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

a surface (along with applicable scale/rotation matrices). After this layout processing is done, available SVG descriptions are used in rendering, instead of the TrueType/CFF descriptions. For each glyph ID in the final glyph array, if an SVG glyph description is available for that glyph ID, then it is rendered using an SVG engine; otherwise, the TrueType or CFF glyph description is rendered. Glyph advance widths or heights are the same for SVG glyphs as for TrueType/CFF glyphs, though there may be small differences in glyph ink bounding boxes. Because advances are the same, switching between SVG and non-SVG rendering should not require re-layout of lines unless the line layout depends on bounding boxes.

Coordinate systems and glyph metrics

The default SVG coordinate system is vertically mirrored in comparison to the TrueType/CFF design grid: the positive y-axis points downward, rather than usual convention for OpenType of the positive y-axis pointing upward. In other respects, the default coordinate system of an SVG document corresponds to the TrueType/CFF design grid: the default units for the SVG document are equivalent to font design units; the SVG origin (0,0) is aligned with the origin in the TrueType/CFF design grid; and y = 0 is the default horizontal baseline used for text layout.

The size of the initial viewport for the SVG document is the em square: height and width both equal to head.unitsPerEm. If a viewBox attribute is specified on the <svg> element with width or height values different from the unitsPerEm value, this will have the effect of a scale transformation on the SVG user coordinate system. Similarly, if height or width attributes are specified on the <svg> element, this will also have the effect of a scale transformation on the coordinate system.

Although the initial viewport size is the em square, the viewport must not be clipped. The <svg> element is assumed to have a clip property value of auto, and an overflow property value of visible. A font should not specify clip or overflow properties on the <svg> element. If clip or overflow properties are specified on the <svg> element with any other values, they must be ignored.

NOTE Because SVG uses a y-down coordinate system, then by default, glyphs will often be drawn in the +x –y quadrant of the SVG coordinate system. (See Example 2.) In many other environments, however, graphic elements need to be in the +x +y quadrant to be visible. Font development tools are expected to provide an appropriate transfer between a design environment and the representation within the font's SVG table. In Example 3, a viewBox attribute is used to shift the viewport up. In Example 4, a translate transform is used on container elements to shift the graphic elements that comprise the glyph descriptions.

Glyph advance widths are specified in the 'hmtx' table; advance heights are specified in the 'vmtx' table. Note that glyph advances are static and cannot be animated.

As with CFF glyphs, no explicit glyph bounding boxes are recorded. Note that left side bearing values in the 'hmtx' table, top side bearings in the 'vmtx' table, and bit 1 in the flags field of the 'head' table are not used for SVG glyph descriptions. The "ink" bounding box of the rendered SVG glyph should be used if a bounding box is desired; this box may be different for animated versus static renderings of the glyph.

Glyph advances and positions can be adjusted by data in the GPOS or 'kern' tables. Note that data in the GPOS and kern table use the y-up coordinate system, as with TrueType or CFF glyph descriptions. When applied to SVG glyph descriptions, applications must handle the translation between the y-up coordinate system and the y-down coordinates used for the SVG glyph descriptions.

5.5.6

Rename subclause 5.5.6 as "**Animations**" and replace the content with the following:

Some implementations may support use of animations—either SVG animation or CSS animation. Note that support for animation is optional and is not recommended in fonts intended for wide distribution.

Applications that support animations may, in some cases, require a static rendering for glyphs that include animations. This may be needed, for example, when printing. Note that a static rendering is obtained by ignoring and not running any animations in the SVG document, not by allowing animations to run and capturing the initial frame at time = 0.

Note that glyph advance widths and advance heights are defined in the 'hmtx' and 'vmtx' tables and cannot be animated. A glyph's bounding box may change during animation but should remain within the glyph advance width/height and the font's default line metrics to avoid collision with other text elements.

5.5.7

Rename the subclause 5.5.7 as "**Examples**" and replace the first two paragraphs with the following:

The SVG code in these examples is presented exactly as could be used in the SVG documents of an OFF font with SVG glyph descriptions. The code is not optimized for brevity.

Add the following text as Example 1:

Example 1: SVG table header and document list

This example shows an SVG table header and document list.

In the SVG Document List, multiple SVG Document Records can point to the same SVG document. In this way, a single SVG document can provide glyph descriptions for a discontinuous range of glyph IDs. This example shows multiple records in the Document List pointing to the same SVG document.

