# Scalable Vector Graphics (SVG) 1.0 Specification

## W3C Recommendation *04 September 2001*

**This version:**
http://www.w3.org/TR/2001/REC-SVG-20010904/
(Available as: PDF, zip archive of HTML)
**Latest version:**
http://www.w3.org/TR/SVG/
**Previous version:**
http://www.w3.org/TR/2001/PR-SVG-20010719/
**Editor:**
Jon Ferraiolo <jferraio@adobe.com>
**Authors:**
See author list



## Abstract

This specification defines the features and syntax for Scalable Vector Graphics (SVG), a language for describing two-dimensional vector and mixed vector/raster graphics in XML.

## Status of this document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.*

This document has been reviewed by W3C Members and other interested parties and has been

endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced by the W3C SVG Working Group as part of the activity of the Graphics Activity within the W3C Document Formats Domain. The goals of the W3C SVG 1.0 Working Group are discussed in the W3C SVG WG Charter (11 November 2000 - members only). The W3C SVG Working Group has maintained a public Web page http://www.w3.org/Graphics/SVG/ which contains further background information.

The SVG Working Group believes that all features in the SVG 1.0 Recommendation are implementable due to substantial implementation experience with generators, viewers and transcoders based on the SVG specification, the amount of SVG content that has been developed to date, and interoperability results against the SVG conformance test suite. A report on implementation status was made at the end of the Candidate Recommendation review period. The implementation results are publicly released and are intended solely to be used as proof of SVG 1.0 implementability. It is only a snap shot of the actual implementation behaviors at one moment of time, as these implementations may not be immediately available to the public. The interoperability data is not intended to be used for assessing or grading the performance of any individual implementation. It is intended that this report will be updated from time to time by the follow-on activity that oversees the SVG Recommendation.

There are patent disclosures and license commitments associated with the SVG 1.0 specification. These may be found on the SVG 1.0 Patent Statements in conformance with W3C policy.

This version of this document incorporates some editorial changes from earlier versions. A list of changes since the Proposed Recommendation specification of 19 July, 2001 is available.

Public discussion of issues related to vector graphics on the Web and SVG in particular takes place on the www-svg@w3.org (public mailing list of the SVG Working Group - list archives). To subscribe send an email to `www-svg-request@w3.org` with the word `subscribe` in the subject line.

The authors of this document are the SVG Working Group members (see author list). The editor is Jon Ferraiolo. The W3C staff contact for work on SVG is Dean Jackson.

Please report errors in this document to www-svg@w3.org. The list of known errors in this specification is available at http://www.w3.org/2001/09/REC-SVG-20010904-errata.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

## Available languages

The English version of this specification is the only normative version. However, for translations in other languages see http://www.w3.org/Graphics/SVG/svg-updates/translations.html.

## Table of Contents

---

The authors of the SVG 1.0 specification are the people who participated in the SVG 1.0 Working Group as members or alternates.

**Authors:**


John Bowler, Microsoft Corporation <johnbo@microsoft.com>
Craig Brown, Canon <cmb@research.canon.com.au>
Milt Capsimalis, Autodesk Inc. <milt@autodesk.com>
Richard Cohn, Adobe Systems Incorporated <richard@covero.com>
Lee Cole, Quark <lcole@quark.com>
Thomas E Deweese, Kodak <thomas.deweese@kodak.com>
David Dodds, Lexica <ddodds@lexica.net>
Andrew Donoho, IBM <awd@us.ibm.com>
David Duce, Oxford Brookes University <daduce@brookes.ac.uk>
Jerry Evans, Sun Microsystems <jerry.evans@Eng.sun.com>
Jon Ferraiolo, Adobe Systems Incorporated <jferraio@adobe.com>
Jun Fujisawa, Canon <fujisawa.jun@canon.co.jp>
Scott Furman, Netscape Communications Corporation <fur@netscape.com>
Brent Getlin, Macromedia <bgetlin@macromedia.com>
Peter Graffagnino, Apple <pgraff@apple.com>
Rick Graham, BitFlash Inc. <rick@bitflash.com>
Vincent Hardy, Sun Microsystems, <vincent.hardy@sun.com>
Lofton Henderson, OASIS, <lofton@rockynet.com>
Jan Christian Herlitz , Excosoft, <J-C.Herlitz@excosoft.se>
Alan Hester, Xerox Corporation <Alan.Hester@usa.xerox.com>
Bob Hopgood, RAL (CCLRC) <frah@inf.rl.ac.uk>
Dean Jackson, CSIRO <dean@w3.org>
Christophe Jolif, ILOG <jolif@ilog.fr>
Kelvin Lawrence, IBM <klawrenc@us.ibm.com>
Håkon Lie, Opera <howcome@operasoftware.com >
Chris Lilley, W3C <chris@w3.org>
Philip Mansfield, IntraNet Solutions, Inc. <philipm@schemasoft.com>
Kevin McCluskey, Netscape Communications Corporation <kmcclusk@netscape.com>
Tuan Nguyen, Microsoft Corporation <tuann@microsoft.com>
Troy Sandal, Visio Corporation <TroyS@visio.com>
Peter Santangeli, Macromedia <psantangeli@macromedia.com>
Haroon Sheikh, Corel Corporation <haroons@corel.ca>
Gavriel State, Corel Corporation <gavriels@COREL.CA>

Robert Stevahn, Hewlett-Packard Company <rstevahn@boi.hp.com>
Timothy Thompson, Kodak <timothy.thompson@kodak.com>
Rick Yardumian, Canon <richard.yardumian@cis.canon.com>
Shenxue Zhou, Quark <szhou@quark.com>

## Acknowledgments

The SVG Working Group would like to acknowledge the great many people outside of the SVG 1.0 Working Group who helped with the process of developing the SVG 1.0 specification. These people are too numerous to list individually. They include but are not limited to the early implementers of the SVG 1.0 language (including viewers, authoring tools, and server-side transcoders), developers of SVG content, people who have contributed on the www-svg@w3.org and svg-developers@yahoogroups.com email lists, other Working Groups at the W3C, and the W3C team. SVG 1.0 was truly a cooperative effort between the SVG Working Group, the rest of the W3C, and the public and benefited greatly from the pioneering work of early implementers and content developers, feedback from the public, and help from the W3C team.

---



---

04 September 2001

# Expanded Table of Contents

---

04 September 2001

# Copyright Notice



## W3C Document Copyright Notice and License

**Note:** This section is a copy of the W3C Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-documents-19990405.

2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

## W3C Software Copyright Notice and License

**Note:** This section is a copy of the W3C Software Copyright Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

**Copyright © 1994-2001 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.**

**http://www.w3.org/Consortium/Legal/**

---

04 September 2001

# 1 Introduction

## Contents



## 1.1 About SVG

This specification defines the features and syntax for Scalable Vector Graphics (SVG).

SVG is a language for describing two-dimensional graphics in XML [XML10]. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting.

Sophisticated applications of SVG are possible by use of a supplemental scripting language which accesses SVG Document Object Model (DOM), which provides complete access to all elements, attributes and properties. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on XHTML and SVG elements simultaneously within the same Web page.

SVG is a language for rich graphical content. For accessibility reasons, if there is an original

source document containing higher-level structure and semantics, it is recommended that the higher-level information be made available somehow, either by making the original source document available, or making an alternative version available in an alternative format which conveys the higher-level information, or by using SVG's facilities to include the higher-level information within the SVG content. For suggested techniques in achieving greater accessibility, see Accessibility.

## 1.2 SVG MIME type, file name extension and Macintosh file type

The MIME type for SVG is "`image/svg+xml`" (see [RFC3023]). The W3C will register this MIME type around the time when SVG is approved as a W3C Recommendation.

It is recommended that SVG files have the extension "`.svg`" (all lowercase) on all platforms. It is recommended that gzip-compressed SVG files have the extension "`.svgz`" (all lowercase) on all platforms.

It is recommended that SVG files stored on Macintosh HFS file systems be given a file type of "`svg `" (all lowercase, with a space character as the fourth letter). It is recommended that gzip-compressed SVG files stored on Macintosh HFS file systems be given a file type of "`svgz`" (all lowercase).

## 1.3 SVG Namespace, Public Identifier and System Identifier

The following are the SVG 1.0 namespace, public identifier and system identifier:

**SVG Namespace:**
http://www.w3.org/2000/svg
**Public Identifier for SVG 1.0:**
PUBLIC "-//W3C//DTD SVG 1.0//EN"
**System Identifier for SVG 1.0:**
http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd

The following is an example document type declaration for an SVG document:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
                  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

## 1.4 Compatibility with Other Standards Efforts

SVG leverages and integrates with other W3C specifications and standards efforts. By leveraging and conforming to other standards, SVG becomes more powerful and makes it easier for users to learn how to incorporate SVG into their Web sites.

The following describes some of the ways in which SVG maintains compatibility with, leverages and integrates with other W3C efforts:

- SVG is an application of XML and is compatible with the "Extensible Markup Language (XML) 1.0" Recommendation [XML10]
- SVG is compatible with the "Namespaces in XML" Recommendation [XML-NS]
- SVG utilizes "XML Linking Language (XLink)" [XLINK] for URI referencing and requires support for base URI specifications defined in "XML Base" [XML-BASE].
- SVG's syntax for referencing element IDs is a compatible subset of the ID referencing syntax in "XML Pointer Language (XPointer)" [XPTR].
- SVG content can be styled by either CSS (see "Cascading Style Sheets (CSS) level 2" specification [CSS2]) or XSL (see "XSL Transformations (XSLT) Version 1.0" [XSLT]). (See Styling with CSS and Styling with XSL)
- SVG supports relevant properties and approaches common to CSS and XSL, plus selected semantics and features of CSS (see SVG's styling properties and SVG's Use of Cascading Style Sheets).
- External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0" [XML-SS].
- SVG includes a complete Document Object Model (DOM) and conforms to the "Document Object Model (DOM) level 1" Recommendation [DOM1]. The SVG DOM has a high level of compatibility and consistency with the HTML DOM that is defined in the DOM Level 1 specification. Additionally, the SVG DOM supports and incorporates many of the facilities described in "Document Object Model (DOM) level 2" [DOM2], including the CSS object model and event handling.
- SVG incorporates some features and approaches that are part of the "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification" [SMIL1], including the **'switch'** element and the **systemLanguage** attribute.
- SVG's animation features (see Animation) were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [SMIL1]. SVG's animation features incorporate and extend the general-purpose XML animation capabilities described in the "SMIL Animation" specification [SMILANIM].
- SVG has been designed to allow future versions of SMIL to use animated or static SVG content as media components.
- SVG attempts to achieve maximum compatibility with both HTML 4 [HTML4] and XHTML(tm) 1.0 [XHTML]. Many of SVG's facilities are modeled directly after HTML, including its use of CSS [CSS2], its approach to event handling, and its approach to its Document Object Model [DOM2].
- SVG is compatible with W3C work on internationalization. References (W3C and otherwise) include: [UNICODE] and [CHARMOD]. Also, see Internationalization Support.
- SVG is compatible with W3C work on Web Accessibility [WAI]. Also, see Accessibility Support.

In environments which support [DOM2] for other XML grammars (e.g., XHTML [XHTML]) and which also support SVG and the SVG DOM, a single scripting approach can be used

simultaneously for both XML documents and SVG graphics, in which case interactive and dynamic effects will be possible on multiple XML namespaces using the same set of scripts.

## 1.5 Terminology

Within this specification, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 (see [RFC2119]). However, for readability, these words do not appear in all uppercase letters in this specification.

At times, this specification recommends good practice for authors and user agents. These recommendations are not normative and conformance with this specification does not depend on their realization. These recommendations contain the expression "We recommend ...", "This specification recommends ...", or some similar wording.

## 1.6 Definitions

**basic shape**
: Standard shapes which are predefined in SVG as a convenience for common graphical operations. Specifically: **'rect'**, **'circle'**, **'ellipse'**, **'line'**, **'polyline'**, **'polygon'**.

**canvas**
: A surface onto which graphics elements are drawn, which can be real physical media such as a display or paper or an abstract surface such as a allocated region of computer memory. See the discussion of the SVG canvas in the chapter on Coordinate Systems, Transformations and Units.

**clipping path**
: A combination of **'path'**, **'text'** and basic shapes which serve as the outline of a (in the absence of anti-aliasing) 1-bit mask, where everything on the "inside" of the outline is allowed to show through but everything on the outside is masked out. See Clipping paths.

**container element**
: An element which can have graphics elements and other container elements as child elements. Specifically: **'svg'**, **'g'**, **'defs'** **'symbol'**, **'clipPath'**, **'mask'**, **'pattern'**, **'marker'**, **'a'** and **'switch'**.

**current innermost SVG document fragment**
: The XML document sub-tree which starts with the most immediate ancestor **'svg'** element of a given SVG element.

**current SVG document fragment**

The XML document sub-tree which starts with the outermost ancestor **'svg'** element of a given SVG element, with the requirement that all container elements between the outermost **'svg'** and this element are all elements in the SVG language.

**current transformation matrix (CTM)**

Transformation matrices define the mathematical mapping from one coordinate system into another using a 3x3 matrix using the equation **[x' y' 1] = [x y 1] * matrix**. The *current transformation matrix* (CTM) defines the mapping from the user coordinate system into the viewport coordinate system. See Coordinate system transformations.

**fill**

The operation of painting the interior of a shape or the interior of the character glyphs in a text string.

**font**

A font represents an organized collection of glyphs in which the various glyph representations will share a common look or styling such that, when a string of characters is rendered together, the result is highly legible, conveys a particular artistic style and provides consistent inter-character alignment and spacing.

**glyph**

A glyph represents a unit of rendered content within a font. Often, there is a one-to-one correspondence between characters to be drawn and corresponding glyphs (e.g., often, the character "A" is rendered using a single glyph), but other times multiple glyphs are used to render a single character (e.g., use of accents) or a single glyph can be used to render multiple characters (e.g., ligatures). Typically, a glyph is defined by one or more shapes such as a path, possibly with additional information such as rendering hints that help a font engine to produce legible text in small sizes.

**graphics element**

One of the element types that can cause graphics to be drawn onto the target canvas. Specifically: **'path'**, **'text'**, **'rect'**, **'circle'**, **'ellipse'**, **'line'**, **'polyline'**, **'polygon'**, **'image'** and **'use'**.

**graphics referencing element**

A graphics element which uses a reference to a different document or element as the source of its graphical content. Specifically: **'use'** and **'image'**.

**local URI reference**

A Uniform Resource Identifier [URI] that does not include an **<absoluteURI>** or **<relativeURI>** and thus represents a reference to an element within the current document. See References and the **'defs'** element.

**mask**

A container element which can contain graphics elements or other container elements which define a set of graphics that is to be used as a semi-transparent mask for compositing foreground objects into the current background. See Masks.

**non-local URI reference**

A Uniform Resource Identifier [URI] that includes an **<absoluteURI>** or **<relativeURI>** and thus (usually) represents a reference to a different document or an element within a different document. See References and the **'defs'** element.

**paint**

A paint represents a way of putting color values onto the canvas. A paint might consist of both color values and associated alpha values which control the blending of colors against already existing color values on the canvas. SVG supports three types of built-in paint: color, gradients and patterns.

**presentation attribute**

An XML attribute on an SVG element which specifies a value for a given property for that element. See Styling.

**property**

A parameter that helps specify how a document should be rendered. A complete list of SVG's properties can be found in Property Index. Properties are assigned to elements in the SVG language either by presentation attributes on elements in the SVG language or by using a styling language such as CSS [CSS2]. See Styling.

**shape**

A graphics element that is defined by some combination of straight lines and curves. Specifically: **'path'**, **'rect'**, **'circle'**, **'ellipse'**, **'line'**, **'polyline'**, **'polygon'**.

**stroke**

The operation of painting the outline of a shape or the outline of character glyphs in a text string.

**SVG canvas**

The canvas onto which the SVG content is rendered. See the discussion of the SVG canvas in the chapter on Coordinate Systems, Transformations and Units.

**SVG document fragment**

The XML document sub-tree which starts with an **'svg'** element. An SVG document fragment can consist of a stand-alone SVG document, or a fragment of a parent XML document enclosed by an **'svg'** element. When an **'svg'** element is a descendant of another **'svg'** element, there are two SVG document fragments, one for each **'svg'** element. (One SVG document fragment is contained within another SVG document fragment.)

**SVG viewport**

The viewport within the SVG canvas which defines the rectangular region into which SVG content is rendered. See the discussion of the SVG viewport in the chapter on Coordinate Systems, Transformations and Units.

**text content element**

One of SVG's elements that can define a text string that is to be rendered onto the canvas. SVG's text content elements are the following: **'text'**, **'tspan'**, **'tref'**, **'textPath'** and **'altGlyph'**.

**transformation**

A modification of the current transformation matrix (CTM) by providing a supplemental transformation in the form of a set of simple transformations specifications (such as scaling, rotation or translation) and/or one or more transformation matrices. See Coordinate system transformations.

**transformation matrix**

Transformation matrices define the mathematical mapping from one coordinate system into another using a 3x3 matrix using the equation **[x' y' 1] = [x y 1] * matrix**. See current transformation matrix (CTM) and Coordinate system transformations.

**URI Reference**

A Uniform Resource Identifier [URI] which serves as a reference to a file or to an element within a file. See References and the **'defs'** element.

**user agent**

The general definition of a user agent is an application that retrieves and renders Web content, including text, graphics, sounds, video, images, and other content types. A user agent may require additional user agents that handle some types of content. For instance, a browser may run a separate program or plug-in to render sound or video. User agents include graphical desktop browsers, multimedia players, text browsers, voice browsers, and assistive technologies such as screen readers, screen magnifiers, speech synthesizers, onscreen keyboards, and voice input software.

A "user agent" may or may not have the ability to retrieve and render SVG content; however, an "SVG user agent" retrieves and renders SVG content.

**user coordinate system**

In general, a coordinate system defines locations and distances on the current canvas.

The current **user coordinate system** is the coordinate system that is currently active and which is used to define how coordinates and lengths are located and computed, respectively, on the current canvas. See initial user coordinate system and Coordinate system transformations.

**user space**

A synonym for user coordinate system.

**user units**

A coordinate value or length expressed in user units represents a coordinate value or length in the current user coordinate system. Thus, 10 user units represents a length of 10 units in the current user coordinate system.

**viewport**

A rectangular region within the current canvas onto which graphics elements are to be rendered. See the discussion of the SVG viewport in the chapter on Coordinate Systems, Transformations and Units.

**viewport coordinate system**

In general, a coordinate system defines locations and distances on the current canvas. The **viewport coordinate system** is the coordinate system that is active at the start of processing of an **'svg'** element, before processing the optional **viewBox** attribute. In the case of an SVG document fragment that is embedded within a parent document which uses CSS to manage its layout, then the viewport coordinate system will have the same orientation and lengths as in CSS, with the origin at the top-left on the viewport. See The initial viewport and Establishing a new viewport.

**viewport space**

A synonym for viewport coordinate system.

**viewport units**

A coordinate value or length expressed in viewport units represents a coordinate value or length in the viewport coordinate system. Thus, 10 viewport units represents a length of 10 units in the viewport coordinate system.

---

04 September 2001

# 2 Concepts

## Contents



## 2.1 Explaining the name: SVG

SVG stands for Scalable Vector Graphics, an XML grammar for stylable graphics, usable as an XML namespace.

### Scalable

To be scalable means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size. On the Web, scalable means that a particular technology can grow to a large number of files, a large number of users, a wide variety of applications. SVG, being a graphics technology for the Web, is scalable in both senses of the word.

SVG graphics are scalable to different display resolutions, so that for example printed output uses the full resolution of the printer and can be displayed at the same size on screens of different resolutions. The same SVG graphic can be placed at different sizes on the same Web page, and re-used at different sizes on different pages. SVG graphics can be magnified to see fine detail, or to aid those with low vision.

SVG graphics are scalable because the same SVG content can be a stand-alone graphic or can be referenced or included inside other SVG graphics, thereby allowing a complex illustration to be built up in parts, perhaps by several people. The symbol, marker and font capabilities promote re-use of graphical components, maximize the advantages of HTTP caching and avoid the need for a centralized registry of approved symbols.

### Vector

Vector graphics contain geometric objects such as lines and curves. This gives greater flexibility compared to raster-only formats (such as PNG and JPEG) which have to store information for every pixel of the graphic. Typically, vector formats can also integrate raster images and can combine them with vector information such as clipping paths to produce a complete illustration; SVG is no exception.

Since all modern displays are raster-oriented, the difference between raster-only and vector graphics comes down to where they are rasterized; client side in the case of vector graphics, as opposed to already rasterized on the server. SVG gives control over the rasterization process, for example to allow anti-aliased artwork without the ugly aliasing typical of low quality vector implementations. SVG also provides client-side raster filter effects, so that moving to a vector format does not mean the loss of popular effects such as soft drop shadows.

## Graphics

Most existing XML grammars represent either textual information, or represent raw data such as financial information. They typically provide only rudimentary graphical capabilities, often less capable than the HTML 'img' element. SVG fills a gap in the market by providing a rich, structured description of vector and mixed vector/raster graphics; it can be used stand-alone, or as an XML namespace with other grammars.

## XML

XML, a W3C Recommendation for structured information exchange, has become extremely popular and is both widely and reliably implemented. By being written in XML, SVG builds on this strong foundation and gains many advantages such as a sound basis for internationalization, powerful structuring capability, an object model, and so on. By building on existing, cleanly-implemented specifications, XML-based grammars are open to implementation without a huge reverse engineering effort.

## Namespace

It is certainly useful to have a stand-alone, SVG-only viewer. But SVG is also intended to be used as one component in a multi-namespace XML application. This multiplies the power of each of the namespaces used, to allow innovative new content to be created. For example, SVG graphics may be included in a document which uses any text-oriented XML namespace - including XHTML. A scientific document, for example, might also use MathML for mathematics in the document. The combination of SVG and SMIL leads to interesting, time based, graphically rich presentations.

SVG is a good, general-purpose component for any multi-namespace grammar that needs to use graphics.

## Stylable

The advantages of style sheets in terms of presentational control, flexibility, faster download and

improved maintenance are now generally accepted, certainly for use with text. SVG extends this control to the realm of graphics.

The combination of scripting, DOM and CSS is often termed "Dynamic HTML" and is widely used for animation, interactivity and presentational effects. SVG allows the same script-based manipulation of the document tree and the style sheet.

## 2.2 Important SVG concepts

### Graphical Objects

With any XML grammar, consideration has to be given to what exactly is being modelled. For textual formats, modelling is typically at the level of paragraphs and phrases, rather than individual nouns, adverbs, or phonemes. Similarly, SVG models graphics at the level of graphical objects rather than individual points.

SVG provides a general path element, which can be used to create a huge variety of graphical objects, and also provides common basic shapes such as rectangles and ellipses. These are convenient for hand coding and may be used in the same ways as the more general path element. SVG provides fine control over the coordinate system in which graphical objects are defined and the transformations that will be applied during rendering.

### Symbols

It would have been possible to define some standard symbols that SVG would provide. But which ones? There would always be additional symbols for electronics, cartography, flowcharts, etc., that people would need that were not provided until the "next version". SVG allows users to create, re-use and share their own symbols without requiring a centralized registry. Communities of users can create and refine the symbols that they need, without having to ask a committee. Designers can be sure exactly of the graphical appearance of the symbols they use and not have to worry about unsupported symbols.

Symbols may be used at different sizes and orientations, and can be restyled to fit in with the rest of the graphical composition.

### Raster Effects

Many existing Web graphics use the filtering operations found in paint packages to create blurs, shadows, lighting effects and so on. With the client-side rasterization used with vector formats, such effects might be thought impossible. SVG allows the declarative specification of filters, either singly or in combination, which can be applied on the client side when the SVG is rendered. These are specified in such a way that the graphics are still scalable and displayable at different resolutions.

### Fonts

Graphically rich material is often highly dependent on the particular font used and the exact spacing of the glyphs. In many cases, designers convert text to outlines to avoid any font substitution problems. This means that the original text is not present and thus searchability and accessibility suffer. In response to feedback from designers, SVG includes font elements so that both text and graphical appearance are preserved.

### Animation

Animation can be produced via script-based manipulation of the document, but scripts are difficult to edit and interchange between authoring tools is harder. Again in response to feedback from the design community, SVG includes declarative animation elements which were designed collaboratively by the SVG and SYMM Working Groups. This allows the animated effects common in existing Web graphics to be expressed in SVG.

## 2.3 Options for using SVG in Web pages

There are a variety of ways in which SVG content can be included within a Web page. Here are some of the options:

- **A stand-alone SVG Web page**
  In this case, an SVG document (i.e., a Web resource whose MIME type is "`image/svg+xml`") is loaded directly into a user agent such as a Web browser. The SVG document is the Web page that is presented to the user.
- **Embedding by reference**
  In this case, a parent Web page references a separately stored SVG document and specifies that the given SVG document should be embedded as a component of the parent Web page. For HTML or XHTML, here are three options:
  - The HTML/XHTML **'img'** element is the most common method for using graphics in HTML pages. For faster display, the width and height of the image can be given as attributes. One attribute that is required is **alt**, used to give an alternate textual string for people browsing with images off, or who cannot see the images. The string cannot contain any markup. A **longdesc** attribute lets you point to a longer description - often in HTML - which can have markup and richer formatting.
  - The HTML/XHTML **'object'** element can contain other elements nested within it, unlike **'img'**, which is empty. This means that several different formats can be offered, using nested **'object'** elements, with a final textual alternative (including markup, links, etc). The outermost element which can be displayed will be used.
  - The HTML/XHTML **'applet'** element which can invoke a Java applet to view SVG content within the given Web page. These applets can do many things, but a common task is to use them to display images, particularly ones in unusual formats or which need to be presented under the control of a program for some other reason.
- **Embedding inline**
  In this case, SVG content is embedded inline directly within the parent Web page. An example is an XHTML Web page with an SVG document fragment textually included within the XHTML.

- **External link, using the HTML 'a' element**
  This allows any stand-alone SVG viewer to be used, which can (but need not) be a different program to that used to display HTML. This option typically is used for unusual image formats.
- **Referenced from a CSS2 or XSL property**
  When a user agent supports CSS-styled XML content or XSL Formatting Objects and the user agent is a Conforming SVG Viewer, then that user agent must support the ability to reference SVG resources wherever CSS or XSL properties allow for the referencing of raster images, including the ability to tile SVG graphics wherever necessary and the ability to composite the SVG into the background if it has transparent portions. Examples include the **'background-image'** and **'list-style-image'** properties that are included in both CSS and XSL.

---

04 September 2001

# 3 Rendering Model

## Contents



## 3.1 Introduction

Implementations of SVG are expected to behave as though they implement a rendering (or imaging) model corresponding to the one described in this chapter. A real implementation is not required to implement the model in this way, but the result on any device supported by the implementation shall match that described by this model.

The appendix on conformance requirements describes the extent to which an actual implementation may deviate from this description. In practice an actual implementation will deviate slightly because of limitations of the output device (e.g. only a limited range of colors might be supported) and because of practical limitations in implementing a precise mathematical model (e.g. for realistic performance curves are approximated by straight lines, the approximation need only be sufficiently precise to match the conformance requirements).

## 3.2 The painters model

SVG uses a "painters model" of rendering. Paint is applied in successive operations to the output device such that each operation paints over some area of the output device. When the area overlaps a previously painted area the new paint partially or completely obscures the old. When the paint is not completely opaque the result on the output device is defined by the (mathematical) rules for compositing described under Simple Alpha Blending.

## 3.3 Rendering Order

Elements in an SVG document fragment have an implicit drawing order, with the first elements in the SVG document fragment getting "painted" first. Subsequent elements are painted on top of previously painted elements.

