK '	afr
Afar	'AFR '	aar
Agaw	'AGW '	ahg
Aiton	'AIO '	aio
Akan	'AKA '	aka, fat, twi
Batak Angkola	'AKB '	akb
Alsatian	'ALS '	gsw
Altai	'ALT '	atv, alt
Amharic	'AMH '	amh
Anglo-Saxon	'ANG '	ang
Phonetic transcription—Americanist conventions	APPH	
Arabic	'ARA '	ara
Aragonese	'ARG '	arg
Aari	'ARI '	aiw
Rakhine	'ARK '	mhv, rmz, rki
Assamese	'ASM '	asm
Asturian	'AST '	ast
Athapaskan languages	'ATH '	aht, apa, apk, apj, apl, apm, apw, ath, bea, sek, bcr, caf, chp, clc, coq, crx, ctc, den, dgr, gce, gwi, haa, hoi, hup, ing, kkz, koy, ktw, kuu, mvb, nav, qwt, scs, srs, taa, tau, tcb, tce, tfn, tgx, tht, tol, ttm, tuu, txc, wlk, xup, xsl
Avatime	'AVN '	avn
Avar	'AVR '	ava
Awadhi	'AWA '	awa
Aymara	'AYM '	aym
Torki	'AZB '	azb
Azerbaijani	'AZE '	aze
Badaga	'BAD '	bfq
Banda	BAD0	bad, bbp, bfl, bjo, bpd, bqk, gox, kuw, liy, lna, lnl, mnh, nue, nuu, tor, yaj, zmz
Baghelkhandi	'BAG '	bfy
Balkar	'BAL '	krc
Balinese	'BAN '	ban
Bavarian	'BAR '	bar
Baulé	'BAU '	bci
Batak Toba	'BBC '	bbc

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Berber	'BBR '	auj, ber, cnu, gha, gho, grr, jbe, jbn, kab, mzb, oua, rif, sds, shi, shy, siz, sjs, swn, taq, tez, thv, thz, tia, tjo, tmh, ttq, tzm, zen, zgh
Bench	'BCH '	bcq
Bible Cree	'BCR '	
Bandjalang	'BDY '	bdy
Belarussian	'BEL '	bel
Bemba	'BEM '	bem
Bengali	'BEN '	ben
Haryanvi	'BGC '	bgc
Bagri	'BGQ '	bgq
Bulgarian	'BGR '	bul
Bhili	'BHI '	bhi, bhb
Bhojpuri	'BHO '	bho
Bikol	'BIK '	bik, bhk, bcl, bto, cts, bln, fbl, lbl, rbl, ubl
Bilen	'BIL '	byn
Bislama	'BIS '	bis
Kanauji	'BJJ '	bjj
Blackfoot	'BKF '	bla
Baluchi	'BLI '	bal
Pa'o Karen	'BLK '	blk
Balante	'BLN '	bjt, ble
Balti	'BLT '	bft
Bambara (Bamanankan)	'BMB '	bam
Bamileke	'BML '	bai, bbj, bko, byv, fmp, jgo, nla, nnh, nnz, nwe, xmg, ybb
Bosnian	'BOS '	bos
Bishnupriya Manipuri	'BPY '	bpy
Breton	'BRE '	bre
Brahui	'BRH '	brh
Braj Bhasha	'BRI '	bra
Burmese	'BRM '	mya
Bodo	'BRX '	brx
Bashkir	'BSH '	bak
Burushaski	'BSK '	bsk
Batak Dairi (Pakpak)	'BTD '	btd
Beti	'BTI '	btb, beb, bum, bxp, eto, ewo, mct
Batak languages	'BTK '	akb, bbc, btd, btk, btm, bts, btx, btz
Batak Mandailing	'BTM '	btm
Batak Simalungun	'BTS '	bts
Batak Karo	'BTX '	btx
Batak Alas-Kluet	'BTZ '	btz
Bugis	'BUG '	bug
Medumba	'BYV '	byv
Kaqchikel	'CAK '	cak
Catalan	'CAT '	cat