Example 1:

Hex Data	Source	Comments
	SVGHeader	
0000	Version	Table version = 0
0000000A	offsetToSVGDocumentList	Offset to document list
00000000	Reserved	Offset to AttachList table
	SVGDocumentList	
0005	numEntries	Five documentRecord entries, index 0 to 4
	documentRecords[0]	Document record for glyph ID range [1,1]
0001	startGlyphID	
0001	endGlyphID	
0000003E	svgDocOffset	Offset to SVG document for glyph1
0000019F	svgDocLength	Length of SVG document for glyph1
	documentRecords[1]	Document record for glyph ID range [2,2]
0002	startGlyphID	
0002	endGlyphID	
000001DD	svgDocOffset	Offset to SVG document for glyph2. The same SVG document is also used for glyph13 and glyph14.
000002FF	svgDocLength	Length of SVG document for glyph2 – length of the entire SVG document, covering glyphs 2, 13 and 14.
	documentRecords[2]	Document record for glyph ID range [3,12]
0003	startGlyphID	
000C	endGlyphID	
000004DC	svgDocOffset	Offset to SVG document for glyphs 3 to 12
000006F4	svgDocLength	Length of SVG document for glyphs 3 to 12
	documentRecords[3]	Document record for glyph ID range [13,14]

000D	startGlyphID	
000E	endGlyphID	
000001DD	svgDocOffset	Offset to SVG document for glyphs 2, 13 and 14. Offset is the same as for documentRecords[1].
000002FF	svgDocLength	Length of SVG document for glyphs 2, 13 and 14. Length is the same as for documentRecords[1].
	documentRecords[4]	Document record for glyph ID range [15,19]
000F	startGlyphID	
0013	endGlyphID	
00000BD0	svgDocOffset	Offset to SVG document for glyphs 15 to 19
00000376	svgDocLength	Length of SVG document for glyphs 15 to 19

Example 4 illustrates an SVG document with glyph descriptions for the discontinuous ranges [2, 2], [13, 14].

Rename "**Example: Glyph specified directly in expected position**" as "**Example 2: Glyph specified directly in expected position**" and rearrange the content of the example to move the textual description to the beginning, followed by the code example and the illustration.

Replace the textual description of "**Example 2: Glyph specified directly in expected position**" with the following:

In this example, the letter "i" is drawn directly in the +x –y quadrant of the SVG coordinate system, upright, with its baseline on the x axis. Note that the y attribute of the <rect> elements specifies the top edge, with the height of the rectangle below that.

Rename "**Example: Glyph shifted up with viewBox**" as "**Example 3: Glyph shifted up with viewBox**" and rearrange the content of the example to move the textual description to the beginning, followed by the code example.

Replace the textual description of "**Example 3: Glyph shifted up with viewBox**" with the following:

When designing in an SVG illustration application, it may be most natural to draw objects in the +x +y quadrant of the SVG coordinate system. In this example, the glyph description of the letter "i" is specified with upright orientation in the +x +y quadrant as though the baseline were at y = 1000 in the SVG coordinate system. A viewBox in the <svg> element is then used to shift the viewport down by 1000 units so that the actual baseline aligns with the design's baseline.

NOTE When using a viewBox attribute on the <svg> element, it is important to specify unitsPerEm for width and height values to avoid a scaling effect. See "Coordinate Systems and Glyph Metrics" above for more information.

Replace the last sentence of "**Example 3: Glyph shifted up with viewBox**" with the following:

The visual result is the same as that shown for Example 2.

Rename the "**Example: Common elements shared across glyphs in same SVG doc**" to "**Example 4: Common elements shared across glyphs in same SVG doc**" and rearrange the content of the example to move textual description to the beginning, followed by the code example and the illustration.

Replace the textual description of "**Example 4: Common elements shared across glyphs in same SVG doc**" with the following:

In this example, the base of the letter 'i' is specified as a component within the <defs> element so that it can be re-used across three glyphs. This shared component is referenced using the identifier "i-base". In glyph2, the component is used alone to comprise the dotless 'i'. In glyph13, a dot is added on top. In glyph14, an acute accent is adds a dot on top. Glyph ID 14 adds an acute accent on top.

This example also illustrates the use of a translate transform to shift elements drawn in the +x +y quadrant of the SVG coordinate system so that they appear in the +x –y quadrant, above the baseline.

Replace the last paragraph of "**Example 4: Common elements shared across glyphs in same SVG doc**" with the following:

The following image shows the visual results for glyph IDs 2, 13, and 14, from left to right.