## 3.4 How groups are rendered

Grouping elements such as the 'g' (see container elements) have the effect of producing a temporary separate canvas initialized to transparent black onto which child elements are painted. Upon the completion of the group, any filter effects specified for the group are applied to create a modified temporary canvas. The modified temporary canvas is composited into the background, taking into account any group-level masking and opacity settings on the group.

## 3.5 How elements are rendered

Individual graphics elements are rendered as if each graphics element represented its own group; thus, the effect is as if a temporary separate canvas is created for each graphics element. The element is first painted onto the temporary canvas (see Painting shapes and text and Painting raster images below). Then any filter effects specified for the graphics element are applied to create a modified temporary canvas. The modified temporary canvas is then composited into the background, taking into account any clipping, masking and object opacity settings on the graphics element.

## 3.6 Types of graphics elements

SVG supports three fundamental types of graphics elements that can be rendered onto the canvas:

- Shapes, which represent some combination of straight line and curves
- Text, which represents some combination of character glyphs
- Raster images, which represent an array of values that specify the paint color and opacity (often termed alpha) at a series of points on a rectangular grid. (SVG requires support for specified raster image formats under conformance requirements.)

### 3.6.1 Painting shapes and text

Shapes and text can be filled (i.e., apply paint to the interior of the shape) and stroked (i.e., apply paint along the outline of the shape). A stroke operation is centered on the outline of the object; thus, in effect, half of the paint falls on the interior of the shape and half of the paint falls outside of the shape.

For certain types of shapes, marker symbols (which themselves can consist of any combination of shapes, text and images) can be drawn at selected vertices. Each marker symbol is painted as if its graphical content were expanded into the SVG document tree just after the shape object which is using the given marker symbol. The graphical contents of a marker symbol are rendered using the same methods as graphics elements. Marker symbols are not applicable to text.

The fill is painted first, then the stroke, and then the marker symbols. The marker symbols are rendered in order along the outline of the shape, from the start of the shape to the end of the shape.

Each fill and stroke operation has its own opacity settings; thus, you can fill and/or stroke a shape with a semi-transparently drawn solid color, with different opacity values for the fill and stroke operations.

The fill and stroke operations are entirely independent painting operations; thus, if you both fill and stroke a shape, half of the stroke will be painted on top of part of the fill.

SVG supports the following built-in types of paint which can be used in fill and stroke operations:

- Solid color
- Gradients (linear and radial)
- Patterns

### 3.6.2 Painting raster images

When a raster image is rendered, the original samples are "resampled" using standard algorithms to produce samples at the positions required on the output device. Resampling requirements are discussed under conformance requirements.

## 3.7 Filtering painted regions

SVG allows any painting operation to be filtered. (See Filter Effects.)

In this case the result must be as though the paint operations had been applied to an intermediate canvas initialized to transparent black, of a size determined by the rules given in Filter Effects then filtered by the processes defined in Filter Effects.

## 3.8 Clipping, masking and object opacity

SVG allows any painting operation to be limited to a subregion of the output device by clipping and masking. This is described in Clipping, Masking and Compositing.

Clipping uses a path to define a region of the output device to which paint can be applied. Any painting operation executed within the scope of the clipping must be rendered such that only those parts of the device that fall within the clipping region are affected by the painting operation. A clipping path can be thought of as a mask wherein those pixels outside the clipping path are black with an alpha value of zero and those pixels inside the clipping path are white with an alpha value of one. "Within" is defined by the same rules used to determine the interior of a path for painting. The clipping path is typically anti-aliased on low-resolution devices (see **'shape-rendering'**). Clipping is described in Clipping paths.

Masking uses the luminance of the color channels and alpha channel in a referenced SVG element to define a supplemental set of alpha values which are multiplied to the alpha values already present in the graphics to which the mask is applied. Masking is described in Masking.

A supplemental masking operation may also be specified by applying a "global" opacity to a set of rendering operations. In this case the mask is infinite, with a color of white and an alpha channel of the given opacity value. (See **'opacity'** property.)

In all cases the SVG implementation must behave as though all painting and filtering is first performed to an intermediate canvas which has been initialized to transparent black. Then, alpha values on the intermediate canvas are multiplied by the implicit alpha values from the clipping path, the alpha values from the mask, and the alpha values from the **'opacity'** property. The resulting canvas is composited into the background using simple alpha blending. Thus if an area of the output device is painted with a group opacity of 50% using opaque red paint followed by opaque green paint the result is as though it had been painted with just 50% opaque green paint. This is because the opaque green paint completely obscures the red paint on the intermediate canvas before the intermediate as a whole is rendered onto the output device.

## 3.9 Parent Compositing

SVG document fragments can be semi-opaque. In many environments (e.g., Web browsers), the SVG document fragment has a final compositing step where the document as a whole is blended translucently into the background canvas.

---

04 September 2001

# 4 Basic Data Types and Interfaces

## Contents



## 4.1 Basic data types

The common data types for SVG's properties and attributes fall into the following categories:

- **<integer>**: An <integer> is specified as an optional sign character ('+' or '-') followed by one or more digits "0" to "9". If the sign character is not present, the number is non-negative.
  Unless stated otherwise for a particular attribute or property, the range for a <integer> encompasses (at a minimum) -2147483648 to 2147483647.
  Within the SVG DOM, an <integer> is represented as an **long** or an **SVGAnimatedInteger**.
- **<number>** (real number value): The specification of real number values is different for property values than for XML attribute values.
  - CSS2 [CSS2] states that a property value which is a <number> is specified in **decimal notation** (i.e., a **<decimal-number>**), which consists of either an <integer>, or an optional sign character followed by zero or more digits followed by a dot (.) followed by one or more digits. Thus, for conformance with CSS2, any property in SVG which accepts <number> values is specified in decimal notation only.
  - For SVG's XML attributes, to provide as much scalability in numeric values as possible, real number values can be provided either in decimal notation or in **scientific notation** (i.e., a **<scientific-number>**), which consists of a <decimal-number> immediately followed by the letter "e" or "E" immediately followed by an <integer>.

  Unless stated otherwise for a particular attribute or property, a <number> has the capacity for at least a single-precision floating point number (see [ICC32]) and has a

range (at a minimum) of -3.4e+38F to +3.4e+38F.
It is recommended that higher precision floating point storage and computation be performed on operations such as coordinate system transformations to provide the best possible precision and to prevent round-off errors.
Conforming High-Quality SVG Viewers are required to use at least double-precision floating point (see [ICC32]) for intermediate calculations on certain numerical operations.
Within the SVG DOM, a <number> is represented as a **float** or an **SVGAnimatedNumber**.

- **<length>**: A length is a distance measurement. The format of a <length> is a <number> optionally followed immediately by a unit identifier. (Note that the specification of a <number> is different for property values than for XML attribute values.)
  If the <length> is expressed as a value without a unit identifier (e.g., **48**), then the <length> represents a distance in the current user coordinate system.
  If one of the unit identifiers is provided (e.g., **12mm**), then the <length> is processed according to the description in Units.
  Percentage values (e.g., **10%**) depend on the particular property or attribute to which the percentage value has been assigned. Two common cases are: (a) when a percentage value represents a percent of the viewport (refer to the section that discusses Units in general), and (b) when a percentage value represents a percent of the bounding box on a given object (refer to the section that describes Object bounding box units).
  Within the SVG DOM, a <length> is represented as an **SVGLength** or an **SVGAnimatedLength**.
- **<coordinate>**: A <coordinate> represents a <length> in the user coordinate system that is the given distance from the origin of the user coordinate system along the relevant axis (the x-axis for X coordinates, the y-axis for Y coordinates).
  Within the SVG DOM, a <coordinate> is represented as an **SVGLength** or an **SVGAnimatedLength** since both values have the same syntax.
- **<list of xxx>** (where xxx represents a value of some type): A list consists of a separated sequence of values. The specification of lists is different for property values than for XML attribute values.
  - Lists in property values are either comma-separated, with optional white space before or after the comma, or space-separated, as specified either in the CSS2 specification (if the property is defined there) or in this specification (if the property is not defined in the CSS2 specification).
  - Unless explicitly described differently, lists within SVG's XML attributes can be either comma-separated, with optional white space before or after the comma, or white space-separated.

  White space in lists is defined as one or more of the following consecutive characters: "space" (Unicode code 32), "tab" (9), "line feed" (10), "carriage return" (13) and "form-feed" (12).
  Within the SVG DOM, a <list of xxx> is represented by various custom interfaces, such as **SVGTransformList**.
  Here is a description of the grammar for a <list of xxx>:

```
ListOfXXX:
    XXX
    | XXX comma-wsp ListOfXXX

comma-wsp:
    (wsp+ comma? wsp*) | (comma wsp*)
```

```
comma:
    ","

wsp:
    (#x20 | #x9 | #xD | #xA)
```

where XXX represents a particular type of value.

- **<number-optional-number>**: A special case of <list of xxx> where there are at least one and at most two entries in the list and the entries are of type <number>.
- **<angle>**: An angle value is a <number> optionally followed immediately with an angle unit identifier. Angle unit identifiers are:
  - **deg**: degrees
  - **grad**: grads
  - **rad**: radians

  For properties defined in [CSS2], an angle unit identifier must be provided. For SVG-specific attributes and properties, the angle unit identifier is optional. If not provided, the angle value is assumed to be in degrees.
  The corresponding SVG DOM interface definition for <angle> is an **SVGAngle** or an **SVGAnimatedAngle**.
- **<color>**: The basic type <color> is a CSS2-compatible specification for a color in the sRGB color space [SRGB]. <color> applies to SVG's use of the **'color'** property and is a component of the definitions of properties **'fill'**, **'stroke'** **'stop-color'**, **'flood-color'** and **'lighting-color'**, which also offer optional ICC-based color specifications.
  SVG supports all of the syntax alternatives for <color> defined in [CSS2-color-types], with the exception that SVG contains an expanded list of recognized color keywords names.
  A <color> is either a keyword (see Recognized color keyword names) or a numerical RGB specification.
  In addition to these color keywords, users may specify keywords that correspond to the colors used by objects in the user's environment. The normative definition of these keywords is [CSS2 system colors].
  The format of an RGB value in hexadecimal notation is a '#' immediately followed by either three or six hexadecimal characters. The three-digit RGB notation (#rgb) is converted into six-digit form (#rrggbb) by replicating digits, not by adding zeros. For example, #fb0 expands to #ffbb00. This ensures that white (#ffffff) can be specified with the short notation (#fff) and removes any dependencies on the color depth of the display. The format of an RGB value in the functional notation is 'rgb(' followed by a comma-separated list of three numerical values (either three integer values or three percentage values) followed by ')'. The integer value 255 corresponds to 100%, and to F or FF in the hexadecimal notation: rgb(255,255,255) = rgb(100%,100%,100%) = #FFF. White space characters are allowed around the numerical values. All RGB colors are specified in the sRGB color space (see [SRGB]). Using sRGB provides an unambiguous and objectively measurable definition of the color, which can be related to international standards (see [COLORIMETRY]).
  The corresponding SVG DOM interface definitions for <color> are defined in [DOM2-CSS]; in particular, see the [DOM2-CSS-RGBCOLOR]. SVG's extension to color, including the ability to specify ICC-based colors, are represented in DOM interface **SVGColor**.

- **<paint>** : The values for properties **'fill'** and **'stroke'** are specifications of the type of paint to use when filling or stroking a given graphics element. The available options and syntax for **<paint>** are described in Specifying paint.
  Within the SVG DOM, <paint> is represented as an **SVGPaint**.
- **<percentage>**: The format of a percentage value is a <number> immediately followed by a '%'. Percentage values are always relative to another value, for example a length. Each attribute or property that allows percentages also defines the reference distance measurement to which the percentage refers.
  Within the SVG DOM, a <percentage> is usually represented as an **SVGLength** or an **SVGAnimatedLength**.
- **<transform-list>** : The detailed description of the possible values for a <transform-list> are detailed in Modifying the User Coordinate System: the transform attribute.
  Within the SVG DOM, <transform-list> is represented as an **SVGTransformList** or an **SVGAnimatedTransformList**.
- **<uri>** (Uniform Resource Identifiers [URI] references): A URI is the address of a resource on the Web. For the specification of URI references in SVG, see URI references.
  Within the SVG DOM, <uri> is represented as a **DOMString** or an **SVGAnimatedString**.
- **<frequency>**: Frequency values are used with aural properties. The normative definition of frequency values can be found in [CSS2-AURAL]. A frequency value is a <number> immediately followed by a frequency unit identifier. Frequency unit identifiers are:
  - **Hz**: Hertz
  - **kHz**: kilo Hertz

  Frequency values may not be negative.
  The corresponding SVG DOM interface definitions for <frequency> are defined in [DOM2-CSS].
- **<time>**: A time value is a <number> immediately followed by a time unit identifier. Time unit identifiers are:
  - **ms**: milliseconds
  - **s**: seconds

  Time values are used in CSS properties and may not be negative.
  The corresponding SVG DOM interface definitions for <time> are defined in [DOM2-CSS].

## 4.2 Recognized color keyword names

The following is the list of recognized color keywords that can be used as a keyword value for data type <color>:

| | |
|---|---|
| aliceblue | rgb(240, 248, 255) |
| antiquewhite | rgb(250, 235, 215) |
| aqua | rgb( 0, 255, 255) |
| aquamarine | rgb(127, 255, 212) |
| azure | rgb(240, 255, 255) |
| beige | rgb(245, 245, 220) |
| bisque | rgb(255, 228, 196) |
| black | rgb( 0, 0, 0) |
| blanchedalmond | rgb(255, 235, 205) |
| blue | rgb( 0, 0, 255) |
| blueviolet | rgb(138, 43, 226) |
| brown | rgb(165, 42, 42) |
| burlywood | rgb(222, 184, 135) |
| cadetblue | rgb( 95, 158, 160) |
| chartreuse | rgb(127, 255, 0) |
| chocolate | rgb(210, 105, 30) |
| coral | rgb(255, 127, 80) |
| cornflowerblue | rgb(100, 149, 237) |
| cornsilk | rgb(255, 248, 220) |
| crimson | rgb(220, 20, 60) |
| cyan | rgb( 0, 255, 255) |
| darkblue | rgb( 0, 0, 139) |
| darkcyan | rgb( 0, 139, 139) |
| darkgoldenrod | rgb(184, 134, 11) |
| darkgray | rgb(169, 169, 169) |
| darkgreen | rgb( 0, 100, 0) |
| darkgrey | rgb(169, 169, 169) |
| darkkhaki | rgb(189, 183, 107) |
| darkmagenta | rgb(139, 0, 139) |
| darkolivegreen | rgb( 85, 107, 47) |
| darkorange | rgb(255, 140, 0) |
| darkorchid | rgb(153, 50, 204) |
| darkred | rgb(139, 0, 0) |
| darksalmon | rgb(233, 150, 122) |
| lightpink | rgb(255, 182, 193) |
| lightsalmon | rgb(255, 160, 122) |
| lightseagreen | rgb( 32, 178, 170) |
| lightskyblue | rgb(135, 206, 250) |
| lightslategray | rgb(119, 136, 153) |
| lightslategrey | rgb(119, 136, 153) |
| lightsteelblue | rgb(176, 196, 222) |
| lightyellow | rgb(255, 255, 224) |
| lime | rgb( 0, 255, 0) |
| limegreen | rgb( 50, 205, 50) |
| linen | rgb(250, 240, 230) |
| magenta | rgb(255, 0, 255) |
| maroon | rgb(128, 0, 0) |
| mediumaquamarine | rgb(102, 205, 170) |
| mediumblue | rgb( 0, 0, 205) |
| mediumorchid | rgb(186, 85, 211) |
| mediumpurple | rgb(147, 112, 219) |
| mediumseagreen | rgb( 60, 179, 113) |
| mediumslateblue | rgb(123, 104, 238) |
| mediumspringgreen | rgb( 0, 250, 154) |
| mediumturquoise | rgb( 72, 209, 204) |
| mediumvioletred | rgb(199, 21, 133) |
| midnightblue | rgb( 25, 25, 112) |
| mintcream | rgb(245, 255, 250) |
| mistyrose | rgb(255, 228, 225) |
| moccasin | rgb(255, 228, 181) |
| navajowhite | rgb(255, 222, 173) |
| navy | rgb( 0, 0, 128) |
| oldlace | rgb(253, 245, 230) |
| olive | rgb(128, 128, 0) |
| olivedrab | rgb(107, 142, 35) |
| orange | rgb(255, 165, 0) |
| orangered | rgb(255, 69, 0) |

| | |
|---|---|
| darkseagreen | rgb(143, 188, 143) |
| darkslateblue | rgb( 72, 61, 139) |
| darkslategray | rgb( 47, 79, 79) |
| darkslategrey | rgb( 47, 79, 79) |
| darkturquoise | rgb( 0, 206, 209) |
| darkviolet | rgb(148, 0, 211) |
| deeppink | rgb(255, 20, 147) |
| deepskyblue | rgb( 0, 191, 255) |
| dimgray | rgb(105, 105, 105) |
| dimgrey | rgb(105, 105, 105) |
| dodgerblue | rgb( 30, 144, 255) |
| firebrick | rgb(178, 34, 34) |
| floralwhite | rgb(255, 250, 240) |
| forestgreen | rgb( 34, 139, 34) |
| fuchsia | rgb(255, 0, 255) |
| gainsboro | rgb(220, 220, 220) |
| ghostwhite | rgb(248, 248, 255) |
| gold | rgb(255, 215, 0) |
| goldenrod | rgb(218, 165, 32) |
| gray | rgb(128, 128, 128) |
| grey | rgb(128, 128, 128) |
| green | rgb( 0, 128, 0) |
| greenyellow | rgb(173, 255, 47) |
| honeydew | rgb(240, 255, 240) |
| hotpink | rgb(255, 105, 180) |
| indianred | rgb(205, 92, 92) |
| indigo | rgb( 75, 0, 130) |
| ivory | rgb(255, 255, 240) |
| khaki | rgb(240, 230, 140) |
| lavender | rgb(230, 230, 250) |
| lavenderblush | rgb(255, 240, 245) |
| lawngreen | rgb(124, 252, 0) |
| lemonchiffon | rgb(255, 250, 205) |
| lightblue | rgb(173, 216, 230) |
| orchid | rgb(218, 112, 214) |
| palegoldenrod | rgb(238, 232, 170) |
| palegreen | rgb(152, 251, 152) |
| paleturquoise | rgb(175, 238, 238) |
| palevioletred | rgb(219, 112, 147) |
| papayawhip | rgb(255, 239, 213) |
| peachpuff | rgb(255, 218, 185) |
| peru | rgb(205, 133, 63) |
| pink | rgb(255, 192, 203) |
| plum | rgb(221, 160, 221) |
| powderblue | rgb(176, 224, 230) |
| purple | rgb(128, 0, 128) |
| red | rgb(255, 0, 0) |
| rosybrown | rgb(188, 143, 143) |
| royalblue | rgb( 65, 105, 225) |
| saddlebrown | rgb(139, 69, 19) |
| salmon | rgb(250, 128, 114) |
| sandybrown | rgb(244, 164, 96) |
| seagreen | rgb( 46, 139, 87) |
| seashell | rgb(255, 245, 238) |
| sienna | rgb(160, 82, 45) |
| silver | rgb(192, 192, 192) |
| skyblue | rgb(135, 206, 235) |
| slateblue | rgb(106, 90, 205) |
| slategray | rgb(112, 128, 144) |
| slategrey | rgb(112, 128, 144) |
| snow | rgb(255, 250, 250) |
| springgreen | rgb( 0, 255, 127) |
| steelblue | rgb( 70, 130, 180) |
| tan | rgb(210, 180, 140) |
| teal | rgb( 0, 128, 128) |
| thistle | rgb(216, 191, 216) |
| tomato | rgb(255, 99, 71) |
| turquoise | rgb( 64, 224, 208) |

| | | | | | |
|---|---|---|---|---|---|
| **lightcoral** | rgb(240, 128, 128) | | **violet** | rgb(238, 130, 238) |
| **lightcyan** | rgb(224, 255, 255) | | **wheat** | rgb(245, 222, 179) |
| **lightgoldenrodyellow** | rgb(250, 250, 210) | | **white** | rgb(255, 255, 255) |
| **lightgray** | rgb(211, 211, 211) | | **whitesmoke** | rgb(245, 245, 245) |
| **lightgreen** | rgb(144, 238, 144) | | **yellow** | rgb(255, 255, 0) |
| **lightgrey** | rgb(211, 211, 211) | | **yellowgreen** | rgb(154, 205, 50) |

## 4.3 Basic DOM interfaces

The following interfaces are defined below: SVGElement, SVGAnimatedBoolean, SVGAnimatedString, SVGStringList, SVGAnimatedEnumeration, SVGAnimatedInteger, SVGNumber, SVGAnimatedNumber, SVGNumberList, SVGAnimatedNumberList, SVGLength, SVGAnimatedLength, SVGLengthList, SVGAnimatedLengthList, SVGAngle, SVGAnimatedAngle, SVGColor, SVGICCColor, SVGRect, SVGAnimatedRect, SVGUnitTypes, SVGStylable, SVGLocatable, SVGTransformable, SVGTests, SVGLangSpace, SVGExternalResourcesRequired, SVGFitToViewBox, SVGZoomAndPan, SVGViewSpec, SVGURIReference, SVGCSSRule, SVGRenderingIntent.

### Interface SVGElement

All of the SVG DOM interfaces that correspond directly to elements in the SVG language (e.g., the **SVGPathElement** interface corresponds directly to the **'path'** element in the language) are derivative from base class **SVGElement**.

**IDL Definition**

```
interface SVGElement : Element {
           attribute DOMString id;
                       // raises DOMException on setting
           attribute DOMString xmlbase;
                       // raises DOMException on setting
  readonly attribute SVGSVGElement ownerSVGElement;
  readonly attribute SVGElement viewportElement;
};
```

**Attributes**

**DOMString id**

The value of the **id** attribute on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString xmlbase**

Corresponds to attribute **xml:base** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**readonly SVGSVGElement ownerSVGElement**

The nearest ancestor **'svg'** element. Null if the given element is the outermost **'svg'** element.

**readonly SVGElement viewportElement**

The element which established the current viewport. Often, the nearest ancestor **'svg'** element. Null if the given element is the outermost **'svg'** element.

## Interface SVGAnimatedBoolean

Used for attributes of type boolean which can be animated.

**IDL Definition**

```
interface SVGAnimatedBoolean {

           attribute boolean baseVal;
                       // raises DOMException on setting
  readonly attribute boolean animVal;
};
```

**Attributes**

**boolean baseVal**

The base value of the given attribute before applying any animations.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**readonly boolean animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGAnimatedString

Used for attributes of type DOMString which can be animated.

**IDL Definition**

```
interface SVGAnimatedString {

           attribute DOMString baseVal;
                       // raises DOMException on setting
  readonly attribute DOMString animVal;
};
```

**Attributes**

**DOMString baseVal**

The base value of the given attribute before applying any animations.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly DOMString animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGStringList

This interface defines a list of **DOMString** objects.

**SVGStringList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

**IDL Definition**

```
interface SVGStringList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  DOMString initialize ( in DOMString newItem )
                 raises( DOMException, SVGException );
  DOMString getItem ( in unsigned long index )
                 raises( DOMException );
  DOMString insertItemBefore ( in DOMString newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  DOMString replaceItem ( in DOMString newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  DOMString removeItem ( in unsigned long index )
                 raises( DOMException );
  DOMString appendItem ( in DOMString newItem )
```

```
                    raises( DOMException, SVGException );
    };
```

**Attributes**

**readonly unsigned long numberOfItems**

The number of items in the list.

**Methods**

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter.

**Parameters**

| | |
|---|---|
| in DOMString newItem | The item which should become the only member of the list. |

**Return value**

| | |
|---|---|
| DOMString | The item being inserted into the list. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**getItem**

Returns the specified item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item from the list which is to be returned. The first item is number 0. |

**Return value**

| | |
|---|---|
| DOMString | The selected item. |

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in DOMString newItem | The item which is to be inserted into the list. |

in unsigned long index — The index of the item before which the new item is to be inserted. The first item is number 0.
If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list.

**Return value**

DOMString — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in DOMString newItem — The item which is to be inserted into the list.

in unsigned long index — The index of the item which is to be replaced. The first item is number 0.

**Return value**

DOMString — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**removeItem**

Removes an existing item from the list.

**Parameters**

in unsigned long index — The index of the item which is to be removed. The first item is number 0.

**Return value**

DOMString — The removed item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in DOMString newItem — The item which is to be inserted into the list. The first item is number 0.

**Return value**

DOMString — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

## Interface SVGAnimatedEnumeration

Used for attributes whose value must be a constant from a particular enumeration and which can be animated.

**IDL Definition**

```
interface SVGAnimatedEnumeration {

           attribute unsigned short baseVal;
                       // raises DOMException on setting
  readonly attribute unsigned short animVal;
};
```

**Attributes**

**unsigned short baseVal**

The base value of the given attribute before applying any animations.

**Exceptions on setting**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**readonly unsigned short animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGAnimatedInteger

Used for attributes of basic type 'integer' which can be animated.

**IDL Definition**

```
interface SVGAnimatedInteger {

           attribute long baseVal;
                        // raises DOMException on setting
  readonly attribute long animVal;
};
```

**Attributes**

**long baseVal**

The base value of the given attribute before applying any animations.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly long animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGNumber

Used for attributes of basic type 'number'.

**IDL Definition**

```
interface SVGNumber {

           attribute float value;
                        // raises DOMException on setting
};
```

**Attributes**

**float value**

The value of the given attribute.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Interface SVGAnimatedNumber

Used for attributes of basic type 'number' which can be animated.

**IDL Definition**

```
interface SVGAnimatedNumber {

           attribute float baseVal;
                       // raises DOMException on setting
  readonly attribute float animVal;
};
```

**Attributes**

**float baseVal**

The base value of the given attribute before applying any animations.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**readonly float animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGNumberList

This interface defines a list of **SVGNumber** objects.

**SVGNumberList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

**IDL Definition**

```
interface SVGNumberList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  SVGNumber initialize ( in SVGNumber newItem )
                 raises( DOMException, SVGException );
  SVGNumber getItem ( in unsigned long index )
                 raises( DOMException );
  SVGNumber insertItemBefore ( in SVGNumber newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGNumber replaceItem ( in SVGNumber newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGNumber removeItem ( in unsigned long index )
```

```
                    raises( DOMException );
  SVGNumber appendItem ( in SVGNumber newItem )
                    raises( DOMException, SVGException );
};
```

**Attributes**

**readonly unsigned long numberOfItems**

The number of items in the list.

**Methods**

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter.

**Parameters**

| | |
|---|---|
| in SVGNumber newItem | The item which should become the only member of the list. |

**Return value**

| | |
|---|---|
| SVGNumber | The item being inserted into the list. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**getItem**

Returns the specified item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item from the list which is to be returned. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGNumber | The selected item. |

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGNumber newItem | The item which is to be inserted into the list. |

in unsigned long index — The index of the item before which the new item is to be inserted. The first item is number 0.
If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list.

**Return value**

SVGNumber — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGNumber newItem — The item which is to be inserted into the list.

in unsigned long index — The index of the item which is to be replaced. The first item is number 0.

**Return value**

SVGNumber — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**removeItem**

Removes an existing item from the list.

**Parameters**

in unsigned long index — The index of the item which is to be removed. The first item is number 0.

**Return value**

SVGNumber — The removed item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGNumber newItem — The item which is to be inserted into the list. The first item is number 0.

**Return value**

SVGNumber — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

## Interface SVGAnimatedNumberList

Used for attributes which take a list of numbers and which can be animated.

