INTERNATIONAL STANDARD

ISO/IEC 14496-22

Fourth edition
2019-01

AMENDMENT 1
2020-06

Information technology — Coding of audio-visual objects —

Part 22: **Open Font Format**

AMENDMENT 1: Color font technology and other updates

Technologies de l'information — Codage des objets audiovisuels —

Partie 22: Format de police de caractères ouvert

AMENDEMENT 1: Technologie des polices colorées et autres mises à jour

Reference number
ISO/IEC 14496-22:2019/Amd.1:2020(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).

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 http://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.

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 website.

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.

Information technology — Coding of audio-visual objects —

Part 22:
Open Font Format

AMENDMENT 1: Color font technology and other updates

4.5.2

Replace the description of the "Offset" field in the "Table Directory" table with the following:

Offset from beginning of OFF font file.

5.3.4.1.1

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

This is the table information needed if numberOfContours is greater than or equal to zero, that is, a glyph is not a composite.

Replace the final paragraph (immediately following the "Simple Glyph flags" table) with the following:

A non-zero-fill algorithm is needed to avoid dropouts when contours overlap. The OVERLAP_SIMPLE flag is used by some rasterizer implementations to ensure that a non-zero-fill algorithm is used rather than an even-odd-fill algorithm. Implementations that always use a non-zero-fill algorithm will ignore this flag. Note that some implementations might check this flag specifically in non-variable fonts, but always use a non-zero-fill algorithm for variable fonts. This flag can be used in order to provide broad interoperability of fonts — particularly non-variable fonts — when glyphs have overlapping contours.

Note that variable fonts often make use of overlapping contours. This has implications for tools that generate static-font data for a specific instance of a variable font, if broad interoperability of the derived font is desired: if a glyph has overlapping contours in the given instance, then the tool should either set this flag in the derived glyph data, or else should merge contours to remove overlap of separate contours.

5.4.3.10

In the descriptions of both "FDSelect Format3" and "Range3 Record Format", add the following NOTE after the last paragraph:

NOTE Since a sentinel GID is used to delimit the last range in the array, its value, encoded as a uint16, cannot exceed the value 65535. Therefore, the last GID encoded when using FDSelect Format3 cannot exceed 65534.

In the description of "FDSelect Format4", in the first paragraph, replace the reference to 65536 [glyphs] with 65535.

In the description of “Range4 Record Format” replace all references to 65536 [glyphs] with 65535.

5.4.3.11

In the 'blend' row of the table – replace "number of blends" with the "numberOfBlends".

5.5.1

Replace the entire content of subclause 5.5.1 with the following:

This table contains SVG descriptions for some or all of the glyphs in the font.

OFF provides various formats for color fonts, one of which is the SVG table. The SVG table provides the benefits of supporting scalable color graphics using the Scalable Vector Graphics markup language, a vector graphics file format that is widely used on the Web and that provides rich graphics capabilities, such as gradients.

SVG was developed for use in environments that allow for a rich set of functionality, including leveraging the full functionality of Cascading Style Sheets for styling, and programmatic manipulation of graphics objects using the SVG Document Object Model. Adoption of SVG for use in OpenType does not entail wholesale incorporation of all SVG capabilities. Text-rendering engines typically have more stringent security, performance and architectural requirements than general-purpose SVG engines. For this reason, when used within OFF fonts, the expressiveness of the language is limited and simplified to be appropriate for environments in which font processing and text layout occurs.

The SVG table is optional, and may be used in OFF fonts with TrueType, CFF or CFF2 outlines. For every SVG glyph description, there must be a corresponding TrueType, CFF or CFF2 glyph description in the font.

SVG Table Header

Type	Name	Description
uint16	version	Table version (starting at 0). Set to 0.
Offset32	offsetToSVGDocumentList	Offset the SVG Documents List, from the start of the SVG table. Must be non-zero.
uint32	reserved	Set to 0.

SVG Document List

The SVG document list provides a set of SVG documents, each of which defines one or more glyph descriptions.

Type	Name	Description
uint16	numEntries	Number of SVG Document Index Entries. Must be non-zero.
SVGDocumentRecord	documentRecords[numEntries]	Array of SVG document records.

SVGDocumentRecord

Each SVG document record specifies a range of glyph IDs (from startGlyphID to endGlyphID, inclusive), and the location of its associated SVG document in the SVG table.