Language System	**Language System Tag**	**Corresponding ISO 639 ID (if applicable)**
Zamboanga Chavacano	'CBK '	cbk
Chinantec	CCHN	cco, chj, chq, chz, cle, cnl, cnt, cpa, csa, cso, cte, ctl, cuc, cvn
Cebuano	'CEB '	ceb
Chiga	'CGG '	cgg
Chamorro	'CHA '	cha
Chechen	'CHE '	che
Chaha Gurage	'CHG '	sgw
Chattisgarhi	'CHH '	hne
Chichewa (Chewa, Nyanja)	'CHI '	nya
Chukchi	'CHK '	ckt
Chuukese	CHK0	chk
Choctaw	'CHO '	cho
Chipewyan	'CHP '	chp
Cherokee	'CHR '	chr
Chuvash	'CHU '	chv
Cheyenne	'CHY '	chy
Western Cham	'CJA '	cja
Eastern Cham	'CJM '	cjm
Comorian	'CMR '	swb, wlc, wni, zdj
Coptic	'COP '	cop
Cornish	'COR '	cor
Corsican	'COS '	cos
Creoles	'CPP '	abs, acf, afs, aig, aoa, bah, bew, bis, bjs, bpl, bpq, brc, bxo, bzj, bzk, cbk, ccl, ccm, chn, cks, cpe, cpf, cpi, cpp, cri, crp, crs, dcr, dep, djk, fab, fng, fpe, gac, gcf, gcl, gcr, gib, goq, gpe, gul, gyn, hat, hca, hmo, hwc, icr, idb, ihb, jam, jvd, kcn, kea, kmv, kri, kww, lir, lou, lrt, max, mbf, mcm, mfe, mfp, mkn, mod, msi, mud, mzs, nag, nef, ngm, njt, onx, oor, pap, pcm, pea, pey, pga, pih, pis, pln, pml, pmy, pov, pre, rcf, rop, scf, sci, skw, srm, srn, sta, svc, tas, tch, tcs, tgh, tmg, tpi, trf, tvy, uln, vic, vkp, wes, xmm
Cree	'CRE '	cre
Carrier	'CRR '	crx, caf
Crimean Tatar	'CRT '	crh
Kashubian	'CSB '	csb
Church Slavonic	'CSL '	chu
Czech	'CSY '	ces
Wayanad Chetti	'CTT '	ctt
Chittagonian	'CTG '	ctg
San Blas Kuna	'CUK '	cuk
Dagbani	'DAG '	dag
Danish	'DAN '	dan
Dargwa	'DAR '	dar
Dayi	'DAX '	dax
Woods Cree	'DCR '	cwd
German	'DEU '	deu

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Dogri (individual language)	'DGO '	dgo
Dogri (macrolanguage)	'DGR '	doi
Dhangu	'DHG '	dhg
Divehi (Dhivehi, Maldivian)	'DHV ' (deprecated)	div
Dimli	'DIQ '	diq
Divehi (Dhivehi, Maldivian)	'DIV '	div
Zarma	'DJR '	dje
Djambarrpuyngu	DJR0	djr
Dangme	'DNG '	ada
Dan	'DNJ '	dnj
Dinka	'DNK '	din
Dari	'DRI '	prs
Dhuwal	'DUJ '	duj, dwu, dwy
Dungan	'DUN '	dng
Dzongkha	'DZN '	dzo
Ebira	'EBI '	igb
Eastern Cree	'ECR '	crj, crl
Edo	'EDO '	bin
Efik	'EFI '	efi
Greek	'ELL '	ell
Eastern Maninkakan	'EMK '	emk
English	'ENG '	eng
Erzya	'ERZ '	myv
Spanish	'ESP '	spa
Central Yupik	'ESU '	esu
Estonian	'ETI '	est
Basque	'EUQ '	eus
Evenki	'EVK '	evn
Even	'EVN '	eve
Ewe	'EWE '	ewe
French Antillean	'FAN '	acf
Fang	FAN0	fan
Persian	'FAR '	fas
Fanti	'FAT '	fat
Finnish	'FIN '	fin
Fijian	'FJI '	fij
Dutch (Flemish)	'FLE '	vls
Fe’fe’	'FMP '	fmp
Forest Enets	'FNE '	enf
Fon	'FON '	fon
Faroese	'FOS '	fao
French	'FRA '	fra
Cajun French	'FRC '	frc
Frisian	'FRI '	fry