**IDL Definition**

```
interface SVGAnimatedNumberList {

  readonly attribute SVGNumberList baseVal;
  readonly attribute SVGNumberList animVal;
};
```

**Attributes**

**readonly SVGNumberList baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGNumberList animVal**

If the given attribute or property is being animated, then this attribute contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, then this attribute contains the same value as 'baseVal'.

## Interface SVGLength

The **SVGLength** interface corresponds to the **<length>** basic data type.

**IDL Definition**

```
interface SVGLength {
```

```
    // Length Unit Types
    const unsigned short SVG_LENGTHTYPE_UNKNOWN    = 0;
    const unsigned short SVG_LENGTHTYPE_NUMBER     = 1;
    const unsigned short SVG_LENGTHTYPE_PERCENTAGE = 2;
    const unsigned short SVG_LENGTHTYPE_EMS        = 3;
    const unsigned short SVG_LENGTHTYPE_EXS        = 4;
    const unsigned short SVG_LENGTHTYPE_PX         = 5;
    const unsigned short SVG_LENGTHTYPE_CM         = 6;
    const unsigned short SVG_LENGTHTYPE_MM         = 7;
    const unsigned short SVG_LENGTHTYPE_IN         = 8;
    const unsigned short SVG_LENGTHTYPE_PT         = 9;
    const unsigned short SVG_LENGTHTYPE_PC         = 10;

    readonly attribute unsigned short unitType;
             attribute float          value;
                         // raises DOMException on setting
             attribute float          valueInSpecifiedUnits;
                         // raises DOMException on setting
             attribute DOMString      valueAsString;
                         // raises DOMException on setting

    void newValueSpecifiedUnits ( in unsigned short unitType, in float valueInSpecifiedUnits );
    void convertToSpecifiedUnits ( in unsigned short unitType );
  };
```

**Definition group Length Unit Types**

**Defined constants**

| | |
|---|---|
| SVG_LENGTHTYPE_UNKNOWN | The unit type is not one of predefined unit types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_LENGTHTYPE_NUMBER | No unit type was provided (i.e., a unitless value was specified), which indicates a value in user units. |
| SVG_LENGTHTYPE_PERCENTAGE | A percentage value was specified. |
| SVG_LENGTHTYPE_EMS | A value was specified using the "em" units defined in CSS2. |
| SVG_LENGTHTYPE_EXS | A value was specified using the "ex" units defined in CSS2. |
| SVG_LENGTHTYPE_PX | A value was specified using the "px" units defined in CSS2. |
| SVG_LENGTHTYPE_CM | A value was specified using the "cm" units defined in CSS2. |
| SVG_LENGTHTYPE_MM | A value was specified using the "mm" units defined in CSS2. |
| SVG_LENGTHTYPE_IN | A value was specified using the "in" units defined in CSS2. |
| SVG_LENGTHTYPE_PT | A value was specified using the "pt" units defined in CSS2. |
| SVG_LENGTHTYPE_PC | A value was specified using the "pc" units defined in CSS2. |

**Attributes**

**readonly unsigned short unitType**

The type of the value as specified by one of the constants specified above.

**float value**

The value as an floating point value, in user units. Setting this attribute will cause valueInSpecifiedUnits and valueAsString to be updated automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**float valueInSpecifiedUnits**

The value as an floating point value, in the units expressed by unitType. Setting this attribute will cause value and valueAsString to be updated automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**DOMString valueAsString**

The value as a string value, in the units expressed by unitType. Setting this attribute will cause value and valueInSpecifiedUnits to be updated automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Methods

**newValueSpecifiedUnits**

Reset the value as a number with an associated unitType, thereby replacing the values for all of the attributes on the object.

**Parameters**

| | |
|---|---|
| in unsigned short unitType | The unitType for the value (e.g., SVG_LENGTHTYPE_MM). |
| in float valueInSpecifiedUnits | The new value. |

**No Return Value**

**No Exceptions**

**convertToSpecifiedUnits**

Preserve the same underlying stored value, but reset the stored unit identifier to the given unitType. Object attributes unitType, valueAsSpecified and valueAsString might be modified as a result of this method. For example, if the original value were "0.5cm" and the method was invoked to convert to millimeters, then the unitType would be changed to SVG_LENGTHTYPE_MM, valueAsSpecified would be changed to the numeric value 5 and valueAsString would be changed to "5mm".

**Parameters**

| | |
|---|---|
| in unsigned short unitType | The unitType to switch to (e.g., SVG_LENGTHTYPE_MM). |

**No Return Value**

**No Exceptions**

## Interface SVGAnimatedLength

Used for attributes of basic type 'length' which can be animated.

### IDL Definition

```
interface SVGAnimatedLength {

  readonly attribute SVGLength baseVal;
  readonly attribute SVGLength animVal;
};
```

### Attributes

**readonly SVGLength baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGLength animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGLengthList

This interface defines a list of **SVGLength** objects.

**SVGLengthList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

### IDL Definition

```
interface SVGLengthList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  SVGLength initialize ( in SVGLength newItem )
                 raises( DOMException, SVGException );
  SVGLength getItem ( in unsigned long index )
                 raises( DOMException );
  SVGLength insertItemBefore ( in SVGLength newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGLength replaceItem ( in SVGLength newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGLength removeItem ( in unsigned long index )
                 raises( DOMException );
  SVGLength appendItem ( in SVGLength newItem )
```

```
                    raises( DOMException, SVGException );
    };
```

**Attributes**

**readonly unsigned long numberOfItems**

The number of items in the list.

**Methods**

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter.

**Parameters**

| | |
|---|---|
| in SVGLength newItem | The item which should become the only member of the list. |

**Return value**

| | |
|---|---|
| SVGLength | The item being inserted into the list. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**getItem**

Returns the specified item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item from the list which is to be returned. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGLength | The selected item. |

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGLength newItem | The item which is to be inserted into the list. |

in unsigned long index — The index of the item before which the new item is to be inserted. The first item is number 0.
If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list.

**Return value**

SVGLength — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGLength newItem — The item which is to be inserted into the list.

in unsigned long index — The index of the item which is to be replaced. The first item is number 0.

**Return value**

SVGLength — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**removeItem**

Removes an existing item from the list.

**Parameters**

in unsigned long index — The index of the item which is to be removed. The first item is number 0.

**Return value**

SVGLength — The removed item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGLength newItem — The item which is to be inserted into the list. The first item is number 0.

**Return value**

SVGLength — The inserted item.

**Exceptions**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException — SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

## Interface SVGAnimatedLengthList

Used for attributes of type SVGLengthList which can be animated.

**IDL Definition**

```
interface SVGAnimatedLengthList {

  readonly attribute SVGLengthList baseVal;
  readonly attribute SVGLengthList animVal;
};
```

**Attributes**

**readonly SVGLengthList baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGLengthList animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGAngle

The **SVGAngle** interface corresponds to the **<angle>** basic data type.

**IDL Definition**

```
interface SVGAngle {
```

```
    // Angle Unit Types
    const unsigned short SVG_ANGLETYPE_UNKNOWN     = 0;
    const unsigned short SVG_ANGLETYPE_UNSPECIFIED = 1;
    const unsigned short SVG_ANGLETYPE_DEG         = 2;
    const unsigned short SVG_ANGLETYPE_RAD         = 3;
    const unsigned short SVG_ANGLETYPE_GRAD        = 4;

    readonly attribute unsigned short unitType;
             attribute float          value;
                         // raises DOMException on setting
             attribute float          valueInSpecifiedUnits;
                         // raises DOMException on setting
             attribute DOMString      valueAsString;
                         // raises DOMException on setting

    void newValueSpecifiedUnits ( in unsigned short unitType, in float valueInSpecifiedUnits );
    void convertToSpecifiedUnits ( in unsigned short unitType );
  };
```

**Definition group Angle Unit Types**

**Defined constants**

| | |
|---|---|
| SVG_ANGLETYPE_UNKNOWN | The unit type is not one of predefined unit types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_ANGLETYPE_UNSPECIFIED | No unit type was provided (i.e., a unitless value was specified). For angles, a unitless value is treated the same as if degrees were specified. |
| SVG_ANGLETYPE_DEG | The unit type was explicitly set to degrees. |
| SVG_ANGLETYPE_RAD | The unit type is radians. |
| SVG_ANGLETYPE_GRAD | The unit type is grads. |

**Attributes**

**readonly unsigned short unitType**

The type of the value as specified by one of the constants specified above.

**float value**

The angle value as a floating point value, in degrees. Setting this attribute will cause valueInSpecifiedUnits and valueAsString to be updated automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**float valueInSpecifiedUnits**

The angle value as a floating point value, in the units expressed by unitType. Setting this attribute will cause value and valueAsString to be updated automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**DOMString valueAsString**

The angle value as a string value, in the units expressed by unitType. Setting this attribute will cause value and valueInSpecifiedUnits to be updated

automatically to reflect this setting.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**Methods**

**newValueSpecifiedUnits**

Reset the value as a number with an associated unitType, thereby replacing the values for all of the attributes on the object.

**Parameters**

| | |
|---|---|
| in unsigned short unitType | The unitType for the angle value (e.g., SVG_ANGLETYPE_DEG). |
| in float valueInSpecifiedUnits | The angle value. |

**No Return Value**

**No Exceptions**

**convertToSpecifiedUnits**

Preserve the same underlying stored value, but reset the stored unit identifier to the given unitType. Object attributes unitType, valueAsSpecified and valueAsString might be modified as a result of this method.

**Parameters**

| | |
|---|---|
| in unsigned short unitType | The unitType to switch to (e.g., SVG_ANGLETYPE_DEG). |

**No Return Value**

**No Exceptions**

## Interface SVGAnimatedAngle

Corresponds to all properties and attributes whose values can be basic type 'angle' and which are animatable.

**IDL Definition**

```
interface SVGAnimatedAngle {

  readonly attribute SVGAngle baseVal;
  readonly attribute SVGAngle animVal;
};
```

**Attributes**

**readonly SVGAngle baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGAngle animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being

animated, contains the same value as 'baseVal'.

## Interface SVGColor

The **SVGColor** interface corresponds to color value definition for properties **'stop-color'**, **'flood-color'** and **'lighting-color'** and is a base class for interface **SVGPaint**. It incorporates SVG's extended notion of color, which incorporates ICC-based color specifications.

Interface **SVGColor** does *not* correspond to the **<color>** basic data type. For the **<color>** basic data type, the applicable DOM interfaces are defined in [DOM2-CSS]; in particular, see the [DOM2-CSS-RGBCOLOR].

### IDL Definition

```
interface SVGColor : css::CSSValue {
  // Color Types
  const unsigned short SVG_COLORTYPE_UNKNOWN           = 0;
  const unsigned short SVG_COLORTYPE_RGBCOLOR          = 1;
  const unsigned short SVG_COLORTYPE_RGBCOLOR_ICCCOLOR = 2;
  const unsigned short SVG_COLORTYPE_CURRENTCOLOR      = 3;

  readonly attribute unsigned short colorType;
  readonly attribute css::RGBColor  rgbColor;
  readonly attribute SVGICCColor    iccColor;

  void        setRGBColor ( in DOMString rgbColor )
                  raises( SVGException );
  void        setRGBColorICCColor ( in DOMString rgbColor, in DOMString iccColor )
                  raises( SVGException );
  void        setColor ( in unsigned short colorType, in DOMString rgbColor, in DOMString iccColor )
                  raises( SVGException );
};
```

### Definition group Color Types

#### Defined constants

| | |
|---|---|
| SVG_COLORTYPE_UNKNOWN | The color type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_COLORTYPE_RGBCOLOR | An sRGB color has been specified without an alternative ICC color specification. |

| | |
|---|---|
| SVG_COLORTYPE_RGBCOLOR_ICCCOLOR | An sRGB color has been specified along with an alternate ICC color specification. |
| SVG_COLORTYPE_CURRENTCOLOR | Corresponds to when keyword 'currentColor' has been specified. |

**Attributes**

**readonly unsigned short colorType**

The type of the value as specified by one of the constants specified above.

**readonly css::RGBColor rgbColor**

The color specified in the sRGB color space.

**readonly SVGICCColor iccColor**

The alternate ICC color specification.

**Methods**

**setRGBColor**

Modifies the color value to be the specified sRGB color without an alternate ICC color specification.

**Parameters**

| | |
|---|---|
| in DOMString rgbColor | The new color value. |

**No Return Value**

**Exceptions**

| | |
|---|---|
| SVGException | SVG_INVALID_VALUE_ERR: Raised if one of the parameters has an invalid value. |

**setRGBColorICCColor**

Modifies the color value to be the specified sRGB color with an alternate ICC color specification.

**Parameters**

| | |
|---|---|
| in DOMString rgbColor | The new color value. |
| in DOMString iccColor | The alternate ICC color specification. |

**No Return Value**

**Exceptions**

| | |
|---|---|
| SVGException | SVG_INVALID_VALUE_ERR: Raised if one of the parameters has an invalid value. |

**setColor**

Sets the colorType as specified by the parameters. If `colorType` requires an RGBColor, then `rgbColor` must be a valid RGBColor object; otherwise, `rgbColor` must be null. If `colorType` requires an SVGICCColor, then `iccColor` must be a valid SVGICCColor object; otherwise, `iccColor` must be null.

**Parameters**

| | |
|---|---|
| in unsigned short colorType | One of the defined constants for colorType. |
| in DOMString rgbColor | The specification of an sRGB color, or null. |
| in DOMString iccColor | The specification of an ICC color, or null. |

**No Return Value**

**Exceptions**

| | |
|---|---|
| SVGException | SVG_INVALID_VALUE_ERR: Raised if one of the parameters has an invalid value. |

## Interface SVGICCColor

The **SVGICCColor** interface expresses an ICC-based color specification.

### IDL Definition

```
interface SVGICCColor {

           attribute DOMString      colorProfile;
                       // raises DOMException on setting
  readonly attribute SVGNumberList colors;
};
```

### Attributes

**DOMString colorProfile**

The name of the color profile, which is the first parameter of an ICC color specification.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly SVGNumberList colors**

The list of color values that define this ICC color. Each color value is an arbitrary floating point number.

## Interface SVGRect

Rectangles are defined as consisting of a (x,y) coordinate pair identifying a minimum X value, a minimum Y value, and a width and height, which are usually constrained to be non-negative.

### IDL Definition

```
interface SVGRect {
```

```
            attribute float x;
                        // raises DOMException on setting
            attribute float y;
                        // raises DOMException on setting
            attribute float width;
                        // raises DOMException on setting
            attribute float height;
                        // raises DOMException on setting
};
```

**Attributes**

**float x**

Corresponds to attribute **x** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

Corresponds to attribute **y** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float width**

Corresponds to attribute **width** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float height**

Corresponds to attribute **height** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGAnimatedRect

Used for attributes of type SVGRect which can be animated.

**IDL Definition**

```
interface SVGAnimatedRect {

  readonly attribute SVGRect baseVal;
  readonly attribute SVGRect animVal;
};
```

**Attributes**

**readonly SVGRect baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGRect animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGUnitTypes

The **SVGUnitTypes** interface defines a commonly used set of constants and is a base interface used by **SVGGradientElement**, **SVGPatternElement**, **SVGClipPathElement**, **SVGMaskElement**, and **SVGFilterElement**.

**IDL Definition**

```
interface SVGUnitTypes {

  // Unit Types
  const unsigned short SVG_UNIT_TYPE_UNKNOWN           = 0;
  const unsigned short SVG_UNIT_TYPE_USERSPACEONUSE    = 1;
  const unsigned short SVG_UNIT_TYPE_OBJECTBOUNDINGBOX = 2;
};
```

**Definition group Unit Types**

**Defined constants**

| | |
|---|---|
| SVG_UNIT_TYPE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_UNIT_TYPE_USERSPACEONUSE | Corresponds to value **userSpaceOnUse**. |
| SVG_UNIT_TYPE_OBJECTBOUNDINGBOX | Corresponds to value **objectBoundingBox**. |

## Interface SVGStylable

**IDL Definition**

```
interface SVGStylable {

  readonly attribute SVGAnimatedString className;
  readonly attribute css::CSSStyleDeclaration style;

  css::CSSValue getPresentationAttribute ( in DOMString name );
};
```

**Attributes**

**readonly SVGAnimatedString className**

Corresponds to attribute **class** on the given element.

**readonly css::CSSStyleDeclaration style**

Corresponds to attribute **style** on the given element. If the user agent does not support styling with CSS, then this attribute must always have the value of null.

**Methods**

**getPresentationAttribute**

Returns the base (i.e., static) value of a given *presentation attribute* as an object of type CSSValue. The returned object is live; changes to the objects represent immediate changes to the objects to which the CSSValue is attached.

**Parameters**

in DOMString name  Retrieves a "presentation attribute" by name.

**Return value**

css::CSSValue  The static/base value of the given *presentation attribute* as a CSSValue, or NULL if the given attribute does not have a specified value.

**No Exceptions**

## Interface SVGLocatable

Interface **SVGLocatable** is for all elements which either have a **transform** attribute or don't have a **transform** attribute but whose content can have a bounding box in current user space.

**IDL Definition**

```
interface SVGLocatable {

  readonly attribute SVGElement             nearestViewportElement;
  readonly attribute SVGElement             farthestViewportElement;

  SVGRect   getBBox (  );
  SVGMatrix getCTM (  );
  SVGMatrix getScreenCTM (  );
  SVGMatrix getTransformToElement ( in SVGElement element )
```

```
                    raises( SVGException );
    };
```

**Attributes**

**readonly SVGElement nearestViewportElement**

The element which established the current viewport. Often, the nearest ancestor **'svg'** element. Null if the current element is the outermost **'svg'** element.

**readonly SVGElement farthestViewportElement**

The farthest ancestor **'svg'** element. Null if the current element is the outermost **'svg'** element.

**Methods**

**getBBox**

Returns the tight bounding box in current user space (i.e., after application of the **transform** attribute, if any) on the geometry of all contained graphics elements, exclusive of stroke-width and filter effects).

**No Parameters**

**Return value**

SVGRect An SVGRect object that defines the bounding box.

**No Exceptions**

**getCTM**

Returns the transformation matrix from current user units (i.e., after application of the **transform** attribute, if any) to the viewport coordinate system for the nearestViewportElement.

**No Parameters**

**Return value**

SVGMatrix An SVGMatrix object that defines the CTM.

**No Exceptions**

**getScreenCTM**

Returns the transformation matrix from current user units (i.e., after application of the **transform** attribute, if any) to the parent user agent's notice of a "pixel". For display devices, ideally this represents a physical screen pixel. For other devices or environments where physical pixel sizes are not known, then an algorithm similar to the CSS2 definition of a "pixel" can be used instead.

**No Parameters**

**Return value**

SVGMatrix An SVGMatrix object that defines the given transformation matrix.

**No Exceptions**

**getTransformToElement**

Returns the transformation matrix from the user coordinate system on the current element (after application of the **transform** attribute, if any) to the user coordinate system on parameter `element` (after application of its **transform** attribute, if any).

**Parameters**

in SVGElement element The target element.

**Return value**

SVGMatrix An SVGMatrix object that defines the transformation.

**Exceptions**

| | |
|---|---|
| SVGException | SVG_MATRIX_NOT_INVERTABLE: Raised if the currently defined transformation matrices make it impossible to compute the given matrix (e.g., because one of the transformations is singular). |

## Interface SVGTransformable

Interface **SVGTransformable** contains properties and methods that apply to all elements which have attribute **transform**.

### IDL Definition

```
interface SVGTransformable : SVGLocatable {
  readonly attribute SVGAnimatedTransformList transform;
};
```

### Attributes

**readonly SVGAnimatedTransformList transform**

Corresponds to attribute **transform** on the given element.

## Interface SVGTests

Interface **SVGTests** defines an interface which applies to all elements which have attributes **requiredFeatures**, **requiredExtensions** and **systemLanguage**.

### IDL Definition

```
interface SVGTests {

  readonly attribute SVGStringList requiredFeatures;
  readonly attribute SVGStringList requiredExtensions;
  readonly attribute SVGStringList systemLanguage;

  boolean hasExtension ( in DOMString extension );
};
```

### Attributes

**readonly SVGStringList requiredFeatures**

Corresponds to attribute **requiredFeatures** on the given element.

**readonly SVGStringList requiredExtensions**

Corresponds to attribute **requiredExtensions** on the given element.

**readonly SVGStringList systemLanguage**

Corresponds to attribute **systemLanguage** on the given element.

**Methods**

**hasExtension**

Returns true if the user agent supports the given extension, specified by a URI.

**Parameters**

| | | |
|---|---|---|
| in DOMString | extension | The name of the extension, expressed as a URI. |

**Return value**

| | |
|---|---|
| boolean | True or false, depending on whether the given extension is supported. |

**No Exceptions**

## Interface SVGLangSpace

Interface **SVGLangSpace** defines an interface which applies to all elements which have attributes **xml:lang** and **xml:space**.

**IDL Definition**

```
interface SVGLangSpace {

           attribute DOMString xmllang;
                       // raises DOMException on setting
           attribute DOMString xmlspace;
                       // raises DOMException on setting
};
```

**Attributes**

**DOMString xmllang**

Corresponds to attribute **xml:lang** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**DOMString xmlspace**

Corresponds to attribute **xml:space** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Interface SVGExternalResourcesRequired

Interface **SVGExternalResourcesRequired** defines an interface which applies to all elements where this element or one of its descendants can reference an external resource.

### IDL Definition

```
interface SVGExternalResourcesRequired {

  readonly attribute SVGAnimatedBoolean externalResourcesRequired;
};
```

### Attributes

**readonly SVGAnimatedBoolean externalResourcesRequired**
Corresponds to attribute **externalResourcesRequired** on the given element.

## Interface SVGFitToViewBox

Interface **SVGFitToViewBox** defines DOM attributes that apply to elements which have XML attributes **viewBox** and **preserveAspectRatio**.

### IDL Definition

```
interface SVGFitToViewBox {

  readonly attribute SVGAnimatedRect               viewBox;
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

### Attributes

**readonly SVGAnimatedRect viewBox**
Corresponds to attribute **viewBox** on the given element.
**readonly SVGAnimatedPreserveAspectRatio preserveAspectRatio**
Corresponds to attribute **preserveAspectRatio** on the given element.

## Interface SVGZoomAndPan

The **SVGZoomAndPan** interface defines attribute "zoomAndPan" and associated constants.

### IDL Definition

```
interface SVGZoomAndPan {

  // Zoom and Pan Types
  const unsigned short SVG_ZOOMANDPAN_UNKNOWN   = 0;
  const unsigned short SVG_ZOOMANDPAN_DISABLE = 1;
  const unsigned short SVG_ZOOMANDPAN_MAGNIFY = 2;

           attribute unsigned short zoomAndPan;
                       // raises DOMException on setting
};
```

### Definition group Zoom and Pan Types

#### Defined constants

| | |
|---|---|
| SVG_ZOOMANDPAN_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_ZOOMANDPAN_DISABLE | Corresponds to value **disable**. |
| SVG_ZOOMANDPAN_MAGNIFY | Corresponds to value **magnify**. |

### Attributes

#### unsigned short zoomAndPan

Corresponds to attribute **zoomAndPan** on the given element. The value must be one of the zoom and pan constants specified above.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Interface SVGViewSpec

The interface corresponds to an SVG View Specification.

### IDL Definition

```
interface SVGViewSpec :
               SVGZoomAndPan,
               SVGFitToViewBox {

  readonly attribute SVGTransformList transform;
  readonly attribute SVGElement       viewTarget;
  readonly attribute DOMString        viewBoxString;
  readonly attribute DOMString        preserveAspectRatioString;
```

```
  readonly attribute DOMString        transformString;
  readonly attribute DOMString        viewTargetString;
};
```

**Attributes**

**readonly SVGTransformList transform**
Corresponds to the **transform** setting on the SVG View Specification.

**readonly SVGElement viewTarget**
Corresponds to the **viewTarget** setting on the SVG View Specification.

**readonly DOMString viewBoxString**
Corresponds to the **viewBox** setting on the SVG View Specification.

**readonly DOMString preserveAspectRatioString**
Corresponds to the **preserveAspectRatio** setting on the SVG View Specification.

**readonly DOMString transformString**
Corresponds to the **transform** setting on the SVG View Specification.

**readonly DOMString viewTargetString**
Corresponds to the **viewTarget** setting on the SVG View Specification.

## Interface SVGURIReference

Interface **SVGURIReference** defines an interface which applies to all elements which have the collection of XLink attributes, such as xlink:href, which define a URI reference.

**IDL Definition**

```
interface SVGURIReference {

  readonly attribute SVGAnimatedString href;
};
```

**Attributes**

**readonly SVGAnimatedString href**
Corresponds to attribute **xlink:href** on the given element.

## Interface SVGCSSRule

SVG extends interface **CSSRule** with interface **SVGCSSRule** by adding an **SVGColorProfileRule** rule to allow for specification of ICC-based color.

It is likely that this extension will become part of a future version of CSS and DOM.

**IDL Definition**

```
interface SVGCSSRule : css::CSSRule {
  // Additional CSS RuleType to support ICC color specifications
  const unsigned short COLOR_PROFILE_RULE = 7;
};
```

**Definition group Additional CSS RuleType to support ICC color specifications**

**Defined constants**

| | |
|---|---|
| COLOR_PROFILE_RULE | The rule is an @color-profile. |

## Interface SVGRenderingIntent

The **SVGRenderingIntent** interface defines the enumerated list of possible values for 'rendering-intent' attributes or descriptors.

**IDL Definition**

```
interface SVGRenderingIntent {

  // Rendering Intent Types
  const unsigned short RENDERING_INTENT_UNKNOWN               = 0;
  const unsigned short RENDERING_INTENT_AUTO                  = 1;
  const unsigned short RENDERING_INTENT_PERCEPTUAL            = 2;
  const unsigned short RENDERING_INTENT_RELATIVE_COLORIMETRIC = 3;
  const unsigned short RENDERING_INTENT_SATURATION            = 4;
  const unsigned short RENDERING_INTENT_ABSOLUTE_COLORIMETRIC = 5;
};
```

**Definition group Rendering Intent Types**

**Defined constants**

| | |
|---|---|
| RENDERING_INTENT_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| RENDERING_INTENT_AUTO | Corresponds to a value of **auto**. |

| | |
|---|---|
| RENDERING_INTENT_PERCEPTUAL | Corresponds to a value of **perceptual**. |
| RENDERING_INTENT_RELATIVE_COLORIMETRIC | Corresponds to a value of **relative-colorimetric**. |
| RENDERING_INTENT_SATURATION | Corresponds to a value of **saturation**. |
| RENDERING_INTENT_ABSOLUTE_COLORIMETRIC | Corresponds to a value of **absolute-colorimetric**. |

---

04 September 2001

# 5 Document Structure

## Contents



## 5.1 Defining an SVG document fragment: the 'svg' element

### 5.1.1 Overview

An **SVG document fragment** consists of any number of SVG elements contained within an **'svg'** element.

An SVG document fragment can range from an empty fragment (i.e., no content inside of the **'svg'** element), to a very simple SVG document fragment containing a single SVG graphics element such as a **'rect'**, to a complex, deeply nested collection of container elements and graphics elements.

An SVG document fragment can stand by itself as a self-contained file or resource, in which case the SVG document fragment is an **SVG document**, or it can be embedded inline as a fragment within a parent XML document.

The following example shows simple SVG content embedded inline as a fragment within a parent XML document. Note the use

of XML namespaces to indicate that the **'svg'** and **'ellipse'** elements belong to the SVG namespace:

