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)



ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

# 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.



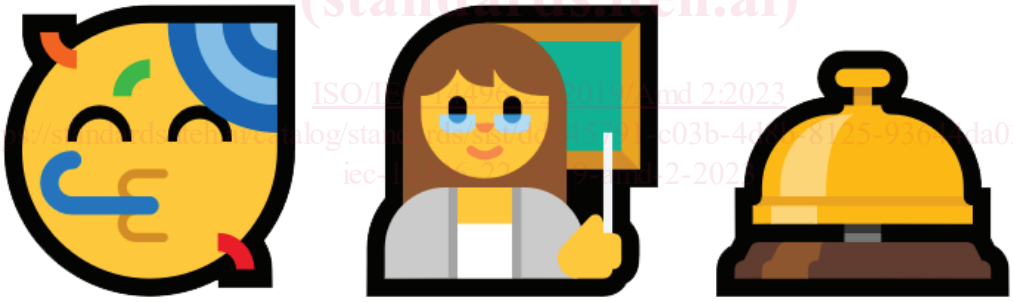


**Figure 5.6 — Examples of the graphic capabilities of COLR version 0**

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

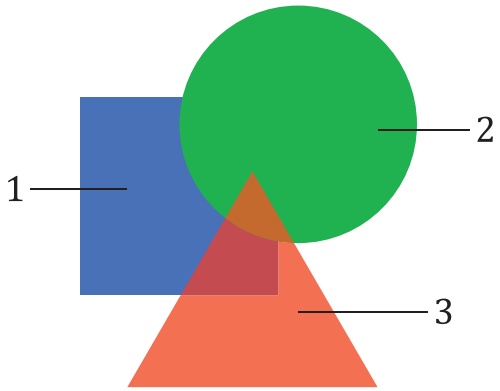
— Composition and blending modes—different ways that the content of a layer is combined with the content of layers above or below it

— 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.

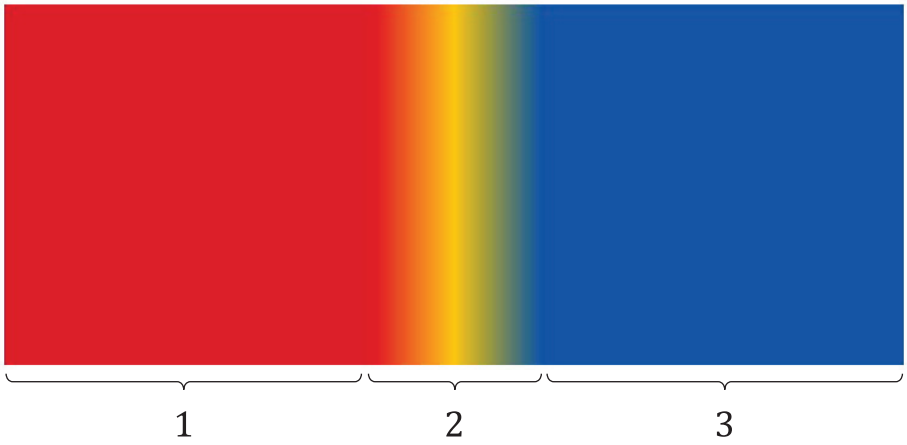
The color patterns outside the defined interval are determined by the color line's extend mode. Three extend modes are supported:

— **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*…".

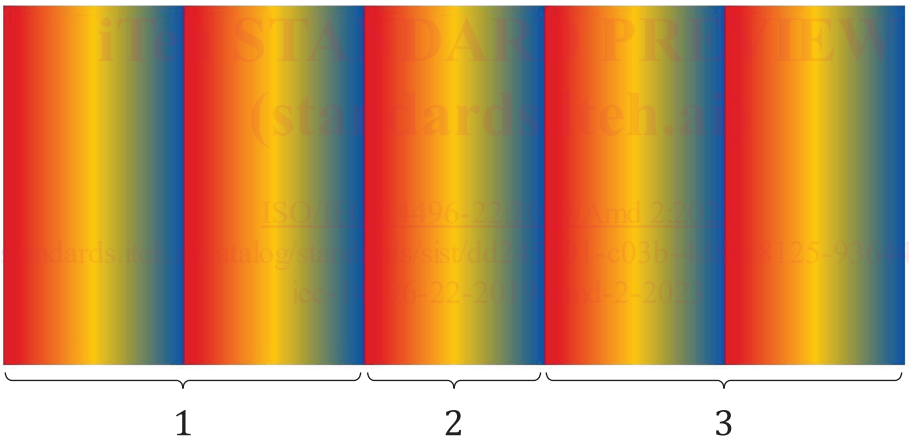
Figures 5.9–5.11 illustrate the different color line extend modes. The figures show the color line extended over a limited interval, but the extension is unbounded in either direction.