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Friulian	'FRL '	fur
Arpitan	'FRP '	frp
Futa	'FTA '	fuf
Fulah	'FUL '	ful
Nigerian Fulfulde	'FUV '	fuv
Ga	'GAD '	gaa
Scottish Gaelic (Gaelic)	'GAE '	gla
Gagauz	'GAG '	gag
Galician	'GAL '	glg
Garshuni	'GAR '	
Garhwali	'GAW '	gbm
Geez	'GEZ '	gez
Githabul	'GIH '	gih
Gilyak	'GIL '	niv
Kiribati (Gilbertese)	GIL0	gil
Kpelle (Guinea)	'GKP '	gkp
Gilaki	'GLK '	glk
Gumuz	'GMZ '	guk
Gumatj	'GNN '	gnn
Gogo	'GOG '	gog
Gondi	'GON '	gon
Greenlandic	'GRN '	kal
Garo	'GRO '	grt
Guarani	'GUA '	grn
Wayuu	'GUC '	guc
Gupapuyngu	'GUF '	guf
Gujarati	'GUJ '	guj
Gusii	'GUZ '	guz
Haitian (Haitian Creole)	'HAI '	hat
Haida	'HAI0'	hai, hax, hdn
Halam (Falam Chin)	'HAL '	cfm
Harauti	'HAR '	hoj
Hausa	'HAU '	hau
Hawaiian	'HAW '	haw
Haya	'HAY '	hay
Hazaragi	'HAZ '	haz
Hammer-Banna	'HBN '	amf
Heiltsuk	'HEI '	hei
Herero	'HER '	her
Hiligaynon	'HIL '	hil
Hindi	'HIN '	hin
High Mari	'HMA '	mrj
A-Hmao	'HMD '	hmd
Hmong	'HMN '	hmn

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Hiri Motu	'HMO '	hmo
Hmong Shuat	'HMZ '	hmz
Hindko	'HND '	hno, hnd
Ho	'HO '	hoc
Harari	'HRI '	har
Croatian	'HRV '	hrv
Hungarian	'HUN '	hun
Armenian	'HYE '	hye, hyw
Armenian East	HYE0	hye
Iban	'IBA '	iba
Ibibio	'IBB '	ibb
Igbo	'IBO '	ibo
Ido	'IDO '	ido
Ijo languages	'IJO '	iby, ijc, ije, ijn, ijo, ijs, nkx, okd, okr, orr
Interlingue	'ILE '	ile
Ilokano	'ILO '	ilo
Interlingua	'INA '	ina
Indonesian	'IND '	ind
Ingush	'ING '	inh
Inuktitut	'INU '	iku
Nunavik Inuktitut	'INUK'	ike, iku
Inupiat	'IPK '	ipk
Phonetic transcription—IPA conventions	IPPH	
Irish	'IRI '	gle
Irish Traditional	'IRT '	gle
Irula	'IRU '	iru
Icelandic	'ISL '	isl
Inari Sami	'ISM '	smn
Italian	'ITA '	ita
Hebrew	'IWR '	heb
Jamaican Creole	'JAM '	jam
Japanese	'JAN '	jpn
Javanese	'JAV '	jav
Lojban	'JBO '	jbo
Krymchak	'JCT '	jct
Yiddish	'JII '	yid
Ladino	'JUD '	lad
Jula	'JUL '	dyu
Kabardian	'KAB '	kbd
Kabyle	KAB0	kab
Kachchi	'KAC '	kfr
Kalenjin	'KAL '	kln
Kannada	'KAN '	kan