```
<?xml version="1.0" standalone="yes"?>
<parent xmlns="http://example.org"
       xmlns:svg="http://www.w3.org/2000/svg">
   <!-- parent contents here -->
   <svg:svg width="4cm" height="8cm">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
   </svg:svg>
   <!-- ... -->
</parent>
```

This example shows a slightly more complex (i.e., it contains multiple rectangles) stand-alone, self-contained SVG document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="4cm"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Four separate rectangles
  </desc>
    <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
    <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
    <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
    <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="3.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

**'svg'** elements can appear in the middle of SVG content. This is the mechanism by which SVG document fragments can be embedded within other SVG document fragments.

Another use for **'svg'** elements within the middle of SVG content is to establish a new viewport. (See Establishing a new viewport.)

In all cases, for compliance with the "Namespaces in XML" Recommendation [XML-NS], an SVG namespace declaration must be provided so that all SVG elements are identified as belonging to the SVG namespace. The following are possible ways to provide a namespace declaration. An **xmlns** attribute without a namespace prefix could be specified on an **'svg'** element, which means that SVG is the default namespace for all elements within the scope of the element with the **xmlns** attribute:

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

If a namespace prefix is specified on the **xmlns** attribute (e.g., `xmlns:svg="http://www.w3.org/2000/svg"`), then the corresponding namespace is not the default namespace, so an explicit namespace prefix must be assigned to the elements:

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

Namespace prefixes can be specified on ancestor elements (illustrated in the above example). For more information, refer to the "Namespaces in XML" Recommendation [XML-NS].

### 5.1.2 The 'svg' element

```
<!ENTITY % svgExt "" >
<!ELEMENT svg (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%svgExt;)* >
<!ATTLIST svg
  xmlns CDATA #FIXED "http://www.w3.org/2000/svg"
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  zoomAndPan (disable | magnify) 'magnify'
  %graphicsElementEvents;
  %documentEvents;
  version %Number; #FIXED "1.0"
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED
  contentScriptType %ContentType; "text/ecmascript"
  contentStyleType %ContentType; "text/css" >
```

*Attribute definitions:*

**`xmlns` [:*prefix*] = "resource-name"**
: Standard XML attribute for identifying an XML namespace. Refer to the "Namespaces in XML" Recommendation [XML-NS].
*Animatable: no.*

**`version` = "<number>"**
: Indicates the SVG language version to which this document fragment conforms.
For SVG 1.0, this attribute is fixed to the value "1.0". For document fragments corresponding to future versions of the specification, this attribute will be required.
*Animatable: no.*

**`x` = "<coordinate>"**
: (Has no meaning or effect on outermost **'svg'** elements.)
The x-axis coordinate of one corner of the rectangular region into which an embedded **'svg'** element is placed.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`y` = "<coordinate>"**
: (Has no meaning or effect on outermost **'svg'** elements.)
The y-axis coordinate of one corner of the rectangular region into which an embedded **'svg'** element is placed.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`width` = "<length>"**
: For outermost **'svg'** elements, the intrinsic width of the SVG document fragment. For embedded **'svg'** elements, the width of the rectangular region into which the **'svg'** element is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
If the attribute is not specified, the effect is as if a value of "100%" were specified.
*Animatable: yes.*

**`height` = "<length>"**
: For outermost **'svg'** elements, the intrinsic height of the SVG document fragment. For embedded **'svg'** elements, the height of the rectangular region into which the **'svg'** element is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
If the attribute is not specified, the effect is as if a value of "100%" were specified.

*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, %graphicsElementEvents;, %documentEvents;, %testAttrs;, externalResourcesRequired, viewBox, preserveAspectRatio, zoomAndPan, contentScriptType, contentStyleType, style, %PresentationAttributes-All;.**

If an SVG document is likely to be referenced as a component of another document, the author will often want to include a **viewBox** attribute on the outermost **'svg'** element of the referenced document. This attribute provides a convenient way to design SVG documents to scale-to-fit into an arbitrary viewport.

## 5.2 Grouping: the 'g' element

### 5.2.1 Overview

The **'g'** element is a container element for grouping together related graphics elements.

Grouping constructs, when used in conjunction with the **'desc'** and **'title'** elements, provide information about document structure and semantics. Documents that are rich in structure may be rendered graphically, as speech, or as braille, and thus promote accessibility.

A group of elements, as well as individual objects, can be given a name using the **id** attribute. Named groups are needed for several purposes such as animation and re-usable objects.

An example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="5cm"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Two groups, each of two rectangles
  </desc>
  <g id="group1" fill="red" >
    <rect x="1cm" y="1cm" width="1cm" height="1cm" />
    <rect x="3cm" y="1cm" width="1cm" height="1cm" />
  </g>
  <g id="group2" fill="blue" >
    <rect x="1cm" y="3cm" width="1cm" height="1cm" />
    <rect x="3cm" y="3cm" width="1cm" height="1cm" />
  </g>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="4.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

A **'g'** element can contain other **'g'** elements nested within it, to an arbitrary depth. Thus, the following is possible:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Groups can nest
  </desc>
  <g>
     <g>
       <g>
```

```
        </g>
      </g>
    </g>
  </svg>
```

Any element that is not contained within a 'g' is treated (at least conceptually) as if it were in its own group.

### 5.2.2 The 'g' element

```
<!ENTITY % gExt "" >
<!ELEMENT g (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%gExt;)* >
<!ATTLIST g
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;  >
```

***Attributes defined elsewhere:***

**%stdAttrs;**, **%langSpaceAttrs;**, **class**, **transform**, **%graphicsElementEvents;**, **%testAttrs;**, **externalResourcesRequired**, **style**, **%PresentationAttributes-All;**.

## 5.3 References and the 'defs' element

### 5.3.1 Overview

SVG makes extensive use of **URI references** [URI] to other objects. For example, to fill a rectangle with a linear gradient, you first define a **'linearGradient'** element and give it an ID, as in:

```
<linearGradient id="MyGradient">...</linearGradient>
```

You then reference the linear gradient as the value of the 'fill' property for the rectangle, as in:

```
<rect style="fill:url(#MyGradient)"/>
```

In SVG, the following facilities allow URI references:

- the **'a'** element
- the **'altGlyph'** element
- the **'animate'** element
- the **'animateColor'** element
- the **'animateMotion'** element
- the **'animateTransform'** element
- the **'clip-path'** property
- the **'color-profile'** element, the **'color-profile'** property and the 'src' descriptor on an @color-profile definition

- the **'cursor'** element and **'cursor'** property
- the **'feImage'** element
- the **'fill'** property
- the **'filter'** element and **'filter'** property
- the **'image'** element
- the **'linearGradient'** element
- the **'marker','marker-start','marker-mid' and 'marker-end'** properties
- the **'mask'** property
- the **'pattern'** element
- the **'radialGradient'** element
- the **'script'** element
- the **'stroke'** property
- the **'textpath'** element
- the **'tref'** element
- the **'set'** element
- the **'use'** element

URI references are defined in either of the following forms:

```
<URI-reference> = [ <absoluteURI> | <relativeURI> ] [ "#" <elementID> ]    -or-
<URI-reference> = [ <absoluteURI> | <relativeURI> ] [ "#xpointer(id(" <elementID> "))" ]
```

where **<elementID>** is the ID of the referenced element.

(Note that the two forms above (i.e., **#<elementID>** and **#xpointer(id(<elementID>))**) are formulated in syntaxes compatible with "XML Pointer Language (XPointer)" [XPTR]. These two formulations of URI references are the only XPointer formulations that are required in SVG 1.0 user agents.)

SVG supports two types of URI references:

- **local URI references**, where the URI reference does not contain an **<absoluteURI>** or **<relativeURI>** and thus only contains a fragment identifier (i.e., **#<elementID>** or **#xpointer(id<elementID>)**)
- **non-local URI references**, where the URI reference does contain an **<absoluteURI>** or **<relativeURI>**

The following rules apply to the processing of URI references:

- URI references to nodes that do not exist shall be treated as invalid references.
- URI references to elements which are inappropriate targets for the given reference shall be treated as invalid references. For example, the **'clip-path'** property can only refer to **'clipPath'** elements. The property setting **clip-path:url(#MyElement)** is an invalid reference if the referenced element is not a **'clipPath'**.

Except for the **xlink:href** attribute on the **'a'** element and for some properties will allow for backup values in case a URI reference is invalid (see **'fill'** and **'stroke'**):

- An invalid local URI reference (i.e., an invalid references to a node within the current document) represents an error (see Error processing).
- When attribute **externalResourcesRequired** has been set to **true** on the referencing element or one of its ancestors, then an unresolved external URI reference (i.e., a resource that cannot be located) represents an error (see Error processing).

It is recommended that, wherever possible, referenced elements be defined inside of a **'defs'** element. Among the elements that are always referenced: **'altGlyphDef'**, **'clipPath'**, **'cursor'**, **'filter'**, **'linearGradient'**, **'marker'**, **'mask'**, **'pattern'**, **'radialGradient'** and **'symbol'**. Defining these elements inside of a **'defs'** element promotes understandability of the SVG content and thus promotes accessibility.

### 5.3.2 URI reference attributes

A URI reference is specified within an **href** attribute in the XLink [XLINK] namespace. If the default prefix of 'xlink:' is used for attributes in the XLink namespace, then the attribute will be specified as **xlink:href**. The value of this attribute is a URI reference for the desired resource (or resource fragment).

The value of the **href** attribute must be a URI reference as defined in [RFC2396], or must result in a URI reference after the escaping procedure described below is applied. The procedure is applied when passing the URI reference to a URI resolver.

Some characters are disallowed in URI references, even if they are allowed in XML; the disallowed characters include all non-ASCII characters, plus the excluded characters listed in Section 2.4 of [RFC2396], except for the number sign (#) and percent sign (%) and the square bracket characters re-allowed in [RFC2732]. Disallowed characters must be escaped as follows:

1. Each disallowed character is converted to UTF-8 [RFC2279] as one or more bytes.
2. Any bytes corresponding to a disallowed character are escaped with the URI escaping mechanism (that is, converted to %HH, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

Because it is impractical for any application to check that a value is a URI reference, this specification follows the lead of [RFC2396] in this matter and imposes no such conformance testing requirement on SVG applications.

If the URI reference is relative, its absolute version must be computed by the method of [XML-Base] before use.

For locators into XML resources, the format of the fragment identifier (if any) used within the URI reference is specified by the XPointer specification [XPTR].

Additional XLink attributes can be specified that provide supplemental information regarding the referenced resource. These additional attributes are included in the DTD in the following entities. The two entity definitions differ only in the value of **xlink:show**, which has the value **other** in the first entity and the value **embed** in the second. The first entity definition is used in most element definitions which reference resources. The second entity definition is used by elements **'use'**, **'image'** and **'feImage'**.

```
<!ENTITY % xlinkRefAttrs
 "xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'
  xlink:type (simple) #FIXED 'simple'
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (other) 'other'
  xlink:actuate (onLoad) #FIXED 'onLoad'" >

<!ENTITY % xlinkRefAttrsEmbed
 "xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'
  xlink:type (simple) #FIXED 'simple'
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (embed) 'embed'
  xlink:actuate (onLoad) #FIXED 'onLoad'" >
```

**`xmlns` [:*prefix*] = "resource-name"**
: Standard XML attribute for identifying an XML namespace. This attribute makes the XLink [XLink] namespace available to the current element. Refer to the "Namespaces in XML" Recommendation [XML-NS].
*Animatable: no.*

**`xlink:type` = 'simple'**
: Identifies the type of XLink being used. In SVG, only simple links are available. Refer to the "XML Linking Language (XLink)" [XLink].
*Animatable: no.*

**`xlink:role` = '<uri>'**

A URI reference that identifies some resource that describes the intended property. The value must be a URI reference as defined in [RFC2396], except that if the URI scheme used is allowed to have absolute and relative forms, the URI portion must be absolute. When no value is supplied, no particular role value is to be inferred. Disallowed URI reference characters in these attribute values must be specially encoded as described earlier in this section. Refer to the "XML Linking Language (XLink)" [XLink].

*Animatable: no.*

**`xlink:arcrole` = '<uri>'**

A URI reference that identifies some resource that describes the intended property. The value must be a URI reference as defined in [RFC2396], except that if the URI scheme used is allowed to have absolute and relative forms, the URI portion must be absolute. When no value is supplied, no particular role value is to be inferred. Disallowed URI reference characters in these attribute values must be specially encoded as described earlier in this section. The arcrole attribute corresponds to the [RDF] notion of a property, where the role can be interpreted as stating that "starting-resource HAS arc-role ending-resource." This contextual role can differ from the meaning of an ending resource when taken outside the context of this particular arc. For example, a resource might generically represent a "person," but in the context of a particular arc it might have the role of "mother" and in the context of a different arc it might have the role of "daughter." Refer to the "XML Linking Language (XLink)" [XLink].

*Animatable: no.*

**`xlink:title` = '<string>'**

The title attribute is used to describe the meaning of a link or resource in a human-readable fashion, along the same lines as the role or arcrole attribute. A value is optional; if a value is supplied, it should contain a string that describes the resource. The use of this information is highly dependent on the type of processing being done. It may be used, for example, to make titles available to applications used by visually impaired users, or to create a table of links, or to present help text that appears when a user lets a mouse pointer hover over a starting resource. Refer to the "XML Linking Language (XLink)" [XLink].

*Animatable: no.*

**`xlink:show` = 'embed'**

An application traversing to the ending resource should load its presentation in place of the presentation of the starting resource. Refer to the "XML Linking Language (XLink)" [XLink].

*Animatable: no.*

**`xlink:actuate` = 'onLoad'**

Indicates that the application should traverse to the ending resource immediately on loading the starting resource. Refer to the "XML Linking Language (XLink)" [XLink].

*Animatable: no.*

In all cases, for compliance with the "Namespaces in XML" Recommendation [XML-NS], an explicit XLink namespace declaration must be provided whenever one of the above XLink attributes is used within SVG content. One simple way to provide such an XLink namespace declaration is to include an **xmlns** attribute for the XLink namespace on the outermost **'svg'** element for content that uses XLink attributes. For example:

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink"...>
  <image xlink:href="foo.png" .../>
</svg>
```

### 5.3.3 The 'defs' element

The **'defs'** element is a container element for referenced elements. For understandability and accessibility reasons, it is recommended that, whenever possible, referenced elements be defined inside of a **'defs'**.

The content model for **'defs'** is the same as for the **'g'** element; thus, any element that can be a child of a **'g'** can also be a child of a **'defs'**, and vice versa.

Elements that are descendants of a **'defs'** are not rendered directly; they are prevented from becoming part of the rendering tree just as if the **'defs'** element were a **'g'** element and the **'display'** property were set to **none**. Note, however, that the descendants of a **'defs'** are always present in the source tree and thus can always be referenced by other elements; thus, the value of the **'display'** property on the **'defs'** element or any of its descendants does not prevent those elements from being referenced by other elements.

```
<!ENTITY % defsExt "" >
<!ELEMENT defs (desc|title|metadata|defs|
                path|text|rect|circle|ellipse|line|polyline|polygon|
                use|image|svg|g|view|switch|a|altGlyphDef|
                script|style|symbol|marker|clipPath|mask|
                linearGradient|radialGradient|pattern|filter|cursor|font|
                animate|set|animateMotion|animateColor|animateTransform|
                color-profile|font-face
                %ceExt;%defsExt;)* >
<!ATTLIST defs
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;  >
```

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-All;, %graphicsElementEvents;.**

To provide some SVG user agents with an opportunity to implement efficient implementations in streaming environments, creators of SVG content are encouraged to place all elements which are targets of local URI references within a **'defs'** element which is a direct child of one of the ancestors of the referencing element. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="3cm"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Local URI references within ancestor's 'defs' element.</desc>
  <defs>
    <linearGradient id="Gradient01">
      <stop offset="20%" stop-color="#39F" />
      <stop offset="90%" stop-color="#F3F" />
    </linearGradient>
  </defs>
  <rect x="1cm" y="1cm" width="6cm" height="1cm"
        fill="url(#Gradient01)"  />

  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="7.98cm" height="2.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

In the document above, the linear gradient is defined within a **'defs'** element which is the direct child of the **'svg'** element, which in turn is an ancestor of the **'rect'** element which references the linear gradient. Thus, the above document conforms to the guideline.

## 5.4 The 'desc' and 'title' elements

Each container element or graphics element in an SVG drawing can supply a **'desc'** and/or a **'title'** description string where the description is text-only. When the current SVG document fragment is rendered as SVG on visual media, **'desc'** and **'title'** elements are not rendered as part of the graphics. User agents may, however, for example, display the **'title'** element as a tooltip, as the pointing device moves over particular elements. Alternate presentations are possible, both visual and aural, which display the **'desc'** and **'title'** elements but do not display **'path'** elements or other graphics elements. This is readily achieved by using a different (perhaps user) style sheet. For deep hierarchies, and for following **'use'** element references, it is sometimes desirable to allow the user to control how deep they drill down into descriptive text.

```
<!ENTITY % descExt "" >
<!ELEMENT desc (#PCDATA %descExt;)* >
<!ATTLIST desc
  %stdAttrs;
  %langSpaceAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %StructuredText; >
```

***Attributes defined elsewhere:***
**%stdAttrs;, %langSpaceAttrs;, class, style.**

```
<!ENTITY % titleExt "" >
<!ELEMENT title (#PCDATA %titleExt;)* >
<!ATTLIST title
  %stdAttrs;
  %langSpaceAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %StructuredText; >
```

***Attributes defined elsewhere:***
**%stdAttrs;, %langSpaceAttrs;, class, style.**

The following is an example. In typical operation, the SVG user agent would not render the **'desc'** and **'title'** elements but would render the remaining contents of the **'g'** element.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
     xmlns="http://www.w3.org/2000/svg">
<g>
  <title>
    Company sales by region
  </title>
  <desc>
    This is a bar chart which shows
    company sales by region.
  </desc>
  <!-- Bar chart defined as vector data -->
</g>
</svg>
```

Description and title elements can contain marked-up text from other namespaces. Here is an example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns="http://www.w3.org/2000/svg">
   <desc xmlns:mydoc="http://example.org/mydoc">
      <mydoc:title>This is an example SVG file</mydoc:title>
      <mydoc:para>The global description uses markup from the
        <mydoc:emph>mydoc</mydoc:emph> namespace.</mydoc:para>
   </desc>
   <g>
   <!-- the picture goes here -->
   </g>
</svg>
```

Authors should always provide a **'title'** child element to the outermost **'svg'** element within a stand-alone SVG document. The **'title'** child element to an **'svg'** element serves the purposes of identifying the content of the given SVG document fragment. Since users often consult documents out of context, authors should provide context-rich titles. Thus, instead of a title such as "Introduction", which doesn't provide much contextual background, authors should supply a title such as "Introduction to Medieval Bee-Keeping" instead. For reasons of accessibility, user agents should always make the content of the **'title'** child element to the outermost **'svg'** element available to users. The mechanism for doing so depends on the user agent (e.g., as a

caption, spoken).

The DTD definitions of many of SVG's elements (particularly, container and text elements) place no restriction on the placement or number of the **'desc'** and **'title'** sub-elements. This flexibility is only present so that there will be a consistent content model for container elements, because some container elements in SVG allow for mixed content, and because the mixed content rules for XML [XML-MIXED] do not permit the desired restrictions. Representations of future versions of the SVG language might use more expressive representations than DTDs which allow for more restrictive mixed content rules. It is strongly recommended that at most one **'desc'** and at most one **'title'** element appear as a child of any particular element, and that these elements appear before any other child elements (except possibly **'metadata'** elements) or character data content. If user agents need to choose among multiple **'desc'** or **'title'** elements for processing (e.g., to decide which string to use for a tooltip), the user agent shall choose the first one.

## 5.5 The 'symbol' element

The **'symbol'** element is used to define graphical template objects which can be instantiated by a **'use'** element.

The use of **'symbol'** elements for graphics that are used multiple times in the same document adds structure and semantics. Documents that are rich in structure may be rendered graphically, as speech, or as braille, and thus promote accessibility.

The key distinctions between a **'symbol'** and a **'g'** are:

- A **'symbol'** element itself is not rendered. Only instances of a **'symbol'** element (i.e., a reference to a **'symbol'** by a **'use'** element) are rendered.
- A **'symbol'** element has attributes **viewBox** and **preserveAspectRatio** which allow a **'symbol'** to scale-to-fit within a rectangular viewport defined by the referencing **'use'** element.

Closely related to the **'symbol'** element are the 'marker' and 'pattern' elements.

```
<!ENTITY % symbolExt "" >
<!ELEMENT symbol (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%symbolExt;)* >
<!ATTLIST symbol
  %stdAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  %graphicsElementEvents; >
```

***Attributes defined elsewhere:***

**%stdAttrs;**, **%langSpaceAttrs;**, **class**, **externalResourcesRequired**, **viewBox**, **preserveAspectRatio**, **style**, **%PresentationAttributes-All;**, **%graphicsElementEvents;**.

'symbol' elements are never rendered directly; their only usage is as something that can be referenced using the 'use' element. The **'display'** property does not apply to the 'symbol' element; thus, 'symbol' elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and 'symbol' elements are available for referencing even when the **'display'** property on the 'symbol' element or any of its ancestors is set to **none**.

## 5.6 The 'use' element

Any 'svg', 'symbol', 'g', graphics element or other 'use' is potentially a template object that can be re-used (i.e., "instanced") in the SVG document via a 'use' element. The 'use' element references another element and indicates that the graphical contents of that element is included/drawn at that given point in the document.

Unlike 'image', the 'use' element cannot reference entire files.

The 'use' element has optional attributes **x**, **y**, **width** and **height** which are used to map the graphical contents of the referenced element onto a rectangular region within the current coordinate system.

The effect of a 'use' element is as if the contents of the referenced element were deeply cloned into a separate non-exposed DOM tree which had the 'use' element as its parent and all of the 'use' element's ancestors as its higher-level ancestors. Because the cloned DOM tree is non-exposed, the SVG Document Object Model (DOM) only contains the 'use' element and its attributes. The SVG DOM does not show the referenced element's contents as children of 'use' element.

For user agents that support Styling with CSS, the conceptual deep cloning of the referenced element into a non-exposed DOM tree also copies any property values resulting from the CSS cascade [CSS2-CASCADE] on the referenced element and its contents. CSS2 selectors can be applied to the original (i.e., referenced) elements because they are part of the formal document structure. CSS2 selectors cannot be applied to the (conceptually) cloned DOM tree because its contents are not part of the formal document structure.

Property inheritance, however, works as if the referenced element had been textually included as a deeply cloned child of the 'use' element. The referenced element inherits properties from the 'use' element and the 'use' element's ancestors. An instance of a referenced element does not inherit properties from the referenced element's original parents.

If event attributes are assigned to referenced elements, then the actual target for the event will be the **SVGElementInstance** object within the "instance tree" corresponding to the given referenced element.

The behavior of the **'visibility'** property conforms to this model of property inheritance. Thus, specifying **'visibility:hidden'** on a 'use' element does not guarantee that the referenced content will not be rendered. If the 'use' element specifies **'visibility:hidden'** and the element it references specifies **'visibility:hidden'** or **'visibility:inherit'**, then that one element will be hidden. However, if the referenced element instead specifies **'visibility:visible'**, then that element will be visible even if the 'use' element specifies **'visibility:hidden'**.

Animations on a referenced element will cause the instance to also be animated.

A 'use' element has the same visual effect as if the 'use' element were replaced by the following generated content:

- **If the 'use' element references a 'symbol' element**:

  In the generated content, the 'use' will be replaced by 'g', where all attributes from the 'use' element except for **x**, **y**, **width**, **height** and **xlink:href** are transferred to the generated 'g' element. An additional transformation **translate(x,y)** is appended to the end (i.e., right-side) of the **transform** attribute on the generated 'g', where **x** and **y** represent the values of the **x** and **y** attributes on the 'use' element. The referenced 'symbol' and its contents are deep-cloned into the generated tree, with the exception that the 'symbol' is replaced by an 'svg'. This generated 'svg' will always have explicit values for attributes **width** and **height**. If attributes **width** and/or **height** are provided on the 'use' element, then these attributes will be transferred to the generated 'svg'. If attributes **width** and/or **height** are not specified, the generated 'svg' element will use values of 100% for these attributes.

- **If the 'use' element references an 'svg' element:**

  In the generated content, the 'use' will be replaced by 'g', where all attributes from the 'use' element except for **x**, **y**,

**width**, **height** and **xlink:href** are transferred to the generated **'g'** element. An additional transformation **translate(x,y)** is appended to the end (i.e., right-side) of the **transform** attribute on the generated **'g'**, where **x** and **y** represent the values of the **x** and **y** attributes on the **'use'** element. The referenced **'svg'** and its contents are deep-cloned into the generated tree. If attributes **width** and/or **height** are provided on the **'use'** element, then these values will override the corresponding attributes on the **'svg'** in the generated tree.

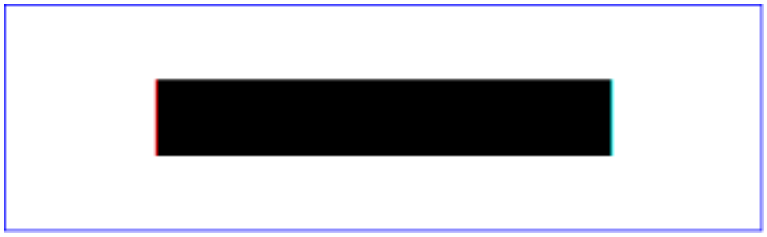
- **Otherwise:**

  In the generated content, the **'use'** will be replaced by **'g'**, where all attributes from the **'use'** element except for **x**, **y**, **width**, **height** and **xlink:href** are transferred to the generated **'g'** element. An additional transformation **translate(x,y)** is appended to the end (i.e., right-side) of the **transform** attribute on the generated **'g'**, where **x** and **y** represent the values of the **x** and **y** attributes on the **'use'** element. The referenced object and its contents are deep-cloned into the generated tree.

For user agents that support Styling with CSS, the generated **'g'** element carries along with it the "cascaded" property values on the **'use'** element which result from the CSS cascade [CSS2-CASCADE]. Additionally, the copy (deep clone) of the referenced resource carries along with it the "cascaded" property values resulting from the CSS cascade on the original (i.e., referenced) elements. Thus, the result of various CSS selectors in combination with the **class** and **style** attributes are, in effect, replaced by the functional equivalent of a **style** attribute in the generated content which conveys the "cascaded" property values.

Example Use01 below has a simple **'use'** on a **'rect'**.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example Use01 - Simple case of 'use' on a 'rect'</desc>
  <defs>
    <rect id="MyRect" width="60" height="10"/>
  </defs>
  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />
  <use x="20" y="10" xlink:href="#MyRect" />
</svg>
```



***Example Use01***

*View this example as SVG (SVG-enabled browsers only)*

The visual effect would be equivalent to the following document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Use01-GeneratedContent - Simple case of 'use' on a 'rect'</desc>
  <!-- 'defs' section left out -->

  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />

  <!-- Start of generated content. Replaces 'use' -->
```

```
  <g transform="translate(20,10)">
    <rect width="60" height="10"/>
  </g>
  <!-- End of generated content -->