Rename "**Example: Specifying current text color in glyphs**" as "**Example 5: Specifying current Color in glyphs**" and rearrange the content of the example to move the textual description to the beginning, followed by the code example and the illustration.

Replace the textual description of "**Example 5: Specifying current Color in glyphs**" with the following:

This example uses the same glyph description for "i" as in Example 2 with one modification: the "darkblue" color value for the dot of the "i" is replaced with the "currentColor" keyword. The application sets the color value for currentColor, typically with the text foreground color.

In the code block of "**Example 5: Specifying current Color in glyphs**", replace "context-fill" in the last <rect> element with "currentColor".

Replace the last paragraph of "**Example 5: Specifying current Color in glyphs**" with the following:

The following image illustrate visual results with currentColor set to two different color values by the application: black (left), and red (right).

Rename "**Example: Specifying color palette variables in glyphs**" as "**Example 6: Specifying color palette variables in glyphs**" and rearrange the content of the example to move textual description to the beginning, followed by the code example and the illustration.

Replace the textual description of "**Example 6: Specifying color palette variables in glyphs**" with the following:

This example uses the same glyph description for "i" as in Example 2, but with a modification: the stop colors of the linear gradient are specified using color variables --color0 and --color1.

The font in this example includes a CPAL table. The value for the color variables is set by the application, typically using CPAL entries. The CPAL table assumed in this example has two palettes, each with two entries, with BGRA color values as follows:

— palette 0: {8B0000FF, B3AA00FF}

— palette 1: {800080FF, D670DAFF}

(SVG equivalents for colors in palette 0 would be {darkblue, #00aab3}. SVG equivalents for colors in palette 1 would be {purple, orchid}.)

In each of the var() invocations used to reference the color variables, a second parameter for a fallback color value has been specified. The values match the values used in palette 0. These fallback values will be used if the application does not support the CPAL table.

After the code block of "**Example 6: Specifying color palette variables in glyphs**", add the following text paragraph:

The following images show the visual results in three situations, from left to right:

After the illustration of "**Example 6: Specifying color palette variables in glyphs**", add the following text:

The first case, on left, shows the result when the value of the color variables have been set using palette 0 from the CPAL table. The second case, in the middle, shows the result when values have been set using palette 1.

The third case, on the right, shows a result in which the application has set the values of the color variables using a custom palette with user-specified colors:

— --color0: red

— --color1: orange

Note that, in all three cases, the dot of the "i" is still dark blue, since this is hard coded in the glyph description and not controlled by a color variable.

If the application has not set values for --color0 and --color1 (because it does not support the CPAL table, for example), then the fallback values provided in the var() functions (darkblue and #00aab3, respectively) are used. Note that these are in fact the same colors as in the first (default) CPAL color palette, which means the glyph will render as in the first case shown above.

Rename "**Example: Embedding a PNG in an SVG glyph**" as "**Example 7: Embedding a PNG in an SVG glyph**" and rearrange the content of the example to move textual description to the beginning, followed by the code example and the illustration.

Replace the textual description of "**Example 7: Embedding a PNG in an SVG glyph**" with the following:

In this example, PNG data is embedded within an <image> element.

A typical use case for embedding PNG data is detailed artwork in a lettering font.

After the code block of "**Example 7: Embedding a PNG in an SVG glyph**", add the following text paragraph:

The following image shows the visual result:

5.7.9

In "**vhea – Vertical header table**" in the version 1.1 vertical header table, in the table entry defining "vertTypoAscender", replace the text of the Description field with the following:

The vertical typographic ascender for this font. It is the distance in FUnits from the ideographic em-box center baseline for the vertical axis to the right edge of the ideographic em-box.

It is usually set to (head.unitsPerEm)/2. For example, a font with an em of 1000 FUnits will set this field to 500. See subclause 6.4.4 for a description of the ideographic em-box.

In "**vhea – Vertical header table**", in the version 1.1 vertical header table, in the table entry defining "vertTypoDescender", replace the text of the Description field with the following:

The vertical typographic descender for this font. It is the distance in FUnits from the ideographic em-box center baseline for the vertical axis to the left edge of the ideographic em-box.

It is usually set to (head.unitsPerEm)/2. For example, a font with an em of 1000 FUnits will set this field to –500.

6.2.3

In the last section entitled "SVG and CPAL", replace the text of the second paragraph with the following:

Foreground color is expressed by the "currentColor" keyword in the SVG glyph descriptions.