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Karachay	'KAR '	krc
Georgian	'KAT '	kat
Kawi (Old Javanese)	‘KAW ’	kaw
Kazakh	'KAZ '	kaz
Makonde	'KDE '	kde
Kabuverdianu (Crioulo)	'KEA '	kea
Kebena	'KEB '	ktb
Kekchi	'KEK '	kek
Khutsuri Georgian	'KGE '	kat
Khakass	'KHA '	kjh
Khanty-Kazim	'KHK '	kca
Khmer	'KHM '	khm
Khanty-Shurishkar	'KHS '	kca
Khamti Shan	'KHT '	kht
Khanty-Vakhi	'KHV '	kca
Khowar	'KHW '	khw
Kikuyu (Gikuyu)	'KIK '	kik
Kirghiz (Kyrgyz)	'KIR '	kir
Kisii	'KIS '	kqs, kss
Kirmanjki	'KIU '	kiu
Southern Kiwai	'KJD '	kjd
Eastern Pwo Karen	'KJP '	kjp
Bumthangkha	'KJZ '	kjz
Kokni	'KKN '	kex
Kalmyk	'KLM '	xal
Kamba	'KMB '	kam
Kumaoni	'KMN '	kfy
Komo	'KMO '	kmw
Komso	'KMS '	kxc
Khorasani Turkic	'KMZ '	kmz
Kanuri	'KNR '	kau
Kodagu	'KOD '	kfa
Korean Old Hangul	'KOH '	kor, okm
Konkani	'KOK '	kok
Komi	'KOM '	kom
Kikongo	'KON '	ktu
Kongo	KON0	kon
Komi-Permyak	'KOP '	koi
Korean	'KOR '	kor
Kosraean	'KOS '	kos
Komi-Zyrian	'KOZ '	kpv
Kpelle	'KPL '	kpe
Krio	'KRI '	kri
Karakalpak	'KRK '	kaa

Language System	Language System Tag	Corresponding ISO 639 ID (if applicable)
Karelian	'KRL '	krl
Karaim	'KRM '	kdr
Karen	'KRN '	blk, bwe, eky, ghk, jkm, jkp, kar, kjp, kjt, ksw, kvl, kvq, kvt, kvu, kvy, kxf, kxk, kyu, pdu, pwo, pww, wea
Koorete	'KRT '	kqy
Kashmiri	'KSH '	kas
Ripuarian	KSH0	ksh
Khasi	'KSI '	kha
Kildin Sami	'KSM '	sjd
S’gaw Karen	'KSW '	ksw
Kuanyama	'KUA '	kua
Kui	'KUI '	kxu
Kulvi	'KUL '	kfx
Kumyk	'KUM '	kum
Kurdish	'KUR '	kur
Kurukh	'KUU '	kru
Kuy	'KUY '	kdt
Kwak’wala	'KWK '	kwk
Koryak	'KYK '	kpy
Western Kayah	'KYU '	kyu
Ladin	'LAD '	lld
Lahuli	'LAH '	bfu
Lak	'LAK '	lbe
Lambani	'LAM '	lmn
Lao	'LAO '	lao
Latin	'LAT '	lat
Laz	'LAZ '	lzz
L-Cree	'LCR '	crm
Ladakhi	'LDK '	lbj
Lelemi	'LEF '	lef
Lezgi	'LEZ '	lez
Ligurian	'LIJ '	lij
Limburgish	'LIM '	lim
Lingala	'LIN '	lin
Lisu	'LIS '	lis
Lampung	'LJP '	ljp
Laki	'LKI '	lki
Low Mari	'LMA '	mhr
Limbu	'LMB '	lif
Lombard	'LMO '	lmo
Lomwe	'LMW '	ngl
Loma	'LOM '	lom
Lipo	'LPO '	lpo
Luri	'LRC '	lrc, luz, bqi, zum