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

Example Use02 below has a **'use'** on a **'symbol'**.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example Use02 - 'use' on a 'symbol'</desc>
  <defs>
    <symbol id="MySymbol" viewBox="0 0 20 20">
      <desc>MySymbol - four rectangles in a grid</desc>
      <rect x="1" y="1" width="8" height="8"/>
      <rect x="11" y="1" width="8" height="8"/>
      <rect x="1" y="11" width="8" height="8"/>
      <rect x="11" y="11" width="8" height="8"/>
    </symbol>
  </defs>
  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />
  <use x="45" y="10" width="10" height="10"
       xlink:href="#MySymbol" />
</svg>
```

***Example Use02***

*View this example as SVG (SVG-enabled browsers only)*

The visual effect would be equivalent to the following document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Use02-GeneratedContent - 'use' on a 'symbol'</desc>

  <!-- 'defs' section left out -->

  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />

  <!-- Start of generated content. Replaces 'use' -->
  <g transform="translate(45, 10)" >
    <!-- Start of referenced 'symbol'. 'symbol' replaced by 'svg',
         with x,y,width,height=0,0,100%,100% -->
    <svg width="10" height="10"
         viewBox="0 0 20 20">
      <rect x="1" y="1" width="8" height="8"/>
      <rect x="11" y="1" width="8" height="8"/>
      <rect x="1" y="11" width="8" height="8"/>
      <rect x="11" y="11" width="8" height="8"/>
```

```
    </svg>
    <!-- End of referenced symbol -->
  </g>
  <!-- End of generated content -->

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

Example Use03 illustrates what happens when a **'use'** has a **transform** attribute.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example Use03 - 'use' with a 'transform' attribute</desc>
  <defs>
    <rect id="MyRect" x="0" y="0" width="60" height="10"/>
  </defs>
  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />
  <use xlink:href="#MyRect"
       transform="translate(20,2.5) rotate(10)" />
</svg>
```

***Example Use03***

*View this example as SVG (SVG-enabled browsers only)*

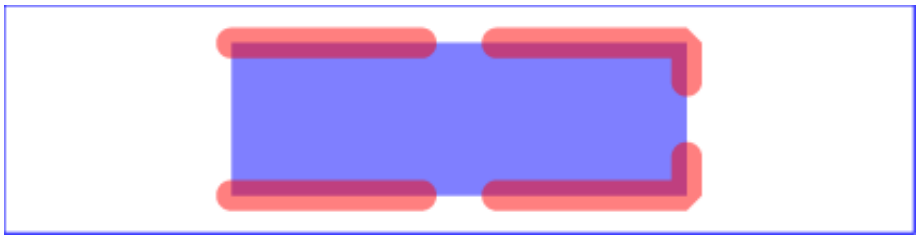
The visual effect would be equivalent to the following document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Use03-GeneratedContent - 'use' with a 'transform' attribute</desc>

  <!-- 'defs' section left out -->

  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />

  <!-- Start of generated content. Replaces 'use' -->
  <g transform="translate(20,2.5) rotate(10)">
    <rect x="0" y="0" width="60" height="10"/>
  </g>
  <!-- End of generated content -->

</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

Example Use04 illustrates a **'use'** element with various methods of applying CSS styling.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3cm" viewBox="0 0 1200 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example Use04 - 'use' with CSS styling</desc>
  <defs style=" /* rule 9 */ stroke-miterlimit: 10" >
    <path id="MyPath" d="M300 50 L900 50 L900 250 L300 250"
                      class="MyPathClass"
                      style=" /* rule 10 */ stroke-dasharray:300,100" />
  </defs>
  <style type="text/css">
    <![CDATA[
      /* rule 1 */ #MyUse { fill: blue }
      /* rule 2 */ #MyPath { stroke: red }
      /* rule 3 */ use { fill-opacity: .5 }
      /* rule 4 */ path { stroke-opacity: .5 }
      /* rule 5 */ .MyUseClass { stroke-linecap: round }
      /* rule 6 */ .MyPathClass { stroke-linejoin: bevel }
      /* rule 7 */ use > path { shape-rendering: optimizeQuality }
      /* rule 8 */ g > path { visibility: hidden }
    ]]>
  </style>

  <rect x="0" y="0" width="1200" height="300"
        style="fill:none; stroke:blue; stroke-width:3"/>
  <g style=" /* rule 11 */ stroke-width:40">
    <use id="MyUse" xlink:href="#MyPath"
         class="MyUseClass"
         style="/* rule 12 */ stroke-dashoffset:50" />
  </g>
</svg>
```



***Example Use04***

*View this example as SVG (SVG-enabled browsers only)*

The visual effect would be equivalent to the following document. Observe that some of the style rules above apply to the generated content (i.e., rules 1-6, 10-12), whereas others do not (i.e., rules 7-9). The rules which do not affect the generated content are:

- Rules 7 and 8: CSS selectors only apply to the formal document tree, not on the generated tree; thus, these selectors will not yield a match.
- Rule 9: The generated tree only inherits from the ancestors of the **'use'** element and does not inherit from the ancestors of the referenced element; thus, this rule does not affect the generated content.

In the generated content below, the selectors that yield a match have been transferred into inline 'style' attributes for illustrative purposes.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3cm" viewBox="0 0 1200 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Use04-GeneratedContent - 'use' with a 'transform' attribute</desc>

  <!-- 'style' and 'defs' sections left out -->

  <rect x="0" y="0" width="1200" height="300"
        style="fill:none; stroke:blue; stroke-width:3"/>
  <g style="/* rule 11 */ stroke-width:40">
```

```
          <!-- Start of generated content. Replaces 'use' -->
          <g style="/* rule 1 */ fill:blue;
                      /* rule 3 */ fill-opacity:.5;
                      /* rule 5 */ stroke-linecap:round;
                      /* rule 12 */ stroke-dashoffset:50" >
            <path d="M300 50 L900 50 L900 250 L300 250"
                  style="/* rule 2 */ stroke:red;
                         /* rule 4 */ stroke-opacity:.5;
                         /* rule 6 */ stroke-linejoin: bevel;
                         /* rule 10 */ stroke-dasharray:300,100" />
          </g>
          <!-- End of generated content -->

        </g>
      </svg>
```

*View this example as SVG (SVG-enabled browsers only)*

When a **'use'** references another element which is another **'use'** or whose content contains a **'use'** element, then the deep cloning approach described above is recursive.

```
<!ENTITY % useExt "" >
<!ELEMENT use (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                  %geExt;%useExt;)*) >
<!ATTLIST use
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >
```

*Attribute definitions:*

**x = "<coordinate>"**
: The x-axis coordinate of one corner of the rectangular region into which the referenced element is placed. If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>"**
: The y-axis coordinate of one corner of the rectangular region into which the referenced element is placed. If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**width = "<length>"**
: The width of the rectangular region into which the referenced element is placed.
*Animatable: yes.*

**height = "<length>"**
: The height of the rectangular region into which the referenced element is placed.
*Animatable: yes.*

**xlink:href = "<uri>"**
: A URI reference to an element/fragment within an SVG document.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;**, **%xlinkRefAttrsEmbed;**, **%testAttrs;**, **%langSpaceAttrs;**, **externalResourcesRequired**, **class**, **style**, **%PresentationAttributes-All;**, **transform**, **%graphicsElementEvents;**.

## 5.7 The 'image' element

The **'image'** element indicates that the contents of a complete file are to be rendered into a given rectangle within the current user coordinate system. The **'image'** element can refer to raster image files such as PNG or JPEG or to files with MIME type of "image/svg+xml". Conforming SVG viewers need to support at least PNG, JPEG and SVG format files.

The result of processing an **'image'** is always a four-channel RGBA result. When an **'image'** element references a raster image file such as PNG or JPEG files which only has three channels (RGB), then the effect is as if the object were converted into a 4-channel RGBA image with the alpha channel uniformly set to 1. For a single-channel raster image, the effect is as if the object were converted into a 4-channel RGBA image, where the single channel from the referenced object is used to compute the three color channels and the alpha channel is uniformly set to 1.

When an **'image'** element references a raster image file such as PNG or JPEG files, then the raster image is fitted into the region specified by the **x**, **y**, **width** and **height** attribute. Attribute **preserveAspectRatio** determines both the size and aspect ratio of the raster when fitted into the region specified by **x**, **y**, **width** and **height**. For example, if **preserveAspectRatio="xMinYMin meet"**, then the aspect ratio of the raster would be preserved (which means that the scale factor from image's coordinates to current user space coordinates would be the same for both X and Y), the raster would be sized as large as possible while ensuring that the entire raster fits within the viewport, and the top/left of the raster would be aligned with the top/left of the viewport. If **preserveAspectRatio="none"**, then the aspect ratio of the raster would not be preserved. The image would be fitted such that the top/left corner of the raster exactly aligns with coordinate (**x**,**y**) and the bottom/right corner of the raster exactly aligns with coordinate (**x**+**width**,**y**+**height**).

When an **'image'** element references an SVG file, then the **'image'** element establishes a new viewport for the SVG file as described in Establishing a new viewport. The bounds for the new viewport are defined by attributes **x**, **y**, **width** and **height**. Except for the implicit coordinate system translation that may occur due to the processing of the **x** and **y** attributes (see establishing a new viewport), the **'image'** element itself does not cause any coordinate system transformations. Thus, the initial coordinate system for the referenced SVG file will be identical to the coordinate system for the new viewport. If it is necessary for the referenced SVG file to be scaled to fit into the viewport established by the **'image'** element, then that outermost **'svg'** element on the referenced SVG file will need to have a **viewBox** attribute or the referencing file will need to include appropriate transformations, perhaps by including an appropriate **transform** attribute on the **'image'** element or one of its ancestors or by placing the **'image'** element within an **'svg'** element which has an appropriate **viewBox** attribute.

The resource referenced by the **'image'** element represents a separate document which generates its own parse tree and document object model (if the resource is XML). Thus, there is no inheritance of properties into the image.

Unlike **'use'**, the **'image'** element cannot reference elements within an SVG file.

```
<!ENTITY % imageExt "" >
<!ELEMENT image (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                  %geExt;%imageExt;)*) >
<!ATTLIST image
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Images;
  %PresentationAttributes-Viewports;
  transform %TransformList; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED >
```

*Attribute definitions:*

**x = "<coordinate>"**
: The x-axis coordinate of one corner of the rectangular region into which the referenced document is placed.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>"**
: The y-axis coordinate of one corner of the rectangular region into which the referenced document is placed.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**width = "<length>"**
: The width of the rectangular region into which the referenced document is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

**height = "<length>"**
: The height of the rectangular region into which the referenced document is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

**xlink:href = "<uri>"**
: A URI reference.
*Animatable: yes.*

***Attributes defined elsewhere:***
: **%stdAttrs;**, **%xlinkRefAttrsEmbed;**, **%testAttrs;**, **%langSpaceAttrs;**, **externalResourcesRequired**, **class**, **style**, **%PresentationAttributes-Color;**, **%PresentationAttributes-Graphics;**, **%PresentationAttributes-Images;**, **%PresentationAttributes-Viewports;**, **transform**, **preserveAspectRatio**, **%graphicsElementEvents;**.

An example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>This graphic links to an external image
  </desc>
```

```
    <image x="200" y="200" width="100px" height="100px"
           xlink:href="myimage.png">
      <title>My image</title>
    </image>
  </svg>
```

## 5.8 Conditional processing

### 5.8.1 Conditional processing overview

SVG contains a **'switch'** element along with attributes **requiredFeatures**, **requiredExtensions** and **systemLanguage** to provide an ability to specify alternate viewing depending on the capabilities of a given user agent or the user's language.

```
<!ENTITY % testAttrs
 "requiredFeatures %FeatureList; #IMPLIED
  requiredExtensions %ExtensionList; #IMPLIED
  systemLanguage %LanguageCodes; #IMPLIED" >
```

Attributes **requiredFeatures**, **requiredExtensions** and **systemLanguage** act as tests and return either true or false results. The **'switch'** renders the first of its children for which all of these attributes test true. If the given attribute is not specified, then a true value is assumed.

### 5.8.2 The 'switch' element

The **'switch'** element evaluates the **requiredFeatures**, **requiredExtensions** and **systemLanguage** attributes on its direct child elements in order, and then processes and renders the first child for which these attributes evaluate to true. All others will be bypassed and therefore not rendered. If the child element is a container element such as a **'g'**, then the entire subtree is either processed/rendered or bypassed/not rendered.

Note that the values of properties **'display'** and **'visibility'** have no effect on **'switch'** element processing. In particular, setting **'display'** to **none** on a child of a **'switch'** element has no effect on true/false testing associated with **'switch'** element processing.

```
<!ENTITY % switchExt "" >
<!ELEMENT switch (%descTitleMetadata;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|foreignObject|
                   animate|set|animateMotion|animateColor|animateTransform
                   %ceExt;%switchExt;)*) >
<!ATTLIST switch
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents; >
```

***Attributes defined elsewhere:***

**%stdAttrs;**, **%langSpaceAttrs;**, **class**, **transform**, **%graphicsElementEvents;**, **%testAttrs;**, **externalResourcesRequired**, **style**, **%PresentationAttributes-All;**.

For more information and an example, see Embedding foreign object types.

### 5.8.3 The requiredFeatures attribute

Definition of **`requiredFeatures`**:

**`requiredFeatures`** = ***list-of-features***

The value is a list of feature strings, with the individual values separated by white space. Determines whether all of the named *features* are supported by the user agent. Only feature strings defined in this section (see below) are allowed. If all of the given features are supported, then the attribute evaluates to true; otherwise, the current element and its children are skipped and thus will not be rendered.
*Animatable: no.*

All feature strings referring to language capabilities begin with **"org.w3c.svg"**. All feature strings referring to SVG DOM capabilities begin with **"org.w3c.dom.svg"**.

The following are the feature strings for the **requiredFeatures** attribute. These same feature strings apply to the hasFeature method call that is part of the SVG DOM's support for the **DOMImplementation** interface defined in [DOM2-CORE] (see Feature strings for the **hasFeature** method call).

- The feature string **"org.w3c.svg"** indicates that the user agent supports at least one of the following (all of which are described subsequently): **"org.w3c.svg.static"**, **"org.w3c.svg.animation"**, **"org.w3c.svg.dynamic"** or **"org.w3c.dom.svg"**. (Because the feature string **"org.w3c.svg"** can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)
- The feature string **"org.w3c.dom.svg"** indicates that the user agent supports at least one of the following (all of which are described subsequently): **"org.w3c.dom.svg.static"**, **"org.w3c.dom.svg.animation"** or **"org.w3c.dom.svg.dynamic"**. (Because the feature string **"org.w3c.dom.svg"** can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)
- The feature string **"org.w3c.svg.static"** indicates the availability of all of the language capabilities defined in:
  - Basic Data Types and Interfaces
  - Document Structure
  - Styling
  - Coordinate Systems, Transformations and Units
  - Paths
  - Basic Shapes
  - Text
  - Painting: Filling, Stroking and Marker Symbols
  - Color
  - Gradients and Patterns
  - Clipping, Masking and Compositing
  - Filter Effects
  - Fonts
  - The **'switch'** element
  - The **requiredFeatures** attribute
  - The **requiredExtensions** attribute
  - The **systemLanguage** attribute

  For SVG viewers, **"org.w3c.svg.static"** indicates that the viewer can process and render successfully all of the language features listed above.
- The feature string **"org.w3c.dom.svg.static"** indicates the availability of all of the DOM interfaces and methods that correspond to the language features for **"org.w3c.svg.static"**.
- The feature string **"org.w3c.svg.animation"** includes all of the language capabilities defined for **"org.w3c.svg.static"** plus the availability of all of the language capabilities defined in Animation. For SVG viewers running on media capable of rendering time-based material, such as displays, **"org.w3c.svg.animation"** indicates that the viewer can process and render successfully all of the corresponding language features.
- The feature string **"org.w3c.dom.svg.animation"** corresponds to the availability of DOM interfaces and methods that

correspond to the language features for **"org.w3c.svg.animation"**.
- The feature string **"org.w3c.svg.dynamic"** includes all of the language capabilities defined for **"org.w3c.svg.animation"** plus the availability of all of the language capabilities defined in Relationship with DOM2 events, Linking and Interactivity and Scripting. For SVG viewers running on media capable of rendering time-based material, such as displays, **"org.w3c.svg.dynamic"** indicates that the viewer can process and render successfully all of the corresponding language features.
- The feature string **"org.w3c.dom.svg.dynamic"** corresponds to the availability of DOM interfaces and methods that correspond to the language features for **"org.w3c.svg.dynamic"**.
- The feature string **"org.w3c.svg.all"** corresponds to the availability of all of the language capabilities defined in this specification.
- The feature string **"org.w3c.dom.svg.all"** corresponds to the availability of all of the DOM interfaces defined in this specification.

If the attribute is not present, then its implicit return value is "true". If a null string or empty string value is given to attribute **requiredFeatures**, the attribute returns "false".

**requiredFeatures** is often used in conjunction with the **'switch'** element. If the **requiredFeatures** is used in other situations, then it represents a simple switch on the given element whether to render the element or not.

### 5.8.4 The requiredExtensions attribute

The **requiredExtensions** attribute defines a list of required language extensions. Language extensions are capabilities within a user agent that go beyond the feature set defined in this specification. Each extension is identified by a URI reference.

Definition of **`requiredExtensions`**:

**`requiredExtensions`** = ***list-of-extensions***
: The value is a list of URI references which identify the required extensions, with the individual values separated by white space. Determines whether all of the named *extensions* are supported by the user agent. If all of the given extensions are supported, then the attribute evaluates to true; otherwise, the current element and its children are skipped and thus will not be rendered.
*Animatable: no.*

If a given URI reference contains white space within itself, that white space must be escaped.

If the attribute is not present, then its implicit return value is "true". If a null string or empty string value is given to attribute **requiredExtensions**, the attribute returns "false".

**requiredExtensions** is often used in conjunction with the **'switch'** element. If the **requiredExtensions** is used in other situations, then it represents a simple switch on the given element whether to render the element or not.

The URI names for the extension should include versioning information, such as "http://example.org/SVGExtensionXYZ/1.0", so that script writers can distinguish between different versions of a given extension.

### 5.8.5 The systemLanguage attribute

The attribute value is a comma-separated list of language names as defined in [RFC3066].

Evaluates to "true" if one of the languages indicated by user preferences exactly equals one of the languages given in the value of this parameter, or if one of the languages indicated by user preferences exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-".

Evaluates to "false" otherwise.

Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags

for which this tag is a prefix.

The prefix rule simply allows the use of prefix tags if this is the case.

Implementation note: When making the choice of linguistic preference available to the user, implementers should take into account the fact that users are not familiar with the details of language matching as described above, and should provide appropriate guidance. As an example, users may assume that on selecting "en-gb", they will be served any kind of English document if British English is not available. The user interface for setting user preferences should guide the user to add "en" to get the best matching behavior.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, content that is presented simultaneously in the original Maori and English versions, would call for:

```
<text systemLanguage="mi, en"><!-- content goes here --></text>
```

However, just because multiple languages are present within the object on which the **systemLanguage** test attribute is placed, this does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-literate audience. In this case, the **systemLanguage** test attribute should only include "en".

Authoring note: Authors should realize that if several alternative language objects are enclosed in a **'switch'**, and none of them matches, this may lead to situations where no content is displayed. It is thus recommended to include a "catch-all" choice at the end of such a **'switch'** which is acceptable in all cases.

For the **systemLanguage** attribute: *Animatable: no.*

If the attribute is not present, then its implicit return value is "true". If a null string or empty string value is given to attribute **systemLanguage**, the attribute returns "false".

**systemLanguage** is often used in conjunction with the **'switch'** element. If the **systemLanguage** is used in other situations, then it represents a simple switch on the given element whether to render the element or not.

## 5.9 Specifying whether external resources are required for proper rendering

Documents often reference and use the contents of other files (and other Web resources) as part of their rendering. In some cases, authors want to specify that particular resources are required for a document to be considered correct.

Attribute **externalResourcesRequired** is available on all container elements and to all elements which potentially can reference external resources. It specifies whether referenced resources that are not part of the current document are required for proper rendering of the given container element or graphics element.

Attribute definition:

**`externalResourcesRequired` = "false | true"**

**false**
: (The default value.) Indicates that resources external to the current document are optional. Document rendering can proceed even if external resources are unavailable to the current element and its descendants.

**true**
: Indicates that resources external to the current document are required. If an external resource is not available, progressive rendering is suspended until that resource and all other required resources become available, have been parsed and are ready to be rendered. If a timeout event occurs on a required resource, then the document goes into an error state (see Error processing). The document remains in an error state until all required resources become available.

This attribute applies to all types of resource references, including style sheets, color profiles (see Color profile descriptions) and fonts specified by a URI reference using a **'font-face'** element or a CSS @font-face specification. In particular, if an

element sets **externalResourcesRequired="true"**, then all style sheets must be available since any style sheet might affect the rendering of that element.

Attribute **externalResourcesRequired** is not inheritable (from a sense of attribute value inheritance), but if set on a container element, its value will apply to all elements within the container.

Because setting **externalResourcesRequired="true"** on a container element can have the effect of disabling progressive display of the contents of that container, tools that generate SVG content are cautioned against using simply setting **externalResourcesRequired="true"** on the outermost **'svg'** element on a universal basis. Instead, it is better to specify **externalResourcesRequired="true"** on those particular graphics elements or container elements which specify need the availability of external resources in order to render properly.

For **externalResourcesRequired**: *Animatable: no.*

## 5.10 Common attributes

### 5.10.1 Attributes common to all elements: id and xml:base

The **id** and **xml:base** attributes are available on all SVG elements:

```
<!ENTITY % stdAttrs
 "id ID #IMPLIED
  xml:base %URI; #IMPLIED" >
```

*Attribute definitions:*

**`id` = "name"**
: Standard XML attribute for assigning a unique *name* to an element. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10].
*Animatable: no.*

**`xml:base` = "<uri>"**
: Specifies a base URI other than the base URI of the document or external entity. Refer to the "XML Base" specification [XML-BASE].
*Animatable: no.*

### 5.10.2 The xml:lang and xml:space attributes

Elements that might contain character data content have attributes **xml:lang** and **xml:space**:

```
<!ENTITY % langSpaceAttrs
 "xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED" >
```

*Attribute definitions:*

**`xml:lang` = "languageID"**
: Standard XML attribute to specify the language (e.g., English) used in the contents and attribute values of particular elements. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10].

*Animatable: no.*

**xml:space = "{default | preserve}"**

Standard XML attribute to specify whether white space is preserved in character data. The only possible values are *default* and *preserve*. Refer to the "Extensible Markup Language (XML) 1.0" Recommendation [XML10] and to the discussion white space handling in SVG.

*Animatable: no.*

## 5.11 DOM interfaces

The following interfaces are defined below: SVGDocument, SVGSVGElement, SVGGElement, SVGDefsElement, SVGDescElement, SVGTitleElement, SVGSymbolElement, SVGUseElement, SVGElementInstance, SVGElementInstanceList, SVGImageElement, SVGSwitchElement, GetSVGDocument.

### Interface SVGDocument

When an **'svg'** element is embedded inline as a component of a document from another namespace, such as when an **'svg'** element is embedded inline within an XHTML document [XHTML], then an **SVGDocument** object will not exist; instead, the root object in the document object hierarchy will be a Document object of a different type, such as an HTMLDocument object.

However, an **SVGDocument** object will indeed exist when the root element of the XML document hierarchy is an **'svg'** element, such as when viewing a stand-alone SVG file (i.e., a file with MIME type "image/svg+xml"). In this case, the **SVGDocument** object will be the root object of the document object model hierarchy.

In the case where an SVG document is embedded by reference, such as when an XHTML document has an **'object'** element whose **href** attribute references an SVG document (i.e., a document whose MIME type is "image/svg+xml" and whose root element is thus an **'svg'** element), there will exist two distinct DOM hierarchies. The first DOM hierarchy will be for the referencing document (e.g., an XHTML document). The second DOM hierarchy will be for the referenced SVG document. In this second DOM hierarchy, the root object of the document object model hierarchy is an **SVGDocument** object.

The **SVGDocument** interface contains a similar list of attributes and methods to the HTMLDocument interface described in the Document Object Model (HTML) Level 1 chapter of the [DOM1] specification.

**IDL Definition**

```
interface SVGDocument :
               Document,
               events::DocumentEvent {

  readonly attribute DOMString    title;
  readonly attribute DOMString     referrer;
  readonly attribute DOMString      domain;
  readonly attribute DOMString      URL;
  readonly attribute SVGSVGElement rootElement;
};
```

**Attributes**

**readonly DOMString title**

The title of a document as specified by the title sub-element of the **'svg'** root element (i.e., **<svg><title>Here is the title</title>...</svg>**)

**readonly DOMString referrer**

Returns the URI of the page that linked to this page. The value is an empty string if the user navigated to the page directly (not through a link, but, for example, via a bookmark).

**readonly DOMString domain**

The domain name of the server that served the document, or a null string if the server cannot be identified by a domain name.

**readonly DOMString URL**
The complete URI of the document.
**readonly SVGSVGElement rootElement**
The root **'svg'** element in the document hierarchy.

## Interface SVGSVGElement

A key interface definition is the **SVGSVGElement** interface, which is the interface that corresponds to the **'svg'** element. This interface contains various miscellaneous commonly-used utility methods, such as matrix operations and the ability to control the time of redraw on visual rendering devices.

**SVGSVGElement** extends **ViewCSS** and **DocumentCSS** to provide access to the computed values of properties and the override style sheet as described in DOM2.

**IDL Definition**