6.2.3

In "**Table Organization**", in third paragraph, replace the last sentence to read:

OFF Layout has eight types of GSUB lookups (described in the GSUB subclause) and nine types of GPOS lookups (described in the GPOS subclause).

6.2.6

In "**Coverage table**", in the description of Coverage Table, replace the last paragraph to read:

A Coverage table defines a unique index value (Coverage Index) for each covered glyph. The Coverage Indexes are sequential, from 0 to the number of covered glyphs minus 1. This unique value specifies the position of the covered glyph in the Coverage table. The client uses the Coverage Index to look up values in the subtable for each glyph.

In "**Coverage table**", in the description of Coverage Format 2 replace the second paragraph to read:

The Coverage Indexes for the first range begin with zero (0) and increase sequentially to (endGlyphId – startGlyphId). For each successive range, the starting Coverage Index is one greater than the ending Coverage Index of the the preceding range. Thus, startCoverageIndex for each non-initial range must

equal the length of the preceding range (endGlyphID – startGlyphID + 1) added to the startGlyphIndex of the preceding range. This allows for a quick calculation of the Coverage Index for any glyph in any range using the formula: Coverage Index (glyphID) = startCoverageIndex + glyphID – startGlyphID.

6.3.3.2

In "**GPOS lookup type descriptions**" in the first paragraph of Lookup Type 1 description, replace the first sentence of the paragraph with the following:

A single adjustment positioning subtable (SinglePos) is used to adjust the placement or advance of a single glyph, such as a subscript or superscript.

In "**GPOS lookup type descriptions**" in the first paragraph of Lookup Type 2 description, replace the first sentence of the paragraph with the following:

A pair adjustment positioning subtable (PairPos) is used to adjust the placement or advance of two glyphs in relation to one another – for instance, to specify kerning data for pairs of glyphs.

In "**GPOS lookup type descriptions**" in the first paragraph of Lookup Type 3 description, add the following paragraphs immediately after the first paragraph, and preceding the CursivePosFormat1 subtable description:

Positioning adjustments from anchor alignment may be either horizontal or vertical. Note that the positioning effects in the text-layout direction (horizontal, for horizontal layout) work differently for than for the cross-stream direction (vertical, in horizontal layout):

— For adjustments in the line-layout direction, the layout engine adjusts the advance of the first glyph (in logical order). This effectively moves the second glyph relative to the first so that the anchors are aligned in that direction.

— For the cross-stream direction, placement of one glyph is adjusted to make the anchors align. Which glyph is adjusted is determined by the rightToLeft flag in the parent lookup table: if the rightToLeft flag is clear, the second glyph is adjusted to align anchors with the first glyph; if the rightToLeft flag is set, the first glyph is adjusted to align anchors with the second glyph.

Note that, if the rightToLeft lookup flag is set, then the last glyph in the connected sequence keeps its initial position in the cross-stream direction relative to the baseline, and the cross-stream positions of the preceding, connected glyphs are adjusted.

In "**GPOS lookup type descriptions**" in the Lookup Type 4 description, replace the last sentence of the second paragraph with the following:

When a mark is combined with a given base, the mark placement is adjusted so that the mark anchor is aligned with the base anchor for the applicable mark class. Placement of the base glyph and advances of both glyphs are not affected.

In "**GPOS lookup type descriptions**" in the Lookup Type 5 description, add the following paragraphs immediately after the fourth paragraph, and preceding the MarkLigPosFormat1 subtable description:

As with mark-to-base attachment, when a mark is combined with a given ligature base, the mark placement is adjusted so that the mark anchor is aligned with the applicable base anchor. Placement of the base glyph and advances of both glyphs are not affected.

In "**GPOS lookup type descriptions**" in the Lookup Type 6 description, replace the fourth paragraph with the following:

The mark2 glyph that combines with a mark1 glyph is the glyph preceding the mark1 glyph in glyph string order (skipping glyphs according to LookupFlags). The subtable applies precisely when that mark2 glyph is covered by mark2Coverage. To combine the mark glyphs, the placement of the mark1 glyph is adjusted such that the relevant attachment points coincide. Advance widths are not affected. The input context for MarkToBase, MarkToLigature and MarkToMark positioning tables is the mark that is being positioned. If a sequence contains several marks, a lookup may act on it several times, to position them.