Type	Name	Description
uint16	startGlyphID	The first glyph ID for the range covered by this record.
uint16	endGlyphID	The last glyph ID for the range covered by this record.
Offset32	svgDocOffset	Offset from the beginning of the SVGDocumentList to an SVG document. Must be non-zero.
uint32	svgDocLength	Length of the SVG document data. Must be non-zero.

Records must be sorted in order of increasing startGlyphID. For any given record, the startGlyphID must be less than or equal to the endGlyphID of that record, and also must be greater than the endGlyphID of any previous record.

NOTE Two or more records can point to the same SVG document. In this way, a single SVG document can provide glyph descriptions for discontinuous glyph ID ranges. See Example 1 in subclause 5.5.6

5.5.2

Insert a new subclause 5.5.2 and renumber the remaining subclauses within subclause 5.5:

5.5.2 SVG Documents

SVG specification

The SVG markup language used in the SVG table shall be as defined in the Scalable Vector Graphics (SVG) 1.1 (2nd edition), W3C Recommendation. Any additional SVG features are not supported, unless explicitly indicated otherwise.

Previous editions of this document allowed use of context-fill and other context-* property values, which are defined in the draft SVG 2 specification. Use of these properties is deprecated: conforming implementations may support these properties, but support is not required or recommended, and use of these properties in fonts is strongly discouraged.

Document encoding and format

SVG documents within an OFF SVG table may either be plain text or gzip-encoded, and applications that support the SVG table shall support both.

The gzip format is defined in RFC 1952 (Reference [31]). Within a gzip-encoded SVG document, the deflate compression method (defined by RFC 1951) must be used. Thus, the first three bytes of the gzip-encoded document header must be 0x1F, 0x8B, 0x08.

Whether compressed or plain-text transfer encoding is used, the SVGDocLength field of the SVG document record specifies the length of the encoded data, not the decoded document.

The encoding of the (uncompressed) SVG document must be UTF-8.

While SVG 1.1 is defined as an XML application, some SVG implementations for the Web use an "HTML dialect". The "HTML dialect" differs from the XML-based definition in various ways, including being case-insensitive (XML is case-sensitive), and not requiring an *xmlns* attribute on the SVG root element. Applications that support the OFF SVG table shall support the XML-based definition for SVG 1.1. Applications may use SVG-parsing libraries that also support the "HTML dialect". However, SVG documents within the OFF fonts must always conform to the XML-based definition.

While SVG 1.1 requires conforming interpreters to support XML namespace constructs, applications that support the OpenType SVG table are not required to have full support for XML namespaces. The root element of each SVG document must declare SVG as the default namespace:

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
```

If the XLink *href* attribute is used, the root must also declare "xlink" as a namespace in the root element:

```
<svg version="1.1"    xmlns="http://www.w3.org/2000/svg"
                      xmlns:xlink="http://www.w3.org/1999/xlink">
```

No other XLink attributes or other mechanisms may be used anywhere in the document. Also, no other namespace declarations should be made in any element.

SVG capability requirements and restrictions

Most SVG 1.1 capabilities are supported in OFF and should be supported in all OFF applications that support the SVG table. Some SVG 1.1 capabilities are not required and may be optionally supported in applications. Certain other capabilities are not supported in OFF and must not be used in SVG documents within OFF fonts.

The following capabilities are restricted from use in OFF and must not be used in conforming fonts. If use of associated elements is encountered within a font, conforming applications must ignore and not render those elements.

— <text>, <font>, and associated elements

— <foreignObject> elements

— <switch> elements

— <script> elements

— <a> elements

— <view> elements

— XSL processing

— Use of relative units em, ex

— Use of SVG data within <image> elements

— Use of color profiles (the <icccolor> data type, the <color-profile> element, the color-profile property, or the CSS @color-profile rule)

— Use of the contentStyleType attribute

— Use of CSS2 system color keywords

SVG documents may include <desc>, <metadata> or <title> elements, but these are ignored by implementations.

Support for the following capabilities is not required in conforming implementations, though some applications may support them. Font developers should evaluate support for these capabilities in the target environments in which their fonts will be used before using them in fonts. To ensure interoperability across the broadest range of environments, use of these capabilities should be avoided.

— Internal CSS stylesheets (expressed using the <style> element)

— CSS inline styles (expressed using the style attribute)

— CSS variables (custom properties) — but see further qualifications below

- CSS media queries, calc() or animations
- SVG animations
- SVG child elements
- <filter> elements and associated attributes, including enableBackground
- <pattern> elements
- <mask> elements
- <marker> elements
- <symbol> elements
- Use of XML entities
- Use of image data within <image> elements in formats other than JPEG or PNG
- Interactivity capabilities (event attributes, zoomAndPan attributes, the cursor property, or <cursor> elements)

NOTE 1 In fonts intended for broad distribution, use of XML presentation attributes for styling is recommended over CSS styling as that will have the widest support across implementations.

NOTE 2 Use of media queries to react to environment changes within a glyph description is not recommended, even when fonts are used in applications that provide CSS media query support. Instead, a higher-level presentation framework is expected to handle environment changes. The higher-level framework can interact with options implemented within the font using OpenType mechanisms such as glyph substitution, or selection of color palettes.

While supporting the use of CSS variables is optional, it is strongly recommended that all implementations support the CSS var() function for color variables defined in the CPAL table. Fonts should not define any variables within an SVG document; var() should only be used in attributes or properties that accept a color value, and should only occur as the first item in the value. See subclause 5.5.3 for more information.

While support for patterns and masks is not required, all conforming implementations must support gradients (<linearGradient> and <radialGradient> elements), clipping paths and opacity properties.

Conforming implementations must support all other capabilities of SVG 1.1 that are not listed above as restricted or as optional and best avoided for broad interoperability.

5.5.3

Rename subclause 5.5.3 as "**Color and color palettes**" and replace the content of subclause 5.5.3 with the following:

In SVG 1.1, color values can be specified in various ways. For some of these, special considerations apply when used in the SVG table. Also, OFF provides a mechanism for alternate, user-selectable color palettes that can be used within SVG glyph descriptions.

Colors

Implementations must support numerical RGB specifications; for example, "#ffbb00", or "rgb(255,187,0)". Implementations must also support all of the recognized color keywords supported in SVG 1.1. However, CSS2 system color keywords are not supported and must not be used.

Some implementations may use graphics engines that happen to support RGBA specifications using the rgba() function. This is not supported in OFF, however, and rgba() specifications must not be used in conforming fonts. Note that SVG 1.1 provides opacity properties that can achieve the same effects.

Implementations must also support the “currentColor” keyword. The initial value must be set by the text-layout engine or application environment. This can be set in whatever way is considered most appropriate for the application. In general, it is recommended that this be set to the text foreground color applied to a given run of text.

NOTE Within an SVG document, the value of “currentColor” for any element is the current color property value for that element. If a color property is set explicitly on an element, it will reset the “currentColor” value for that element and its children. Doing so will override the value set by the host environment. In SVG documents within the SVG table, there is no scenario in which it would be necessary to set a color property value since any effects can be achieved in other ways. It is best practice to avoid setting a color property value.

Color palettes

Implementations can optionally support color palettes defined in the CPAL table in subclause 5.7.12. The CPAL table allows the font designer to define one or more palettes, each containing a number of colors. All palettes defined in the font have the same number of colors, which are referenced by base-zero index. Within an SVG document in the SVG table, colors in a CPAL palette are referenced as implementation-defined CSS variables (custom properties), using the var() function.

Support for the CPAL table and palettes in implementations is strongly recommended. Implementations that support palettes must support the CSS var() function for purposes of referencing palette entries as custom properties. Fonts should only use custom properites and the var() function to reference CPAL palette entries. Fonts should not define any variables within an SVG document. The var() function should only be used in attributes or properties that accept a color value, and should only occur as the first item in the value.

NOTE Even if an implementation does not support CPAL palettes, it is strongly recommended that the var() function be supported, and that the implementation is able to apply a fallback value specified as a second var() argument if the first argument (the color variable) is not supported. This will allow fonts intended for wide distribution to include use of the CPAL table but to be able to specify fallback colors in case CPAL palettes are not supported in some applications.

The text-layout engine or application defines a custom property for each palette entry and assigns color values to each one. Custom color properties should only be defined for fonts that include a CPAL table. In general, the values of the custom properties should be set using palette entries from the CPAL table, though applications can assign values derived by other means, such as user input. When assigning values from CPAL palette entries, the first palette should normally be used by default. If the font has palettes marked with the USABLE_WITH_LIGHT_BACKGROUND or USABLE_WITH_DARK_BACKGROUND flag, however, one of these palettes can be used as the default instead.

However, the values are assigned, the number of custom properties defined must be numPaletteEntries, as specified in the CPAL table header. The custom-property names must be of the form “--color<num>”, where <num> is a non-zero-padded decimal number in the range [0, numPaletteEntries-1]. For example, “--color0”, “--color1”, and so on.

The following illustrates how a color variable might be used in an SVG glyph description:

```
<path fill="var(--color0, yellow)" d="..."/>
```

In implementations that do support color variables and palettes, the color value assigned to the variable will be applied. If an implementation does not support color variables and palettes, however, the color variable will be ignored, and the fallback color value, yellow, will be applied.

Palette entries in the CPAL table are specified as BGRA values. (CPAL alpha values are in the range 0 to 255, where 0 is fully transparent and 255 is fully opaque.) Note that SVG 1.1 supports RGB color values, but not RGBA/BGRA color values. As noted above, use of rgba() color values within SVG documents in the SVG table is not supported and must not be used in conforming fonts. Alpha values in CPAL entries are supported, however. When a CPAL color entry is applied to a fill or stroke property of a shape element, to the stop-color of a gradient stop element, or to the flood-color property of an feFlood filter element, then the alpha value from that palette entry must be converted to a value in the range [0.0 – 1.0] and multiplied into the corresponding fill-opacity, stroke-opacity, stop-opacity or flood-opacity property of

the same element. If an implementation supports feDiffuseLighting or feSpecularLighting filters and a palette entry is applied to the lighting-color property, then the alpha value is ignored. When the alpha value is applied in this way to an opacity property of an element, it is the original opacity property value that is inherited by child elements, not the computed result of applying the alpha value to the opacity property. The alpha value is inherited as a component of the color-related property (fill, stroke, etc.), however.

5.5.4

Replace the content of subclause 5.5.4 with the following:

Each SVG document defines one or more glyph description. For each glyph ID in the glyph ID range of a document record within the SVG Document list, the associated SVG document shall contain an element with ID “glyph<glyphID>”, where <glyphID> is the glyph ID expressed as a non–zero-padded decimal value. This element functions as the SVG glyph description for the given glyph ID.

For example, suppose a font with 100 glyphs (glyph IDs 0 – 99) has SVG glyph definitions only for its last 5 glyphs. Suppose also that the last SVG glyph definition has its own SVG document, but that the other four glyphs are defined in a single SVG document (to take advantage of shared graphical elements, for instance). There will be two document records, the first with glyph ID range [95, 98]; and the second, with glyph ID range [99, 99]. The SVG document referenced by the first record will contain elements with id “glyph95”, “glyph96”, “glyph97”, and “glyph98”. The SVG document referenced by the second record will contain an element with id “glyph99”.

Glyph identifiers may appear deep within an SVG element hierarchy, but SVG itself does not define how partial SVG documents are to be rendered. Thus, font engines shall render an element designated in this way as the glyph description for a given glyph ID according to SVG’s <use> tag behaviour, as though the given element and its content were specified in a <defs> tag and then referenced as the graphic content of an SVG document. For example, consider the following SVG document, which defines two glyphs:


```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
  <defs>...</defs>
  <g id="glyph13">...</g>
  <g id="glyph14">...</g>
</svg>
```

NOTE The <g> element in SVG is a container for grouping of elements, not a “glyph” element.

When a font engine renders glyph 14, the result shall be the same as rendering the following SVG document:

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/
xlink">
  <defs>
    <defs>...</defs>
    <g id="glyph14">...</g>
  </defs>
  <use xlink:href="#glyph14" />
</svg>
```

5.5.5

Rename subclause 5.5.5 as “**Glyph semantics and text layout processing**” and replace the content with the following:

An SVG glyph description in the SVG table is an alternate to the corresponding glyph description with the same glyph ID in the 'glyf', 'CFF ' or CFF2 table. The SVG glyph description must provide a depiction of the same abstract glyph as the corresponding TrueType/CFF glyph description.

When SVG glyph descriptions are used, text layout is done in the same manner, using the 'cmap', 'hmtx', GSUB and other tables. This results in an array of final glyph IDs arranged at particular x,y positions on