```
interface SVGSVGElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGLocatable,
                SVGFitToViewBox,
                SVGZoomAndPan,
                events::EventTarget,
                events::DocumentEvent,
                css::ViewCSS,
                css::DocumentCSS {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
           attribute DOMString         contentScriptType;
                       // raises DOMException on setting
           attribute DOMString         contentStyleType;
                       // raises DOMException on setting
  readonly attribute SVGRect           viewport;
  readonly attribute float pixelUnitToMillimeterX;
  readonly attribute float pixelUnitToMillimeterY;
  readonly attribute float screenPixelToMillimeterX;
  readonly attribute float screenPixelToMillimeterY;
           attribute boolean useCurrentView;
                       // raises DOMException on setting
  readonly attribute SVGViewSpec currentView;
           attribute float currentScale;
                       // raises DOMException on setting
  readonly attribute SVGPoint currentTranslate;

  unsigned long suspendRedraw ( in unsigned long max_wait_milliseconds );
  void          unsuspendRedraw ( in unsigned long suspend_handle_id )
                  raises( DOMException );
  void          unsuspendRedrawAll (  );
  void          forceRedraw (  );
  void          pauseAnimations (  );
  void          unpauseAnimations (  );
  boolean       animationsPaused (  );
  float         getCurrentTime (  );
  void          setCurrentTime ( in float seconds );
  NodeList      getIntersectionList ( in SVGRect rect, in SVGElement referenceElement );
  NodeList      getEnclosureList ( in SVGRect rect, in SVGElement referenceElement );
  boolean       checkIntersection ( in SVGElement element, in SVGRect rect );
  boolean       checkEnclosure ( in SVGElement element, in SVGRect rect );
  void          deselectAll (  );
  SVGNumber               createSVGNumber (  );
  SVGLength               createSVGLength (  );
  SVGAngle                createSVGAngle (  );
  SVGPoint                createSVGPoint (  );
  SVGMatrix               createSVGMatrix (  );
  SVGRect                 createSVGRect (  );
  SVGTransform            createSVGTransform (  );
  SVGTransform     createSVGTransformFromMatrix ( in SVGMatrix matrix );
  Element         getElementById ( in DOMString elementId );
};
```

**Attributes**
**readonly SVGAnimatedLength x**
Corresponds to attribute **x** on the given **'svg'** element.

**readonly SVGAnimatedLength y**
Corresponds to attribute **y** on the given **'svg'** element.

**readonly SVGAnimatedLength width**
Corresponds to attribute **width** on the given **'svg'** element.

**readonly SVGAnimatedLength height**
Corresponds to attribute **height** on the given **'svg'** element.

**DOMString contentScriptType**
Corresponds to attribute **contentScriptType** on the given **'svg'** element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**DOMString contentStyleType**
Corresponds to attribute **contentStyleType** on the given **'svg'** element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly SVGRect viewport**

The position and size of the viewport (implicit or explicit) that corresponds to this **'svg'** element. When the user agent is actually rendering the content, then the position and size values represent the actual values when rendering. The position and size values are unitless values in the coordinate system of the parent element. If no parent element exists (i.e., **'svg'** element represents the root of the document tree), if this SVG document is embedded as part of another document (e.g., via the HTML **'object'** element), then the position and size are unitless values in the coordinate system of the parent document. (If the parent uses CSS or XSL layout, then unitless values represent pixel units for the current CSS or XSL viewport, as described in the CSS2 specification.) If the parent element does not have a coordinate system, then the user agent should provide reasonable default values for this attribute.

The object itself and its contents are both readonly.

**readonly float pixelUnitToMillimeterX**
Size of a pixel units (as defined by CSS2) along the x-axis of the viewport, which represents a unit somewhere in the range of 70dpi to 120dpi, and, on systems that support this, might actually match the characteristics of the target medium. On systems where it is impossible to know the size of a pixel, a suitable default pixel size is provided.

**readonly float pixelUnitToMillimeterY**
Corresponding size of a pixel unit along the y-axis of the viewport.

**readonly float screenPixelToMillimeterX**
User interface (UI) events in DOM Level 2 indicate the screen positions at which the given UI event occurred. When the user agent actually knows the physical size of a "screen unit", this attribute will express that information; otherwise, user agents will provide a suitable default value such as .28mm.

**readonly float screenPixelToMillimeterY**
Corresponding size of a screen pixel along the y-axis of the viewport.

**boolean useCurrentView**
The initial view (i.e., before magnification and panning) of the current innermost SVG document fragment can be either the "standard" view (i.e., based on attributes on the **'svg'** element such as **fitBoxToViewport**) or to a "custom" view (i.e., a hyperlink into a particular **'view'** or other element - see Linking into SVG content: URI fragments and SVG views). If the initial view is the "standard" view, then this attribute is false. If the initial view is a "custom" view, then this attribute is true.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly SVGViewSpec currentView**
The definition of the initial view (i.e., before magnification and panning) of the current innermost SVG document fragment. The meaning depends on the situation:

- If the initial view was a "standard" view, then:
  - the values for viewBox, preserveAspectRatio and zoomAndPan within currentView will match the values for the corresponding DOM attributes that are on SVGSVGElement directly
  - the values for transform and viewTarget within currentView will be null
- If the initial view was a link into a **'view'** element, then:
  - the values for viewBox, preserveAspectRatio and zoomAndPan within currentView will correspond to the corresponding attributes for the given **'view'** element
  - the values for transform and viewTarget within currentView will be null

- If the initial view was a link into another element (i.e., other than a **'view'**), then:
  - the values for viewBox, preserveAspectRatio and zoomAndPan within currentView will match the values for the corresponding DOM attributes that are on SVGSVGElement directly for the closest ancestor **'svg'** element
  - the values for transform within currentView will be null
  - the viewTarget within currentView will represent the target of the link
- If the initial view was a link into the SVG document fragment using an SVG view specification fragment identifier (i.e., #svgView(...)), then:
  - the values for viewBox, preserveAspectRatio, zoomAndPan, transform and viewTarget within currentView will correspond to the values from the SVG view specification fragment identifier

The object itself and its contents are both readonly.

**float currentScale**

This attribute indicates the current scale factor relative to the initial view to take into account user magnification and panning operations, as described under Magnification and panning. DOM attributes currentScale and currentTranslate are equivalent to the 2x3 matrix [a b c d e f] = [currentScale 0 0 currentScale currentTranslate.x currentTranslate.y]. If "magnification" is enabled (i.e., zoomAndPan="magnify"), then the effect is as if an extra transformation were placed at the outermost level on the SVG document fragment (i.e., outside the outermost **'svg'** element).

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**readonly SVGPoint currentTranslate**

The corresponding translation factor that takes into account user "magnification".

**Methods**

**suspendRedraw**

Takes a time-out value which indicates that redraw shall not occur until: (a) the corresponding unsuspendRedraw(suspend_handle_id) call has been made, (b) an unsuspendRedrawAll() call has been made, or (c) its timer has timed out. In environments that do not support interactivity (e.g., print media), then redraw shall not be suspended. suspend_handle_id = suspendRedraw(max_wait_milliseconds) and unsuspendRedraw(suspend_handle_id) must be packaged as balanced pairs. When you want to suspend redraw actions as a collection of SVG DOM changes occur, then precede the changes to the SVG DOM with a method call similar to suspend_handle_id = suspendRedraw(max_wait_milliseconds) and follow the changes with a method call similar to unsuspendRedraw(suspend_handle_id). Note that multiple suspendRedraw calls can be used at once and that each such method call is treated independently of the other suspendRedraw method calls.

**Parameters**

| | |
|---|---|
| in unsigned long max_wait_milliseconds | The amount of time in milliseconds to hold off before redrawing the device. Values greater than 60 seconds will be truncated down to 60 seconds. |

**Return value**

| | |
|---|---|
| unsigned long | A number which acts as a unique identifier for the given suspendRedraw() call. This value must be passed as the parameter to the corresponding unsuspendRedraw() method call. |

**No Exceptions**

**unsuspendRedraw**

Cancels a specified suspendRedraw() by providing a unique suspend_handle_id.

**Parameters**

| | |
|---|---|
| in unsigned long suspend_handle_id | A number which acts as a unique identifier for the desired suspendRedraw() call. The number supplied must be a value returned from a previous call to suspendRedraw() |

**No Return Value**

**Exceptions**

| | |
|---|---|
| DOMException | This method will raise a DOMException with value **NOT_FOUND_ERR** if an invalid value (i.e., no such suspend_handle_id is active) for suspend_handle_id is provided. |

**unsuspendRedrawAll**

Cancels all currently active suspendRedraw() method calls. This method is most useful at the very end of a set of SVG DOM calls to ensure that all pending suspendRedraw() method calls have been cancelled.

**No Parameters**

**No Return Value**

**No Exceptions**

**forceRedraw**

In rendering environments supporting interactivity, forces the user agent to immediately redraw all regions of the viewport that require updating.

**No Parameters**

**No Return Value**

**No Exceptions**

**pauseAnimations**

Suspends (i.e., pauses) all currently running animations that are defined within the SVG document fragment corresponding to this **'svg'** element, causing the animation clock corresponding to this document fragment to stand still until it is unpaused.

**No Parameters**

**No Return Value**

**No Exceptions**

**unpauseAnimations**

Unsuspends (i.e., unpauses) currently running animations that are defined within the SVG document fragment, causing the animation clock to continue from the time at which it was suspended.

**No Parameters**

**No Return Value**

**No Exceptions**

**animationsPaused**

Returns true if this SVG document fragment is in a paused state.

**No Parameters**

**Return value**

boolean Boolean indicating whether this SVG document fragment is in a paused state.

**No Exceptions**

**getCurrentTime**

Returns the current time in seconds relative to the start time for the current SVG document fragment.

**No Parameters**

**Return value**

float The current time in seconds.

**No Exceptions**

**setCurrentTime**

Adjusts the clock for this SVG document fragment, establishing a new current time.

**Parameters**

in float seconds The new current time in seconds relative to the start time for the current SVG document fragment.

**No Return Value**

**No Exceptions**

**getIntersectionList**

Returns the list of graphics elements whose rendered content intersects the supplied rectangle, honoring the 'pointer-events' property value on each candidate graphics element.

**Parameters**

in SVGRect rect The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

in SVGElement referenceElement If not null, then only return elements whose drawing order has them below the given reference element.

**Return value**

NodeList A list of Elements whose content intersects the supplied rectangle.

**No Exceptions**

**getEnclosureList**

Returns the list of graphics elements whose rendered content is entirely contained within the supplied rectangle, honoring the 'pointer-events' property value on each candidate graphics element.

**Parameters**

in SVGRect rect The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

in SVGElement referenceElement If not null, then only return elements whose drawing order has them below the given reference element.

**Return value**

NodeList A list of Elements whose content is enclosed by the supplied rectangle.

**No Exceptions**

**checkIntersection**

Returns true if the rendered content of the given element intersects the supplied rectangle, honoring the 'pointer-events' property value on each candidate graphics element.

**Parameters**

in SVGElement element The element on which to perform the given test.

in SVGRect rect The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

**Return value**

boolean True or false, depending on whether the given element intersects the supplied rectangle.

**No Exceptions**

**checkEnclosure**

Returns true if the rendered content of the given element is entirely contained within the supplied rectangle, honoring the 'pointer-events' property value on each candidate graphics element.

**Parameters**

in SVGElement element The element on which to perform the given test.

in SVGRect rect The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

**Return value**

boolean True or false, depending on whether the given element is enclosed by the supplied rectangle.

**No Exceptions**

**deselectAll**

Unselects any selected objects, including any selections of text strings and type-in bars.

**No Parameters**

**No Return Value**

**No Exceptions**

**createSVGNumber**

Creates an SVGNumber object outside of any document trees. The object is initialized to a value of zero.

**No Parameters**

**Return value**

SVGNumber An SVGNumber object.

**No Exceptions**

**createSVGLength**

Creates an SVGLength object outside of any document trees. The object is initialized to the value of 0 user units.

**No Parameters**

**Return value**

SVGLength An SVGLength object.

**No Exceptions**

**createSVGAngle**

Creates an SVGAngle object outside of any document trees. The object is initialized to the value 0 degrees (unitless).

**No Parameters**

**Return value**

SVGAngle An SVGAngle object.

**No Exceptions**

**createSVGPoint**

Creates an SVGPoint object outside of any document trees. The object is initialized to the point (0,0) in the user coordinate system.

**No Parameters**

**Return value**

SVGPoint An SVGPoint object.

**No Exceptions**

**createSVGMatrix**

Creates an SVGMatrix object outside of any document trees. The object is initialized to the identity matrix.

**No Parameters**

**Return value**

SVGMatrix An SVGMatrix object.

**No Exceptions**

**createSVGRect**

Creates an SVGRect object outside of any document trees. The object is initialized such that all values are set to 0 user units.

**No Parameters**

**Return value**

SVGRect An SVGRect object.

**No Exceptions**

**createSVGTransform**

Creates an SVGTransform object outside of any document trees. The object is initialized to an identity matrix transform (SVG_TRANSFORM_MATRIX).

**No Parameters**

**Return value**

SVGTransform An SVGTransform object.

**No Exceptions**

**createSVGTransformFromMatrix**

Creates an SVGTransform object outside of any document trees. The object is initialized to the given matrix transform (i.e., SVG_TRANSFORM_MATRIX).

**Parameters**

in SVGMatrix matrix The transform matrix.

**Return value**

SVGTransform An SVGTransform object.

**No Exceptions**

**getElementById**

Searches this SVG document fragment (i.e., the search is restricted to a subset of the document tree) for an Element whose **id** is given by elementId. If an Element is found, that Element is returned. If no such element exists, returns null. Behavior is not defined if more than one element has this **id**.

**Parameters**

in DOMString elementId The unique **id** value for an element.

**Return value**

Element The matching element.

**No Exceptions**

## Interface SVGGElement

The **SVGGElement** interface corresponds to the **'g'** element.

**IDL Definition**

```
interface SVGGElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {};
```

## Interface SVGDefsElement

The **SVGDefsElement** interface corresponds to the **'defs'** element.

**IDL Definition**

```
interface SVGDefsElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
```

```
                SVGStylable,
                SVGTransformable,
                events::EventTarget {};
```

## Interface SVGDescElement

The **SVGDescElement** interface corresponds to the **'desc'** element.

**IDL Definition**

```
interface SVGDescElement :
                SVGElement,
                SVGLangSpace,
                SVGStylable {};
```

## Interface SVGTitleElement

The **SVGTitleElement** interface corresponds to the **'title'** element.

**IDL Definition**

```
interface SVGTitleElement :
                SVGElement,
                SVGLangSpace,
                SVGStylable {};
```

## Interface SVGSymbolElement

The **SVGSymbolElement** interface corresponds to the **'symbol'** element.

**IDL Definition**

```
interface SVGSymbolElement :
                SVGElement,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGFitToViewBox,
                events::EventTarget {};
```

## Interface SVGUseElement

The **SVGUseElement** interface corresponds to the **'use'** element.

**IDL Definition**

```
interface SVGUseElement :
                SVGElement,
                SVGURIReference,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedLength   x;
  readonly attribute SVGAnimatedLength   y;
  readonly attribute SVGAnimatedLength   width;
  readonly attribute SVGAnimatedLength   height;
  readonly attribute SVGElementInstance instanceRoot;
  readonly attribute SVGElementInstance animatedInstanceRoot;
};
```

**Attributes**

**readonly SVGAnimatedLength x**
: Corresponds to attribute **x** on the given **'use'** element.

**readonly SVGAnimatedLength y**
: Corresponds to attribute **y** on the given **'use'** element.

**readonly SVGAnimatedLength width**
: Corresponds to attribute **width** on the given **'use'** element.

**readonly SVGAnimatedLength height**
: Corresponds to attribute **height** on the given **'use'** element.

**readonly SVGElementInstance instanceRoot**
: The root of the "instance tree". See description of **SVGElementInstance** for a discussion on the instance tree.

**readonly SVGElementInstance animatedInstanceRoot**
: If the 'href' attribute is being animated, contains the current animated root of the "instance tree". If the 'href' attribute is not currently being animated, contains the same value as 'instanceRoot'. The root of the "instance tree". See description of **SVGElementInstance** for a discussion on the instance tree.

## Interface SVGElementInstance

For each **'use'** element, the SVG DOM maintains a shadow tree (the "instance tree") of objects of type **SVGElementInstance**. A **SVGElementInstance** represents a single node in the instance tree. The root object in the instance tree is pointed to by the instanceRoot attribute on the **SVGUseElement** object for the corresponding **'use'** element.

If the **'use'** element references a simple graphics element such as a **'rect'**, then there is only a single **SVGElementInstance** object, and the correspondingElement attribute on this **SVGElementInstance** object is the **SVGRectElement** that corresponds to the referenced **'rect'** element.

If the **'use'** element references a **'g'** which contains two **'rect'** elements, then the instance tree contains three **SVGElementInstance** objects, a root **SVGElementInstance** object whose correspondingElement is the **SVGGElement** object for the **'g'**, and then two child **SVGElementInstance** objects, each of which has its correspondingElement that is an **SVGRectElement** object.

If the referenced object is itself a **'use'**, or if there are **'use'** subelements within the referenced object, the instance tree will contain recursive expansion of the indirect references to form a complete tree. For example, if a **'use'** element references a **'g'**, and the **'g'** itself contains a **'use'**, and that **'use'** references a **'rect'**, then the instance tree for the original (outermost) **'use'** will

consist of a hierarchy of **SVGElementInstance** objects, as follows:

```
SVGElementInstance #1 (parentNode=null, firstChild=#2, correspondingElement is the 'g')
  SVGElementInstance #2 (parentNode=#1, firstChild=#3, correspondingElement is the other 'use')
    SVGElementInstance #3 (parentNode=#2, firstChild=null, corresponding Element is the 'rect')
```

**IDL Definition**

```
interface SVGElementInstance : events::EventTarget {
  readonly attribute SVGElement correspondingElement;
  readonly attribute SVGUseElement correspondingUseElement;
  readonly attribute SVGElementInstance parentNode;
  readonly attribute SVGElementInstanceList childNodes;
  readonly attribute SVGElementInstance firstChild;
  readonly attribute SVGElementInstance lastChild;
  readonly attribute SVGElementInstance previousSibling;
  readonly attribute SVGElementInstance nextSibling;
};
```

**Attributes**

**readonly SVGElement correspondingElement**
The corresponding element to which this object is an instance. For example, if a **'use'** element references a **'rect'** element, then an **SVGElementInstance** is created, with its correspondingElement being the **SVGElementInstance** object for the **'rect'** element.

**readonly SVGUseElement correspondingUseElement**
The corresponding **'use'** element to which this **SVGElementInstance** object belongs. When **'use'** elements are nested (e.g., a **'use'** references another **'use'** which references a graphics element such as a **'rect'**), then the correspondingUseElement is the outermost **'use'** (i.e., the one which indirectly references the **'rect'**, not the one with the direct reference).

**readonly SVGElementInstance parentNode**
The parent of this **SVGElementInstance** within the instance tree. All **SVGElementInstance** objects have a parent except the **SVGElementInstance** which corresponds to the element which was directly referenced by the **'use'** element, in which case parentNode is null.

**readonly SVGElementInstanceList childNodes**
An **SVGElementInstanceList** that contains all children of this **SVGElementInstance** within the instance tree. If there are no children, this is an **SVGElementInstanceList** containing no entries (i.e., an empty list).

**readonly SVGElementInstance firstChild**
The first child of this **SVGElementInstance** within the instance tree. If there is no such **SVGElementInstance**, this returns null.

**readonly SVGElementInstance lastChild**
The last child of this **SVGElementInstance** within the instance tree. If there is no such **SVGElementInstance**, this returns null.

**readonly SVGElementInstance previousSibling**
The **SVGElementInstance** immediately preceding this **SVGElementInstance**. If there is no such **SVGElementInstance**, this returns null.

**readonly SVGElementInstance nextSibling**
The **SVGElementInstance** immediately following this **SVGElementInstance**. If there is no such **SVGElementInstance**, this returns null.

## Interface SVGElementInstanceList

The **SVGElementInstanceList** interface provides the abstraction of an ordered collection of **SVGElementInstance** objects, without defining or constraining how this collection is implemented.

**IDL Definition**

```
interface SVGElementInstanceList {
  readonly attribute unsigned long length;
```

```
  SVGElementInstance item ( in unsigned long index );
};
```

**Attributes**

**readonly unsigned long length**

The number of **SVGElementInstance** objects in the list. The range of valid child indices is 0 to length-1 inclusive.

**Methods**

**item**

Returns the indexth item in the collection. If index is greater than or equal to the number of nodes in the list, this returns null.

**Parameters**

in unsigned long index  Index into the collection.

**Return value**

SVGElementInstance  The **SVGElementInstance** object at the indexth position in the **SVGElementInstanceList**, or null if that is not a valid index.

**No Exceptions**

## Interface SVGImageElement

The **SVGImageElement** interface corresponds to the **'image'** element.

**IDL Definition**

```
interface SVGImageElement :
               SVGElement,
               SVGURIReference,
               SVGTests,
               SVGLangSpace,
               SVGExternalResourcesRequired,
               SVGStylable,
               SVGTransformable,
               events::EventTarget {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

**Attributes**

**readonly SVGAnimatedLength x**

Corresponds to attribute **x** on the given **'image'** element.

**readonly SVGAnimatedLength y**

Corresponds to attribute **y** on the given **'image'** element.

**readonly SVGAnimatedLength width**

Corresponds to attribute **width** on the given **'image'** element.

**readonly SVGAnimatedLength height**

Corresponds to attribute **height** on the given **'image'** element.

**readonly SVGAnimatedPreserveAspectRatio preserveAspectRatio**

Corresponds to attribute **preserveAspectRatio** on the given element.

## Interface SVGSwitchElement

The **SVGSwitchElement** interface corresponds to the **'switch'** element.

**IDL Definition**

```
interface SVGSwitchElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {};
```

## Interface GetSVGDocument

In the case where an SVG document is embedded by reference, such as when an XHTML document has an **'object'** element whose **href** (or equivalent) attribute references an SVG document (i.e., a document whose MIME type is "image/svg+xml" and whose root element is thus an **'svg'** element), the SVG user agent is required to implement the **GetSVGDocument** interface for the element which references the SVG document (e.g., the HTML **'object'** or comparable referencing elements).

**IDL Definition**

```
interface GetSVGDocument {

  SVGDocument getSVGDocument (  )
                raises( DOMException );
};
```

**Methods**

**getSVGDocument**

Returns the SVGDocument object for the referenced SVG document.

**No Parameters**

**Return value**

SVGDocument The **SVGDocument** object for the referenced SVG document.

**Exceptions**

DOMException NOT_SUPPORTED_ERR: No SVGDocument object is available.

---

04 September 2001

# 6 Styling

## Contents



## 6.1 SVG's styling properties

SVG uses **styling properties** to describe many of its document parameters. Styling properties define how the graphics elements in the SVG content are to be rendered. SVG uses styling properties for the following:

- Parameters which are clearly visual in nature and thus lend themselves to styling. Examples include all attributes that define how an object is "painted," such as fill and stroke colors, linewidths and dash styles.
- Parameters having to do with text styling such as **'font-family'** and **'font-size'**.
- Parameters which impact the way that graphical elements are rendered, such as specifying clipping paths, masks, arrowheads, markers and filter effects.

SVG shares many of its styling properties with CSS [CSS2] and XSL [XSL]. Except for any additional SVG-specific rules explicitly mentioned in this specification, the normative definition of properties that are shared with CSS and XSL is the definition of the property from the CSS2 specification [CSS2].

The following properties are shared between CSS2 and SVG. Most of these properties are also defined in XSL:

- Font properties:
  - **'font'**
  - **'font-family'**
  - **'font-size'**
  - **'font-size-adjust'**
  - **'font-stretch'**
  - **'font-style'**
  - **'font-variant'**
  - **'font-weight'**
- Text properties:
  - **'direction'**
  - **'letter-spacing'**
  - **'text-decoration'**
  - **'unicode-bidi'**
  - **'word-spacing'**
- Other properties for visual media:
  - **'clip'** (Only applicable to outermost 'svg')
  - **'color'** is used to provide a potential indirect value (**currentColor**) for the **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'**, **'lighting-color'** properties. (The SVG properties which support color allow a color specification which is extended from CSS2 to accommodate color definitions in arbitrary color spaces. See Color profile descriptions.
  - **'cursor'**
  - **'display'**
  - **'overflow'** (Only applicable to elements which establish a new viewport)
  - **'visibility'**

The following SVG properties are not defined in [CSS2]. The complete normative definitions for these properties are found in this specification:

- Clipping, Masking and Compositing properties:
  - **'clip-path'**
  - **'clip-rule'**
  - **'mask'**
  - **'opacity'**
- Filter Effects properties:
  - **'enable-background'**
  - **'filter'**
  - **'flood-color'**
  - **'flood-opacity'**
  - **'lighting-color'**
- Gradient properties:
  - **'stop-color'**
  - **'stop-opacity'**
- Interactivity properties:
  - **'pointer-events'**
- Color and Painting properties:
  - **'color-interpolation'**
  - **'color-interpolation-filters'**
  - **'color-profile'**
  - **'color-rendering'**
  - **'fill'**
  - **'fill-opacity'**
  - **'fill-rule'**
  - **'image-rendering'**

    - **'marker'**
    - **'marker-end'**
    - **'marker-mid'**
    - **'marker-start'**
    - **'shape-rendering'**
    - **'stroke'**
    - **'stroke-dasharray'**
    - **'stroke-dashoffset'**
    - **'stroke-linecap'**
    - **'stroke-linejoin'**
    - **'stroke-miterlimit'**
    - **'stroke-opacity'**
    - **'stroke-width'**
    - **'text-rendering'**
- Text properties:
    - **'alignment-baseline'**
    - **'baseline-shift'**
    - **'dominant-baseline'**
    - **'glyph-orientation-horizontal'**
    - **'glyph-orientation-vertical'**
    - **'kerning'**
    - **'text-anchor'**
    - **'writing-mode'**

A table that lists and summarizes the styling properties can be found in the Property Index.

## 6.2 Usage scenarios for styling

SVG has many usage scenarios, each with different needs. Here are three common usage scenarios:

1. **SVG content used as an exchange format (style sheet language-independent)**:

   In some usage scenarios, reliable interoperability of SVG content across software tools is the main goal. Since support for a particular style sheet language is not guaranteed across all implementations, it is a requirement that SVG content can be fully specified without the use of a style sheet language.

2. **SVG content generated as the output from XSLT [XSLT]**:

   XSLT offers the ability to take a stream of arbitrary XML content as input, apply potentially complex transformations, and then generate SVG content as output. XSLT can be used to transform XML data extracted from databases into an SVG graphical representation of that data. It is a requirement that fully specified SVG content can be generated from XSLT.
3. **SVG content styled with CSS [CSS2]**:

   CSS is a widely implemented declarative language for assigning styling properties to XML content, including SVG. It represents a combination of features, simplicity and compactness that makes it very suitable for many applications of SVG. It is a requirement that CSS styling can be applied to SVG content.

## 6.3 Alternative ways to specify styling properties

Styling properties can be assigned to SVG elements in the following two ways:

- **Presentation attributes**

  Styling properties can be assigned using SVG's **presentation attributes**. For each styling property defined in this specification, there is a corresponding XML presentation attribute available on all relevant SVG elements. Detailed information on the presentation attributes can be found in Specifying properties using the presentation attributes.

  The presentation attributes are style sheet language independent and thus are applicable to usage scenario 1 above (i.e., tool interoperability). Because it is straightforward to assign values to XML attributes from XSLT, the presentation attributes are well-suited to usage scenario 2 above (i.e., SVG generation from XSLT). (See Styling with XSL below.)

  Conforming SVG Interpreters and Conforming SVG Viewers are required to support SVG's presentation attributes.

- **CSS**

  To support usage scenario 3 above, SVG content can be styled with CSS. For more information, see Styling with CSS.

  Conforming SVG Interpreters and Conforming SVG Viewers that support CSS styling of generic (i.e., text-based) XML content are required to support CSS styling of SVG content.

## 6.4 Specifying properties using the presentation attributes

For each styling property defined in this specification (see Property Index), there is a corresponding XML attribute (the **presentation attribute**) with the same name that is available on all relevant SVG elements. For example, SVG has a **'fill'** property that defines how to paint the interior of a shape. There is a corresponding presentation attribute with the same name (i.e., **fill**) that can be used to specify a value for the **'fill'** property on a given element.

The following example shows how the **'fill'** and **'stroke'** properties can be assigned to a rectangle using the **fill** and **stroke** presentation attributes. The rectangle will be filled with red and outlined with blue:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
     xmlns="http://www.w3.org/2000/svg">
  <rect x="200" y="100" width="600" height="300"
        fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

The presentation attributes offer the following advantages:

- **Broad support**. All versions of Conforming SVG Interpreters and Conforming SVG Viewers are required to support the presentation attributes.
- **Simplicity**. Styling properties can be attached to elements by simply providing a value for the presentation

attribute on the proper elements.
- **Restyling**. SVG content that uses the presentation attributes is highly compatible with downstream processing using XSLT [XSLT] or supplemental styling by adding CSS style rules to override some of the presentation attributes.
- **Convenient generation using XSLT [XSLT]**. In some cases, XSLT can be used to generate fully styled SVG content. The presentation attributes are compatible with convenient generation of SVG from XSLT.

In some situations, SVG content that uses the presentation attributes has potential limitations versus SVG content that is styled with a style sheet language such as CSS (see Styling with CSS). In other situations, such as when an XSLT style sheet generates SVG content from semantically rich XML source files, the limitations below may not apply. Depending on the situation, some of the following potential limitations may or may not apply to the presentation attributes:

- **Styling attached to content**. The presentation attributes are attached directly to particular elements, thereby diminishing potential advantages that comes from abstracting styling from content, such as the ability to restyle documents for different uses and environments.
- **Flattened data model**. In and of themselves, the presentation attributes do not offer the higher level abstractions that you get with a styling system, such as the ability to define named collections of properties which are applied to particular categories of elements. The result is that, in many cases, important higher level semantic information can be lost, potentially making document reuse and restyling more difficult.
- **Potential increase in file size**. Many types of graphics use similar styling properties across multiple elements. For example, a company organization chart might assign one collection of styling properties to the boxes around temporary workers (e.g., dashed outlines, red fill), and a different collection of styling properties to permanent workers (e.g., solid outlines, blue fill). Styling systems such as CSS allow collections of properties to be defined once in a file. With the styling attributes, it might be necessary to specify presentation attributes on each different element.
- **Potential difficulty when embedded into a CSS-styled parent document**. When SVG content is embedded in other XML, and the desire is to style all aspects of the compound document with CSS, use of the presentation attributes might introduce complexity and difficulty. In this case, it is sometimes easier if the SVG content does not use the presentation attributes and instead is styled using CSS facilities.