6.3.3.3

In "**Shared tables: Value record, Anchor table and MarkArray table**", in the ValueRecord table description – replace the descriptions of the DeviceOffset fields (xPlaDeviceOffset, yPlaDeviceOffset, xAdvDeviceOffset, yAdvDeviceOffset) with the following:

xPlaDeviceOffset :

Offset to Device table (non-variable font) / VariationIndex table (variable font) for horizontal placement, from beginning of the immediate parent table (SinglePos or pairPosFormat2 lookup subtable, PairSet table within a PairPosFormat1 lookup subtable) – may be NULL.

yPlaDeviceOffset :

Offset to Device table (non-variable font) / VariationIndex table (variable font) for vertical placement, from beginning of the immediate parent table (SinglePos or pairPosFormat2 lookup subtable, PairSet table within a PairPosFormat1 lookup subtable) – may be NULL.

xAdvDeviceOffset :

Offset to Device table (non-variable font) / VariationIndex table (variable font) for horizontal advance, from beginning of the immediate parent table (SinglePos or pairPosFormat2 lookup subtable, PairSet table within a PairPosFormat1 lookup subtable) – may be NULL.

yAdvDeviceOffset :

Offset to Device table (non-variable font) / VariationIndex table (variable font) for vertical advance, from beginning of the immediate parent table (SinglePos or pairPosFormat2 lookup subtable, PairSet table within a PairPosFormat1 lookup subtable) – may be NULL.

6.3.3.4

In "**GPOS subtable examples**" in Example 5, replace the Comment field of the "class1Records[0]" with the following:

First Class1Record, for contexts beginning with class 0

6.3.4.1

In "**GSUB – Table overview**" in the fifth paragraph, in the bulleted list entry describing "Contextual substitution", replace the first sentence with the following:

Contextual substitution is a powerful extension of the above lookup types, describing glyph substitutions in context – that is, a substitution of one or more glyphs within a certain pattern of glyphs.

6.3.4.3

In "**GSUB – Lookup type descriptions**" in the Lookup Type 5 description, replace the first paragraph with the following:

A Contextual Substitution (ContextSubst) subtable defines a powerful type of glyph substitution lookup: it describes glyph substitutions in context that replace one or more glyphs within a certain pattern of glyphs.

6.4.2

In "**Language tags**" in the description of Language tags, replace the text of the sixth paragraph with the following:

All tags are four-character strings composed of a limited set of ASCII characters; for details regarding the Tag data type see §4.3 "Data types". By convention, registered language system tags use three or four capital letters (0x41 – 0x5A).

In "**Language tags**" in the table describing Language System tags, make the following table content changes:

— *In the row describing the Language System* "Bikol" – *replace the corresponding ISO 639 ID with* "bik, bhk, bcl, bto, cts, bln*, fbl, lbl, rbl, ubl".*

— *In the row describing the Language System* "Beti" – *replace the corresponding ISO 639 ID with* "btb, beb, bum, bxp, eto, ewo, mct*".*

— *In the row describing the Language System* "Creoles" – *replace the corresponding ISO 639 ID with* "crp, cpe, cpf, cpp*".*

— *In the row describing the Language System* "Dhuwal" – *replace the corresponding ISO 639 ID with* "duj, dwu, dwy*".*

— *In the row describing the Language System* "Forest Nenets" – *replace the corresponding ISO 639 ID with* "enf, yrk*".*

— *In the row describing the Language System* "Halam" – *replace the Language System name of* "Halam" *with* "Halam (Falam Chin)" *and replace the corresponding ISO 639 ID with* "flm, cfm, rnl*".*

— *In the row describing the Language System* "Armenian" – *replace the corresponding ISO 639 ID with* "hye, hyw*".*

— *In the row describing the Language System* "Ijo languages" – *replace the corresponding ISO 639 ID with* "ijc, ijo*".*

— *In the table describing Language System tags – add the new row for Language system* "Bumthngkha" *with the Language System tag* 'KJZ' *and the corresponding ISO 639 ID* "kjz*".*

— *In the row describing the Language System* "Nisi" – *replace the corresponding ISO 639 ID with* "dap, njz, tgj*".*

— *In the row describing the Language System* "Provencal" – *replace the Language System name of* "Provencal" *with* "Provençal / Old Provençal*".*

— *In the row describing the Language System* "Silte Gurage" – *replace the corresponding ISO 639 ID with* "xst, stv, wle*".*

— *In the row describing the Language System* "Tundra Nenets" – *replace the corresponding ISO 639 ID with* "enh, yrk*".*