**Key**

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

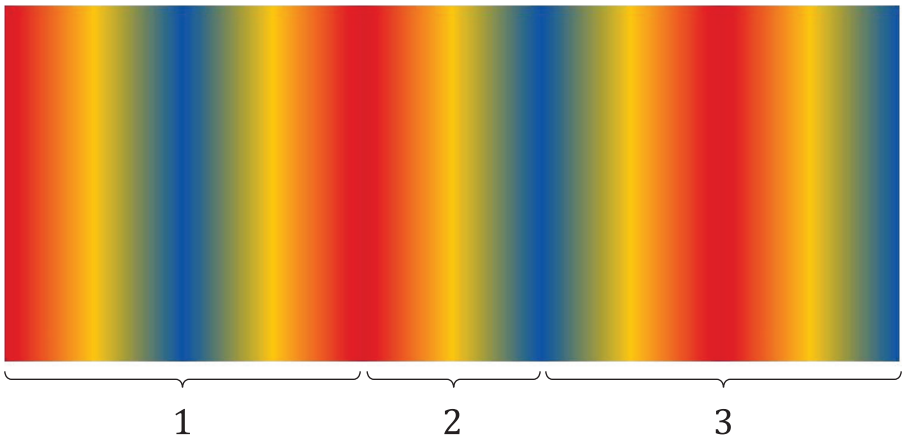
**Figure 5.9 — Color gradation extended using pad mode**




**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**

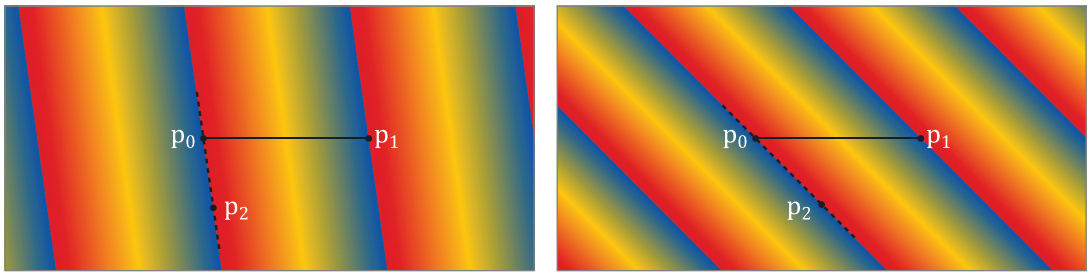



**Figure 5.13 — Linear gradients with different rotations using the repeat extend mode**

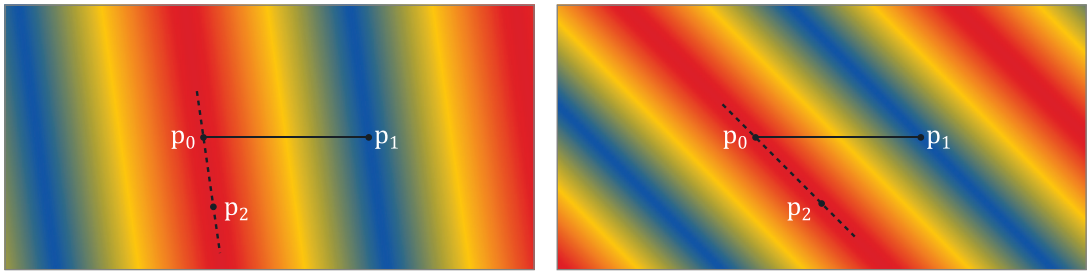



**Figure 5.14 — Linear gradients with different rotations using the reflect extend mode**

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 $x(\omega) = (x_1-x_0)\omega + x_0$ Let $y(\omega) = (y_1-y_0)\omega + y_0$ Let $r(\omega) = (r_1-r_0)\omega + r_0$ 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.