For user agents that support CSS, the presentation attributes must be translated to corresponding CSS style rules according to rules described in section 6.4.4 of the CSS2 specification, Precedence of non-CSS presentational hints, with the additional clarification that the presentation attributes are conceptually inserted into a new author style sheet which is the first in the author style sheet collection. The presentation attributes thus will participate in the CSS2 cascade as if they were replaced by corresponding CSS style rules placed at the start of the author style sheet with a specificity of zero. In general, this means that the presentation attributes have lower priority than other CSS style rules specified in author style sheets or **style** attributes.

User agents that do not support CSS must ignore any CSS style rules defined in CSS style sheets and **style** attributes. In this case, the CSS cascade does not apply. (Inheritance of properties, however, does apply. See Property inheritance.)

An **!important** declaration within a presentation attribute definition is an error.

Animation of presentation attributes is equivalent to animating the corresponding property. Thus, the same effect occurs from animating the presentation attribute with **attributeType="XML"** as occurs with animating the corresponding property with **attributeType="CSS"**.

## 6.5 Entity definitions for the presentation attributes

The following entities are defined in the DTD for all of the presentation attributes in SVG:

```
<!-- The following presentation attributes have to do with specifying color. -->
<!ENTITY % PresentationAttributes-Color
  "color %Color; #IMPLIED
   color-interpolation (auto | sRGB | linearRGB | inherit) #IMPLIED
   color-rendering (auto | optimizeSpeed | optimizeQuality | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to container elements. -->
<!ENTITY % PresentationAttributes-Containers
  "enable-background %EnableBackgroundValue; #IMPLIED " >

<!-- The following presentation attributes apply to 'feFlood' elements. -->
<!ENTITY % PresentationAttributes-feFlood
  "flood-color %SVGColor; #IMPLIED
   flood-opacity %OpacityValue; #IMPLIED " >

<!-- The following presentation attributes apply to filling and stroking operations. -->
<!ENTITY % PresentationAttributes-FillStroke
  "fill %Paint; #IMPLIED
   fill-opacity %OpacityValue; #IMPLIED
   fill-rule %ClipFillRule; #IMPLIED
   stroke %Paint; #IMPLIED
   stroke-dasharray %StrokeDashArrayValue; #IMPLIED
   stroke-dashoffset %StrokeDashOffsetValue; #IMPLIED
   stroke-linecap (butt | round | square | inherit) #IMPLIED
   stroke-linejoin (miter | round | bevel | inherit) #IMPLIED
   stroke-miterlimit %StrokeMiterLimitValue; #IMPLIED
   stroke-opacity %OpacityValue; #IMPLIED
   stroke-width %StrokeWidthValue; #IMPLIED " >

<!-- The following presentation attributes apply to filter primitives. -->
<!ENTITY % PresentationAttributes-FilterPrimitives
  "color-interpolation-filters (auto | sRGB | linearRGB | inherit) #IMPLIED " >

<!-- The following presentation attributes have to do with selecting a font to use. -->
<!ENTITY % PresentationAttributes-FontSpecification
  "font-family %FontFamilyValue; #IMPLIED
   font-size %FontSizeValue; #IMPLIED
   font-size-adjust %FontSizeAdjustValue; #IMPLIED
   font-stretch (normal | wider | narrower | ultra-condensed | extra-condensed |
                 condensed | semi-condensed | semi-expanded | expanded |
                 extra-expanded | ultra-expanded | inherit) #IMPLIED
   font-style (normal | italic | oblique | inherit) #IMPLIED
   font-variant (normal | small-caps | inherit) #IMPLIED
   font-weight (normal | bold | bolder | lighter | 100 | 200 | 300 |
                400 | 500 | 600 | 700 | 800 | 900 | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to gradient 'stop' elements. -->
<!ENTITY % PresentationAttributes-Gradients
  "stop-color %SVGColor; #IMPLIED
   stop-opacity %OpacityValue; #IMPLIED " >

<!-- The following presentation attributes apply to graphics elements. -->
<!ENTITY % PresentationAttributes-Graphics
  "clip-path %ClipPathValue; #IMPLIED
   clip-rule %ClipFillRule; #IMPLIED
   cursor %CursorValue; #IMPLIED
   display (inline | block | list-item | run-in | compact | marker |
            table | inline-table | table-row-group | table-header-group |
            table-footer-group | table-row | table-column-group | table-column |
            table-cell | table-caption | none | inherit) #IMPLIED
   filter %FilterValue; #IMPLIED
   image-rendering (auto | optimizeSpeed | optimizeQuality | inherit) #IMPLIED
   mask %MaskValue; #IMPLIED
   opacity %OpacityValue; #IMPLIED
   pointer-events (visiblePainted | visibleFill | visibleStroke | visible |
                   painted | fill | stroke | all | none | inherit) #IMPLIED
   shape-rendering (auto | optimizeSpeed | crispEdges | geometricPrecision | inherit) #IMPLIED
   text-rendering (auto | optimizeSpeed | optimizeLegibility | geometricPrecision | inherit) #IMPLIED
   visibility (visible | hidden | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to 'image' elements. -->
<!ENTITY % PresentationAttributes-Images
  "color-profile CDATA #IMPLIED " >

<!--The following presentation attributes apply to 'feDiffuseLighting' and 'feSpecularLighting' elements. -->
<!ENTITY % PresentationAttributes-LightingEffects
  "lighting-color %SVGColor; #IMPLIED " >
```

```
<!-- The following presentation attributes apply to marker operations. -->
<!ENTITY % PresentationAttributes-Markers
  "marker-start %MarkerValue; #IMPLIED
   marker-mid %MarkerValue; #IMPLIED
   marker-end %MarkerValue; #IMPLIED " >

<!-- The following presentation attributes apply to text content elements. -->
<!ENTITY % PresentationAttributes-TextContentElements
  "alignment-baseline (baseline | top | before-edge | text-top | text-before-edge |
                       middle | bottom | after-edge | text-bottom | text-after-edge |
                       ideographic | lower | hanging | mathematical | inherit) #IMPLIED
   baseline-shift %BaselineShiftValue; #IMPLIED
   direction (ltr | rtl | inherit) #IMPLIED
   dominant-baseline (auto | autosense-script | no-change | reset|
                      ideographic | lower | hanging | mathematical | inherit ) #IMPLIED
   glyph-orientation-horizontal %GlyphOrientationHorizontalValue; #IMPLIED
   glyph-orientation-vertical %GlyphOrientationVerticalValue; #IMPLIED
   kerning %KerningValue; #IMPLIED
   letter-spacing %SpacingValue; #IMPLIED
   text-anchor (start | middle | end | inherit) #IMPLIED
   text-decoration %TextDecorationValue; #IMPLIED
   unicode-bidi (normal | embed | bidi-override | inherit) #IMPLIED
   word-spacing %SpacingValue; #IMPLIED " >

<!-- The following presentation attributes apply to 'text' elements. -->
<!ENTITY % PresentationAttributes-TextElements
  "writing-mode (lr-tb | rl-tb | tb-rl | lr | rl | tb | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to elements that establish viewports. -->
<!ENTITY % PresentationAttributes-Viewports
  "clip %ClipValue; #IMPLIED
   overflow (visible | hidden | scroll | auto | inherit) #IMPLIED " >

<!--The following represents the complete list of presentation attributes. -->
<!ENTITY % PresentationAttributes-All
  "%PresentationAttributes-Color;
   %PresentationAttributes-Containers;
   %PresentationAttributes-feFlood;
   %PresentationAttributes-FillStroke;
   %PresentationAttributes-FilterPrimitives;
   %PresentationAttributes-FontSpecification;
   %PresentationAttributes-Gradients;
   %PresentationAttributes-Graphics;
   %PresentationAttributes-Images;
   %PresentationAttributes-LightingEffects;
   %PresentationAttributes-Markers;
   %PresentationAttributes-TextContentElements;
   %PresentationAttributes-TextElements;
   %PresentationAttributes-Viewports;" >
```

## 6.6 Styling with XSL

XSL style sheets (see [XSLT]) define how to transform XML content into something else, usually other XML. When XSLT is used in conjunction with SVG, sometimes SVG content will serve as both input and output for XSL style sheets. Other times, XSL style sheets will take non-SVG content as input and generate SVG content as output.

The following example uses an external XSL style sheet to transform SVG content into modified SVG content (see Referencing external style sheets). The style sheet sets the **'fill'** and **'stroke'** properties on all rectangles to red and blue, respectively:

**mystyle.xsl**

```
<?xml version="1.0" standalone="no"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <!-- Add DOCTYPE -->
  <xsl:template match="/">
    <xsl:text disable-output-escaping="yes"><!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
```

```
      "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <!-- Add styling to all 'rect' elements -->
  <xsl:template match="rect">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:attribute name="fill">red</xsl:attribute>
      <xsl:attribute name="stroke">blue</xsl:attribute>
      <xsl:attribute name="stroke-width">3</xsl:attribute>
    </xsl:copy>
  </xsl:template>

  <!-- default is to copy input element -->
  <xsl:template match="*|@*|text()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

**SVG file to be transformed by mystyle.xsl**

```
<?xml version="1.0" standalone="no"?>
<svg width="10cm" height="5cm"
     xmlns="http://www.w3.org/2000/svg">
  <rect x="2cm" y="1cm" width="6cm" height="3cm"/>
</svg>
```

**SVG content after applying mystyle.xsl**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
      "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="5cm"
     xmlns="http://www.w3.org/2000/svg">
  <rect x="2cm" y="1cm" width="6cm" height="3cm" fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

## 6.7 Styling with CSS

SVG implementations that support CSS are required to support the following:

- External CSS style sheets referenced from the current document (see Referencing external style sheets)
- Internal CSS style sheets (i.e., style sheets embedded within the current document, such as within an SVG **'style'** element)
- Inline style (i.e., CSS property declarations within a **style** attribute on a particular SVG element)

The following example shows the use of an external CSS style sheet to set the **'fill'** and **'stroke'** properties on all rectangles to red and blue, respectively:

**mystyle.css**

```
rect {
  fill: red;
  stroke: blue;
  stroke-width: 3
}
```

**SVG file referencing mystyle.css**

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet href="mystyle.css" type="text/css"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
     xmlns="http://www.w3.org/2000/svg">
```

```
  <rect x="200" y="100" width="600" height="300"/>
</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

CSS style sheets can be embedded within SVG content inside of a **'style'** element. The following example uses an internal CSS style sheet to achieve the same result as the previous example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <style type="text/css"><![CDATA[
      rect {
        fill: red;
        stroke: blue;
        stroke-width: 3
      }
    ]]></style>
  </defs>
  <rect x="200" y="100" width="600" height="300"/>
</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

Note how the CSS style sheet is placed within a CDATA construct (i.e., `<![CDATA[ ... ]]>`). Placing internal CSS style sheets within CDATA blocks is sometimes necessary since CSS style sheets can include characters, such as ">", which conflict with XML parsers. Even if a given style sheet does not use characters that conflict with XML parsing, it is highly recommended that internal style sheets be placed inside CDATA blocks.

Implementations that support CSS are also required to support CSS inline style. Similar to the **style** attribute in HTML, CSS inline style can be declared within a **style** attribute in SVG by specifying a semicolon-separated list of property declarations, where each property declaration has the form "name: value".

The following example shows how the **'fill'** and **'stroke'** properties can be assigned to a rectangle using the **style** attribute. Just like the previous example, the rectangle will be filled with red and outlined with blue:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
     xmlns="http://www.w3.org/2000/svg">
  <rect x="200" y="100" width="600" height="300"
        style="fill:red; stroke:blue; stroke-width:3"/>
</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

In an SVG user agent that supports CSS style sheets, the following facilities from [CSS2] must be supported:

- CSS2 selectors within style sheets (reference: [Selectors]).
- External CSS style sheets [XML-SS], CSS style sheets within **'style'** elements and CSS declaration blocks within **style** attributes attached to specific SVG elements.
- CSS2 rules for assigning property values, cascading and inheritance.
- @font-face, @media, @import and @charset rules within style sheets.

- CSS2's dynamic pseudo-classes :hover, :active and :focus and pseudo-classes :first-child, :visited, :link and :lang. The remaining CSS2 pseudo-classes, including those having to do with generated content, are not part of the SVG language definition. (Note: an SVG element gains focus when it is selected. See Text selection.)
- For the purposes of aural media, SVG represents a CSS-stylable XML grammar. In user agents that support aural style sheets, CSS aural style properties can be applied as defined in [CSS2]. (See Aural style sheets.)
- CSS style sheets defined within a **'style'** element can be immediate character data content of the **'style'** element or can be embedded within a `CDATA` block.

SVG defines an @color-profile at-rule [CSS2-ATRULES] for defining color profiles so that ICC color profiles can be applied to CSS-styled SVG content.

Note the following about relative URIs and external CSS style sheets: The CSS2 specification [CSS-URI] says that relative URIs (as defined in [RFC2396]) within style sheets are resolved such that the base URI is that of the style sheet, not that of the referencing document.

## 6.8 Case sensitivity of property names and values

Property declarations via presentation attributes are expressed in XML [XML10], which is case-sensitive. CSS property declarations specified either in CSS style sheets or in a **style** attribute, on the other hand, are generally case-insensitive with some exceptions (see section 4.1.3 Characters and case in the CSS2 specification).

Because presentation attributes are expressed as XML attributes, presentation attributes are case-sensitive and must match the exact name as listed under "Entity definitions for the presentation attributes", above. When using a presentation attribute to specify a value for the **'fill'** property, the presentation attribute must be specified as **'fill'** and not **'FILL'** or **'Fill'**. Keyword values, such as "italic" in `font-style="italic"`, are also case-sensitive and must be specified using the exact case used in the specification which defines the given keyword. For example, the keyword "sRGB" must have lowercase "s" and uppercase "RGB".

Property declarations within CSS style sheets or in a **style** attribute must only conform to CSS rules, which are generally more lenient with regard to case sensitivity. However, to promote consistency across the different ways for expressing styling properties, it is strongly recommended that authors use the exact property names (usually, lowercase letters and hyphens) as defined in the relevant specification and express all keywords using the same case as is required by presentation attributes and not take advantage of CSS's ability to ignore case.

## 6.9 Facilities from CSS and XSL used by SVG

SVG shares various relevant properties and approaches common to CSS and XSL, plus the semantics of many of the processing rules.

SVG shares the following facilities with CSS and XSL:

- Shared properties. Many of SVG's properties are shared between CSS2, XSL and SVG. (See list of shared properties).
- Syntax rules. (The normative references are [CSS2 syntax and basic data types] and [The grammar of CSS2].)
- Allowable data types. (The normative reference is [CSS2 syntax and basic data types]), with the exception that SVG allows <length> and <angle> values without a unit identifier. See Units.)
- Inheritance rules.
- The color keywords from CSS2 that correspond to the colors used by objects in the user's environment. (The

normative reference is [CSS2 system colors].)

- For implementations that support CSS styling of SVG content, then that styling must be compatible with various other rules in CSS. (See Styling with CSS.)

## 6.10 Referencing external style sheets

External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0" [XML-SS].

## 6.11 The 'style' element

The **'style'** element allows style sheets to be embedded directly within SVG content. SVG's **'style'** element has the same attributes as the corresponding element in HTML (see HTML's **'style'** element).

```
<!ELEMENT style (#PCDATA) >
<!ATTLIST style
  %stdAttrs;
  xml:space (preserve) #FIXED "preserve"
  type %ContentType; #REQUIRED
  media %MediaDesc; #IMPLIED
  title %Text; #IMPLIED >
```

Attribute definitions:

**`type` = *content-type***
: This attribute specifies the style sheet language of the element's contents. The style sheet language is specified as a content type (e.g., "text/css"), as per [RFC2045]. Authors must supply a value for this attribute; there is no default value.
*Animatable: no.*

**`media` = *media-descriptors***
: This attribute specifies the intended destination medium for style information. It may be a single media descriptor or a comma-separated list. The default value for this attribute is "all". The set of recognized *media-descriptors* are the list of media types recognized by CSS2 [CSS2 Recognized media types].
*Animatable: no.*

**`title` = *advisory-title***
: (For compatibility with [HTML4]) This attribute specifies an advisory title for the **'style'** element.
*Animatable: no.*

***Attributes defined elsewhere:***
: **%stdAttrs;, xml:space.**

The syntax of style data depends on the style sheet language.

Some style sheet languages might allow a wider variety of rules in the **'style'** element than in the **style** attribute. For example, with CSS, rules can be declared within a **'style'** element that cannot be declared within a **style** attribute.

An example showing the **'style'** element is provided above (see example).

## 6.12 The class attribute

Attribute definitions:

**`class`** = ***list***
: This attribute assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.
*Animatable: yes.*

The **class** attribute assigns one or more class names to an element. The element may be said to belong to these classes. A class name may be shared by several element instances. The class attribute has several roles:

- As a style sheet selector (when an author wishes to assign style information to a set of elements).
- For general purpose processing by user agents.

In the following example, the **'text'** element is used in conjunction with the **class** attribute to markup document messages. Messages appear in both English and French versions.

```
<!-- English messages -->
<text class="info" lang="en">Variable declared twice</text>
<text class="warning" lang="en">Undeclared variable</text>
<text class="error" lang="en">Bad syntax for variable name</text>

<!-- French messages -->
<text class="info" lang="fr">Variable déclarée deux fois</text>
<text class="warning" lang="fr">Variable indéfinie</text>
<text class="error" lang="fr">Erreur de syntaxe pour variable</text>
```

In an SVG user agent that supports CSS styling, the following CSS style rules would tell visual user agents to display informational messages in green, warning messages in yellow, and error messages in red:

```
text.info    { color: green }
text.warning { color: yellow }
text.error   { color: red }
```

## 6.13 The style attribute

The **style** attribute allows per-element style rules to be specified directly on a given element. When CSS styling is used, CSS inline style is specified by including semicolon-separated property declarations of the form "name : value" within the **style** attribute

Attribute definitions:

**`style`** = ***style***
: This attribute specifies style information for the current element. The style attribute specifies style information for a single element. The style sheet language of inline style rules is given by the value of attribute contentStyleType on the **'svg'** element. The syntax of style data depends on the style sheet language.
*Animatable: no.*

The style attribute may be used to apply a particular style to an individual SVG element. If the style will be reused for several elements, authors should use the **'style'** element to regroup that information. For optimal flexibility, authors should define styles in external style sheets.

An example showing the **style** attribute is provided above (see example).

## 6.14 Specifying the default style sheet language

The **contentStyleType** attribute on the **'svg'** element specifies the default style sheet language for the given document fragment.

**`contentStyleType` = "%ContentType;"**

Identifies the default style sheet language for the given document. This attribute sets the style sheet language for the **style** attributes that are available on many elements. The value **%ContentType;** specifies a media type, per [RFC2045]. The default value is "text/css".
*Animatable: no.*

## 6.15 Property inheritance

Whether or not the user agent supports CSS, property inheritance in SVG follows the property inheritance rules defined in the CSS2 specification. The normative definition for property inheritance is section 6.2 of the CSS2 specification (see Inheritance).

The definition of each property indicates whether the property can inherit the value of its parent.

In SVG, as in CSS2, most elements inherit computed values [CSS2-COMPUTED]. For cases where something other than computed values are inherited, the property definition will describe the inheritance rules. For specified values [CSS2-SPECIFIED] which are expressed in user units, in pixels (e.g., "20px") or in absolute values [CSS2-COMPUTED], the computed value equals the specified value. For specified values which use certain relative units (i.e., *em*, *ex* and percentages), the computed value will have the same units as the value to which it is relative. Thus, if the parent element has a **'font-size'** of "10pt" and the current element has a **'font-size'** of "120%", then the computed value for **'font-size'** on the current element will be "12pt". In cases where the referenced value for relative units is not expressed in any of the standard SVG units (i.e., CSS units or user units), such as when a percentage is used relative to the current viewport or an object bounding box, then the computed value will be in user units.

Note that SVG has some facilities wherein a property which is specified on an ancestor element might effect its descendant element, even if the descendant element has a different assigned value for that property. For example, if a **'clip-path'** property is specified on an ancestor element, and the current element has a **'clip-path'** of **'none'**, the ancestor's clipping path still applies to the current element because the semantics of SVG state that the clipping path used on a given element is the intersection of all clipping paths specified on itself and all ancestor elements. The key concept is that property assignment (with possible property inheritance) happens first. After properties values have been assigned to the various elements, then the user agent applies the semantics of each assigned property, which might result in the property assignment of an ancestor element affecting the rendering of its descendants.

## 6.16 The scope/range of styles

The following define the scope/range of style sheets:

**Stand-alone SVG document**

There is one parse tree. Style sheets defined anywhere within the SVG document (in style elements or style attributes, or in external style sheets linked with the style sheet processing instruction) apply across the entire SVG document.

**Stand-alone SVG document embedded in an HTML or XML document with the 'img', 'object' (HTML) or 'image' (SVG) elements**

There are two completely separate parse trees; one for the referencing document (perhaps HTML or XHTML), and one for the SVG document. Style sheets defined anywhere within the referencing document (in style elements or style attributes, or in external style sheets linked with the style sheet processing instruction) apply across the entire referencing document but have no effect on the referenced SVG document. Style sheets defined anywhere within the referenced SVG document (in style elements or style attributes, or in external style sheets linked with the style sheet processing instruction) apply across the entire SVG document, but do not affect the referencing document (perhaps HTML or XHTML). To get the same styling across both the [X]HTML document and the SVG document, link them both to the same style sheet.

**Stand-alone SVG content textually included in an XML document**

There is a single parse tree, using multiple namespaces; one or more subtrees are in the SVG namespace. Style sheets defined anywhere within the XML document (in style elements or style attributes, or in external style sheets linked with the style sheet processing instruction) apply across the entire document, including those parts of it in the SVG namespace. To get different styling for the SVG part, use the style attribute, or put an ID on the 'svg' element and use contextual CSS selectors, or use XSL selectors.

## 6.17 User agent style sheet

The user agent shall maintain a *user agent style sheet* [CSS2-CASCADE-RULES] for elements in the SVG namespace for visual media [CSS2-VISUAL]. The user agent style sheet below is expressed using CSS syntax; however, user agents are required to support the behavior that corresponds to this default style sheet even if CSS style sheets are not supported in the user agent:

```
svg, symbol, image, marker, pattern, foreignObject { overflow: hidden }
svg { width:attr(width); height:attr(height) }
```

The first line of the above user agent style sheet will cause the initial clipping path to be established at the bounds of the initial viewport. Furthermore, it will cause new clipping paths to be established at the bounds of the listed elements, all of which are elements that establish a new viewport. (Refer to the description of SVG's use of the **'overflow'** property for more information.)

The second line of the above user agent style sheet will cause the **width** and **height** attributes on the **'svg'** element to be used as the default values for the **'width'** and **'height'** properties during [CSS2-LAYOUT].

## 6.18 Aural style sheets

For the purposes of aural media, SVG represents a stylable XML grammar. In user agents that support CSS aural style sheets, aural style properties [CSS2-AURAL] can be applied as defined in [CSS2].

Aural style properties can be applied to any SVG element that can contain character data content, including **'desc'**, **'title**, **'tspan'**. **'tref'**. **'altGlyph'** and **'textPath'**. On user agents that support aural style sheets, the following [CSS2] properties can be applied:

| | |
|---|---|
| **'azimuth'** | [CSS2-azimuth] |
| **'cue'** | [CSS2-cue] |
| **'cue-after'** | [CSS2-cue-after] |
| **'cue-before'** | [CSS2-cue-before] |

| | |
|---|---|
| **'elevation'** | [CSS2-elevation] |
| **'pause'** | [CSS2-pause] |
| **'pause-after'** | [CSS2-pause-after] |
| **'pause-before'** | [CSS2-pause-before] |
| **'pitch'** | [CSS2-pitch] |
| **'pitch-range'** | [CSS2-pitch-range] |
| **'play-during'** | [CSS2-play-during] |
| **'richness'** | [CSS2-richness] |
| **'speak'** | [CSS2-speak] |
| **'speak-header'** | [CSS2-speak-header] |
| **'speak-numeral'** | [CSS2-speak-numeral] |
| **'speak-punctuation'** | [CSS2-speak-punctuation] |
| **'speech-rate'** | [CSS2-speech-rate] |
| **'stress'** | [CSS2-stress] |
| **'voice-family'** | [CSS2-voice-family] |
| **'volume'** | [CSS2-volume] |

For user agents that support aural style sheets and also support [DOM2], the user agent is required to support the DOM interfaces defined in [DOM2-CSS] that correspond to aural properties [CSS2-AURAL]. (See Relationship with DOM2 CSS object model.)

## 6.19 DOM interfaces

The following interfaces are defined below: SVGStyleElement.

### Interface SVGStyleElement

The **SVGStyleElement** interface corresponds to the **'style'** element.

**IDL Definition**

```
interface SVGStyleElement : SVGElement {
           attribute DOMString xmlspace;
                       // raises DOMException on setting
           attribute DOMString type;
                       // raises DOMException on setting
           attribute DOMString media;
                       // raises DOMException on setting
           attribute DOMString title;
                       // raises DOMException on setting
};
```

**Attributes**

**DOMString xmlspace**

Corresponds to attribute **xml:space** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString type**

Corresponds to attribute **type** on the given **'style'** element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString media**

Corresponds to attribute **media** on the given **'style'** element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString title**

Corresponds to attribute **title** on the given **'style'** element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

---

04 September 2001

# 7 Coordinate Systems, Transformations and Units

## Contents



## 7.1 Introduction

For all media, the **SVG canvas** describes "the space where the SVG content is rendered." The canvas is infinite for each dimension of the space, but rendering occurs relative to a finite rectangular region of the canvas. This finite rectangular region is called the **SVG viewport**. For visual media [CSS2-VISUAL], the SVG viewport is the viewing area where the user sees the SVG content.

The size of the SVG viewport (i.e., its width and height) is determined by a negotiation process (see Establishing the size of the initial viewport) between the SVG document fragment and its parent (real or implicit). Once that negotiation process is completed, the SVG user agent is provided the following information:

- a number (usually an integer) that represents the width in "pixels" of the viewport
- a number (usually an integer) that represents the height in "pixels" of the viewport
- (highly desirable but not required) a real number value that indicates the size in real world units, such as millimeters, of a "pixel" (i.e., a *px* unit as defined in [CSS2 lengths])

Using the above information, the SVG user agent determines the **viewport**, an initial **viewport coordinate system** and an initial **user coordinate system** such that the two coordinates systems are

identical. Both coordinates systems are established such that the origin matches the origin of the viewport (for the root viewport, the viewport origin is at the top/left corner), and one unit in the initial coordinate system equals one "pixel" in the viewport. (See Initial coordinate system.) The viewport coordinate system is also called **viewport space** and the user coordinate system is also called **user space**.

Lengths in SVG can be specified as:

- (if no unit identifier is provided) values in user space -- for example, "15"
- (if a unit identifier is provided) a length expressed as an absolute or relative unit measure -- for example, "15mm" or "5em"

The supported length unit identifiers are: em, ex, px, pt, pc, cm, mm, in, and percentages.

A new user space (i.e., a new current coordinate system) can be established at any place within an SVG document fragment by specifying **transformations** in the form of **transformation matrices** or simple transformation operations such as rotation, skewing, scaling and translation. Establishing new user spaces via coordinate system transformations are fundamental operations to 2D graphics and represent the usual method of controlling the size, position, rotation and skew of graphic objects.

New viewports also can be established. By establishing a new viewport, you can redefine the meaning of percentages units and provide a new reference rectangle for "fitting" a graphic into a particular rectangular area. ("Fit" means that a given graphic is transformed in such a way that its bounding box in user space aligns exactly with the edges of a given viewport.)

## 7.2 The initial viewport

The SVG user agent negotiates with its parent user agent to determine the viewport into which the SVG user agent can render the document. In some circumstances, SVG content will be embedded (by reference or inline) within a containing document. This containing document might include attributes, properties and/or other parameters (explicit or implicit) which specify or provide hints about the dimensions of the viewport for the SVG content. SVG content itself optionally can provide information about the appropriate viewport region for the content via the **width** and **height** XML attributes on the outermost **'svg'** element. The negotiation process uses any information provided by the containing document and the SVG content itself to choose the viewport location and size.

The **width** attribute on the outermost **'svg'** element establishes the viewport's width, unless the following conditions are met:

- the SVG content is a separately stored resource that is embedded by reference (such as the **'object'** element in [XHTML]), or the SVG content is embedded inline within a containing document;
- and the referencing element or containing document is styled using CSS [CSS2] or XSL [XSL];
- and there are CSS-compatible positioning properties [CSS2-POSN] specified on the referencing element (e.g., the **'object'** element) or on the containing document's outermost **'svg'** element that are sufficient to establish the width of the viewport.

Under these conditions, the positioning properties establish the viewport's width.

Similarly, if there are positioning properties [CSS2-POSN] specified on the referencing element or on the outermost **'svg'** that are sufficient to establish the height of the viewport, then these positioning properties establish the viewport's height; otherwise, the **height** attribute on the outermost **'svg'** element establishes the viewport's height.

If the **width** or **height** attributes on the outermost **'svg'** element are in user units (i.e., no unit identifier has been provided), then the value is assumed to be equivalent to the same number of "px" units (see Units).

In the following example, an SVG graphic is embedded inline within a parent XML document which is formatted using CSS layout rules. Since CSS positioning properties are not provided on the outermost **'svg'** element, the **width="100px"** and **height="200px"** attributes determine the size of the initial viewport:

```
<?xml version="1.0" standalone="yes"?>
<parent xmlns="http://some.url">

   <!-- SVG graphic -->
   <svg xmlns='http://www.w3.org/2000/svg'
      width="100px" height="200px">
      <path d="M100,100 Q200,400,300,100"/>
      <!-- rest of SVG graphic would go here -->
   </svg>

</parent>
```

The initial clipping path for the SVG document fragment is established according to the rules described in The initial clipping path.

## 7.3 The initial coordinate system

For the outermost **'svg'** element, the SVG user agent determines an initial **viewport coordinate system** and an initial **user coordinate system** such that the two coordinates systems are identical. The origin of both coordinate systems is at the origin of the viewport, and one unit in the initial coordinate system equals one "pixel" (i.e., a *px* unit as defined in [CSS2 lengths]) in the viewport. In most cases, such as stand-alone SVG documents or SVG document fragments embedded (by reference or inline) within XML parent documents where the parent's layout is determined by CSS [CSS2] or XSL [XSL], the initial viewport coordinate system (and therefore the initial user coordinate system) has its origin at the top/left of the viewport, with the positive x-axis pointing towards the right, the positive y-axis pointing down, and text rendered with an "upright" orientation, which means glyphs are oriented such that Roman characters and full-size ideographic characters for Asian scripts have the top edge of the corresponding glyphs oriented upwards and the right edge of the corresponding glyphs oriented to the right.

If the SVG implementation is part of a user agent which supports styling XML documents using CSS2-compatible *px* units, then the SVG user agent should get its initial value for the size of a *px* unit in real world units to match the value used for other XML styling operations; otherwise, if the user agent can determine the size of a *px* unit from its environment, it should use that value; otherwise, it should choose an appropriate size for one *px* unit. In all cases, the size of a *px* must be in conformance with the rules described in [CSS2 lengths].

Example InitialCoords below shows that the initial coordinate system has the origin at the top/left with the

x-axis pointing to the right and the y-axis pointing down. The initial user coordinate system has one user unit equal to the parent (implicit or explicit) user agent's "pixel".

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300px" height="100px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example InitialCoords - SVG's initial coordinate system</desc>

  <g fill="none" stroke="black" stroke-width="3" >
    <line x1="0" y1="1.5" x2="300" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="100" />
  </g>
  <g fill="red" stroke="none" >
    <rect x="0" y="0" width="3" height="3" />
    <rect x="297" y="0" width="3" height="3" />
    <rect x="0" y="97" width="3" height="3" />
  </g>
  <g font-size="14" font-family="Verdana" >
    <text x="10" y="20">(0,0)</text>
    <text x="240" y="20">(300,0)</text>
    <text x="10" y="90">(0,100)</text>
  </g>
</svg>
```


***Example InitialCoords***

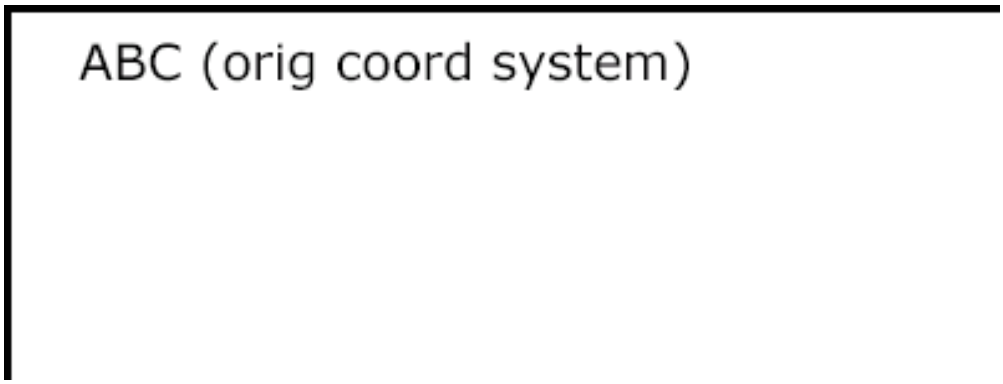
*View this example as SVG (SVG-enabled browsers only)*

## 7.4 Coordinate system transformations

A new user space (i.e., a new current coordinate system) can be established by specifying **transformations** in the form of a **transform** attribute on a container element or graphics element or a **viewBox** attribute on an **'svg'**, **'symbol'**, **'marker'**, **'pattern'** and the **'view'** element. The **transform** and **viewBox** attributes transform user space coordinates and lengths on sibling attributes on the given element (see effect of the **transform** attribute on sibling attributes and effect of the **viewBox** attribute on sibling attributes) and all of its descendants. Transformations can be nested, in which case the effect of the transformations are cumulative.

Example OrigCoordSys below shows a document without transformations. The text string is specified in the initial coordinate system.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example OrigCoordSys - Simple transformations: original picture</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (orig coord system)
    </text>
  </g>
</svg>
```



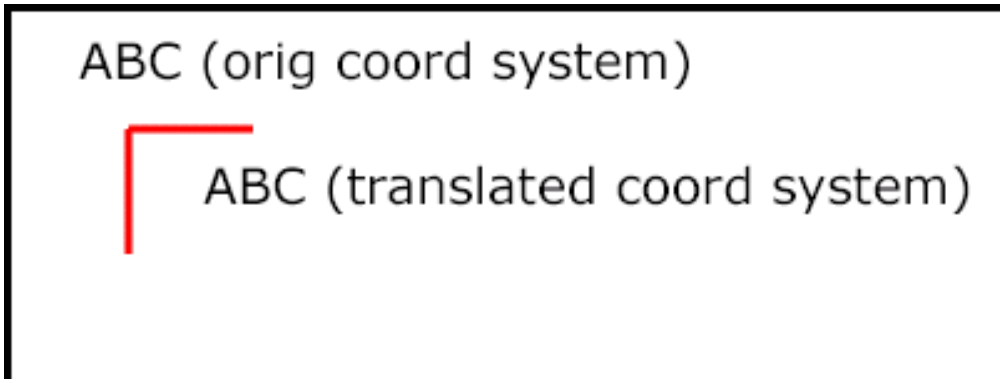

***Example OrigCoordSys***

*View this example as SVG (SVG-enabled browsers only)*

Example NewCoordSys establishes a new user coordinate system by specifying **transform="translate(50,50)"** on the third **'g'** element below. The new user coordinate system has its origin at location (50,50) in the original coordinate system. The result of this transformation is that the coordinate (30,30) in the new user coordinate system gets mapped to coordinate (80,80) in the original coordinate system (i.e., the coordinates have been translated by 50 units in X and 50 units in Y).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example NewCoordSys - New user coordinate system</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (orig coord system)
    </text>
  </g>
  <!-- Establish a new coordinate system, which is
```

```
       shifted (i.e., translated) from the initial coordinate
       system by 50 user units along each axis. -->
  <g transform="translate(50,50)">
    <g fill="none" stroke="red" stroke-width="3" >
      <!-- Draw lines of length 50 user units along
           the axes of the new coordinate system -->
      <line x1="0" y1="0" x2="50" y2="0" stroke="red" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (translated coord system)
    </text>
  </g>
</svg>
```




***Example NewCoordSys***

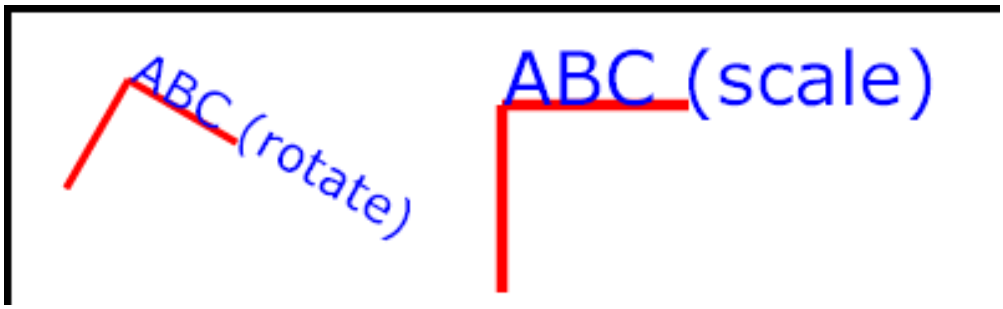
*View this example as SVG (SVG-enabled browsers only)*

Example RotateScale illustrates simple **rotate** and **scale** transformations. The example defines two new coordinate systems:

- one which is the result of a translation by 50 units in X and 30 units in Y, followed by a rotation of 30 degrees
- another which is the result of a translation by 200 units in X and 40 units in Y, followed by a scale transformation of 1.5.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="120px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example RotateScale - Rotate and scale transforms</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Establish a new coordinate system whose origin is at (50,30)
       in the initial coord. system and which is rotated by 30 degrees. -->
  <g transform="translate(50,30)">
    <g transform="rotate(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
```

```
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (rotate)
      </text>
    </g>
  </g>
  <!-- Establish a new coordinate system whose origin is at (200,40)
       in the initial coord. system and which is scaled by 1.5. -->
  <g transform="translate(200,40)">
    <g transform="scale(1.5)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (scale)
      </text>
    </g>
  </g>
</svg>
```



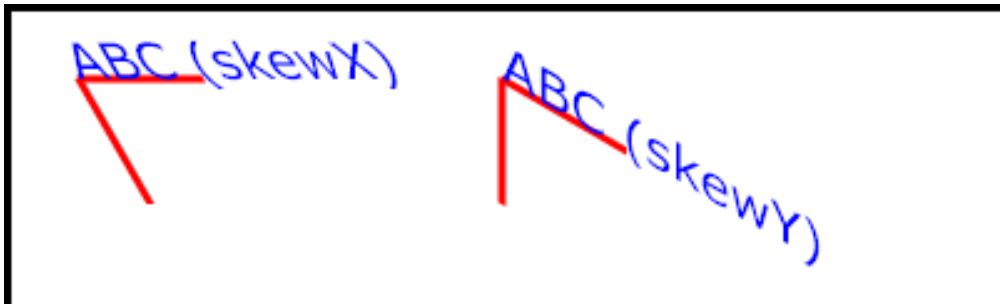

***Example RotateScale***

*View this example as SVG (SVG-enabled browsers only)*

Example Skew defines two coordinate systems which are **skewed** relative to the origin coordinate system.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="120px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Skew - Show effects of skewX and skewY</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Establish a new coordinate system whose origin is at (30,30)
       in the initial coord. system and which is skewed in X by 30 degrees. -->
  <g transform="translate(30,30)">
    <g transform="skewX(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (skewX)
```

```
      </text>
    </g>
  </g>
  <!-- Establish a new coordinate system whose origin is at (200,30)
       in the initial coord. system and which is skewed in Y by 30 degrees. -->
  <g transform="translate(200,30)">
    <g transform="skewY(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (skewY)
      </text>
    </g>
  </g>
</svg>
```




***Example Skew***

*View this example as SVG (SVG-enabled browsers only)*

Mathematically, all transformations can be represented as 3x3 **transformation matrices** of the following form:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Since only six values are used in the above 3x3 matrix, a transformation matrix is also expressed as a vector: **[a b c d e f]**.

Transformations map coordinates and lengths from a new coordinate system into a previous coordinate system:

$$\begin{bmatrix} x_{prevCoordSys} \\ y_{prevCoordSys} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{newCoordSys} \\ y_{newCoordSys} \\ 1 \end{bmatrix}$$

Simple transformations are represented in matrix form as follows:

- Translation is equivalent to the matrix

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[1 0 0 1 tx ty]**, where *tx* and *ty* are the distances to translate coordinates in *X* and *Y*, respectively.

- Scaling is equivalent to the matrix

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[sx 0 0 sy 0 0]**. One unit in the *X* and *Y* directions in the new coordinate system equals *sx* and *sy* units in the previous coordinate system, respectively.

- Rotation about the origin is equivalent to the matrix

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[cos(a) sin(a) -sin(a) cos(a) 0 0]**, which has the effect of rotating the coordinate system axes by angle *a*.

- A skew transformation along the x-axis is equivalent to the matrix

$$\begin{bmatrix} 1 & \tan(a) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[1 0 tan(a) 1 0 0]**, which has the effect of skewing X coordinates by angle *a*.

- A skew transformation along the y-axis is equivalent to the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ \tan(a) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[1 tan(a) 0 1 0 0]**, which has the effect of skewing Y coordinates by angle *a*.

## 7.5 Nested transformations

Transformations can be nested to any level. The effect of nested transformations is to post-multiply (i.e., concatenate) the subsequent transformation matrices onto previously defined transformations:

$$\begin{bmatrix} x_{prev} \\ y_{prev} \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{curr} \\ y_{curr} \\ 1 \end{bmatrix}$$

For each given element, the accumulation of all transformations that have been defined on the given element and all of its ancestors up to and including the element that established the current viewport (usually, the **'svg'** element which is the most immediate ancestor to the given element) is called the **current transformation matrix** or **CTM**. The CTM thus represents the mapping of current user coordinates to viewport coordinates:

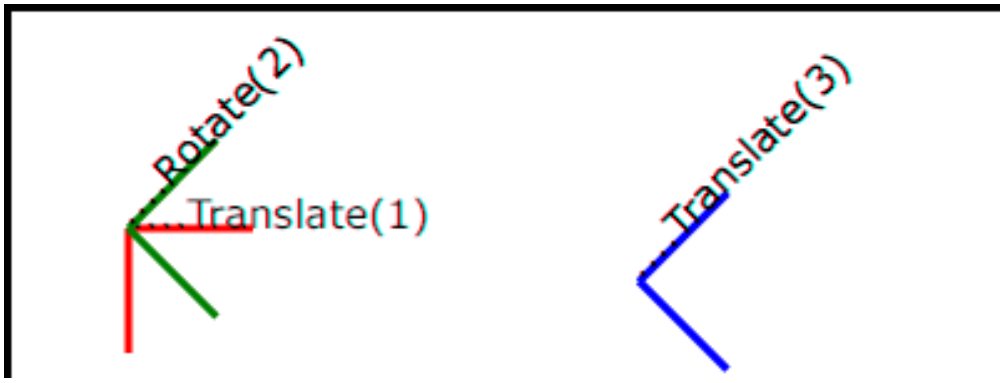
$$CTM = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} a_n & c_n & e_n \\ b_n & d_n & f_n \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{viewport} \\ y_{viewport} \\ 1 \end{bmatrix} = CTM \cdot \begin{bmatrix} x_{userspace} \\ y_{userspace} \\ 1 \end{bmatrix}$$

Example Nested illustrates nested transformations.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example Nested - Nested transformations</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <!-- First, a translate -->
  <g transform="translate(50,90)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="16" font-family="Verdana" >
      ....Translate(1)
    </text>
    <!-- Second, a rotate -->
    <g transform="rotate(-45)">
      <g fill="none" stroke="green" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
```

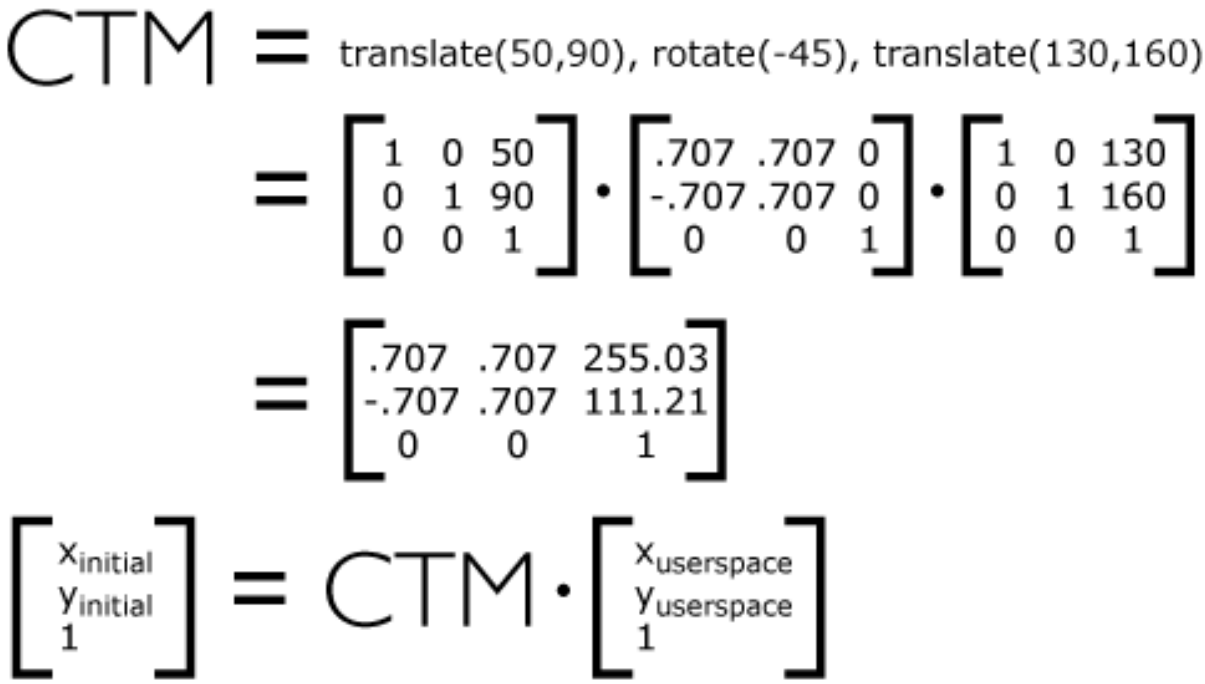
```
      </g>
      <text x="0" y="0" font-size="16" font-family="Verdana" >
        ....Rotate(2)
      </text>
      <!-- Third, another translate -->
      <g transform="translate(130,160)">
        <g fill="none" stroke="blue" stroke-width="3" >
          <line x1="0" y1="0" x2="50" y2="0" />
          <line x1="0" y1="0" x2="0" y2="50" />
        </g>
        <text x="0" y="0" font-size="16" font-family="Verdana" >
          ....Translate(3)
        </text>
      </g>
    </g>
  </g>
</svg>
```




***Example Nested***

*View this example as SVG (SVG-enabled browsers only)*

In the example above, the CTM within the third nested transformation (i.e., the **transform="translate(130,160)"**) consists of the concatenation of the three transformations, as follows:




## 7.6 The transform attribute

The value of the **transform** attribute is a <transform-list>, which is defined as a list of transform definitions, which are applied in the order provided. The individual transform definitions are separated by whitespace and/or a comma. The available types of transform definitions include:

- **matrix(<a> <b> <c> <d> <e> <f>)**, which specifies a transformation in the form of a transformation matrix of six values. **matrix(a,b,c,d,e,f)** is equivalent to applying the transformation matrix **[a b c d e f]**.
- **translate(<tx> [<ty>])**, which specifies a translation by *tx* and *ty*. If *<ty>* is not provided, it is assumed to be zero.
- **scale(<sx> [<sy>])**, which specifies a scale operation by *sx* and *sy*. If *<sy>* is not provided, it is assumed to be equal to *<sx>*.
- **rotate(<rotate-angle> [<cx> <cy>])**, which specifies a rotation by **<rotate-angle>** degrees about a given point.
  If optional parameters **<cx>** and **<cy>** are not supplied, the rotate is about the origin of the current user coordinate system. The operation corresponds to the matrix **[cos(a) sin(a) -sin(a) cos(a) 0 0]**.
  If optional parameters **<cx>** and **<cy>** are supplied, the rotate is about the point **(<cx>, <cy>)**. The operation represents the equivalent of the following specification: **translate(<cx>, <cy>) rotate(<rotate-angle>) translate(-<cx>, -<cy>)**.
- **skewX(<skew-angle>)**, which specifies a skew transformation along the x-axis.

- **skewY(<skew-angle>)**, which specifies a skew transformation along the y-axis.

All numeric values are real <number>s.

If a list of transforms is provided, then the net effect is as if each transform had been specified separately in the order provided. For example,

```
<g transform="translate(-10,-20) scale(2) rotate(45) translate(5,10)">
  <!-- graphics elements go here -->
</g>
```

is functionally equivalent to:

```
<g transform="translate(-10,-20)">
  <g transform="scale(2)">
    <g transform="rotate(45)">
      <g transform="translate(5,10)">
        <!-- graphics elements go here -->
      </g>
    </g>
  </g>
</g>
```

The **transform** attribute is applied to an element before processing any other coordinate or length values supplied for that element. In the element

```
<rect x="10" y="10" width="20" height="20" transform="scale(2)"/>
```

the x, y, width and height values are processed after the current coordinate system has been scaled uniformly by a factor of 2 by the **transform** attribute. Attributes x, y, width and height (and any other attributes or properties) are treated as values in the new user coordinate system, not the previous user coordinate system. Thus, the above **'rect'** element is functionally equivalent to:

```
<g transform="scale(2)">
  <rect x="10" y="10" width="20" height="20"/>
</g>
```

The following is the Backus-Naur Form (BNF) for values for the **transform** attribute. The following notation is used:

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- (): grouping
- |: separates alternatives
- double quotes surround literals

```
transform-list:
    wsp* transforms? wsp*

transforms:
    transform
    | transform comma-wsp+ transforms
```

```
transform:
    matrix
    | translate
    | scale
    | rotate
    | skewX
    | skewY

matrix:
    "matrix" wsp* "(" wsp*
       number comma-wsp
       number comma-wsp
       number comma-wsp
       number comma-wsp
       number comma-wsp
       number wsp* ")"

translate:
    "translate" wsp* "(" wsp* number ( comma-wsp number )? wsp* ")"

scale:
    "scale" wsp* "(" wsp* number ( comma-wsp number )? wsp* ")"

rotate:
    "rotate" wsp* "(" wsp* number ( comma-wsp number comma-wsp number )? wsp* ")"

skewX:
    "skewX" wsp* "(" wsp* number wsp* ")"

skewY:
    "skewY" wsp* "(" wsp* number wsp* ")"

number:
    sign? integer-constant
    | sign? floating-point-constant

comma-wsp:
    (wsp+ comma? wsp*) | (comma wsp*)

comma:
    ","

integer-constant:
    digit-sequence

floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent

fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."

exponent:
    ( "e" | "E" ) sign? digit-sequence

sign:
    "+" | "-"

digit-sequence:
    digit
    | digit digit-sequence

digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

wsp:
    (#x20 | #x9 | #xD | #xA)
```

For the **transform** attribute:

*Animatable: yes.*

See the **'animateTransform'** element for information on animating transformations.

## 7.7 The viewBox attribute

It is often desirable to specify that a given set of graphics stretch to fit a particular container element. The **viewBox** attribute provides this capability.

All elements that establish a new viewport (see elements that establish viewports), plus the **'marker'**, **'pattern'** and **'view'** elements have attribute **viewBox**. The value of the **viewBox** attribute is a list of four numbers **<min-x>**, **<min-y>**, **<width>** and **<height>**, separated by whitespace and/or a comma, which specify a rectangle in user space which should be mapped to the bounds of the viewport established by the given element, taking into account attribute **preserveAspectRatio**. If specified, an additional transformation is applied to all descendants of the given element to achieve the specified effect.

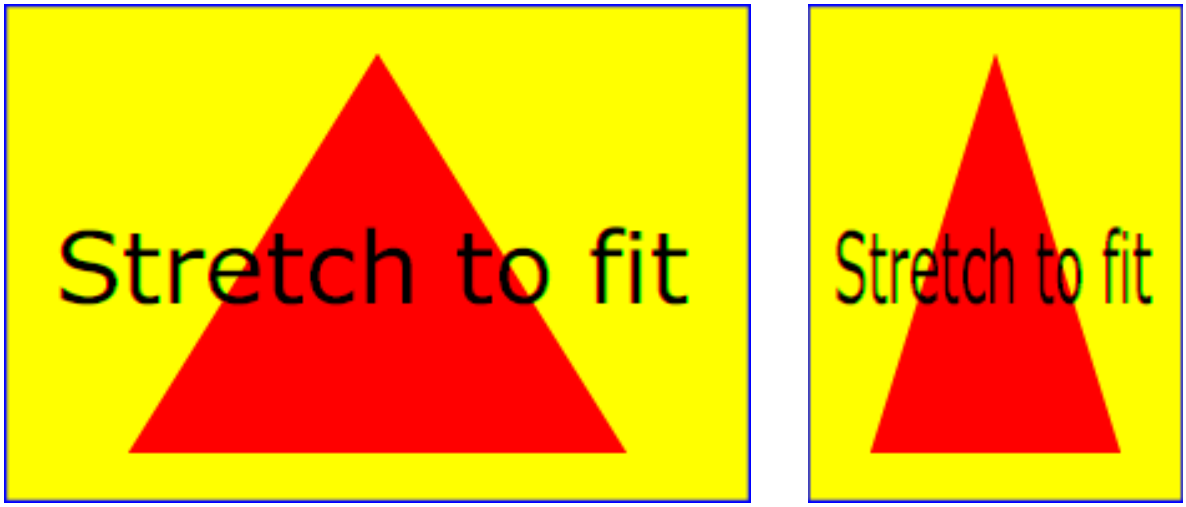
A negative value for **<width>** or **<height>** is an error (see Error processing). A value of zero disables rendering of the element.

Example ViewBox illustrates the use of the **viewBox** attribute on the outermost **'svg'** element to specify that the SVG content should stretch to fit bounds of the viewport.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300px" height="200px"
     viewBox="0 0 1500 1000" preserveAspectRatio="none"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example ViewBox - uses the viewBox
   attribute to automatically create an initial user coordinate
   system which causes the graphic to scale to fit into the
   viewport no matter what size the viewport is.</desc>

  <!-- This rectangle goes from (0,0) to (1500,1000) in user space.
       Because of the viewBox attribute above,
       the rectangle will end up filling the entire area
       reserved for the SVG content. -->
  <rect x="0" y="0" width="1500" height="1000"
        fill="yellow" stroke="blue" stroke-width="12"  />

  <!-- A large, red triangle -->
  <path fill="red"  d="M 750,100 L 250,900 L 1250,900 z"/>

  <!-- A text string that spans most of the viewport -->
  <text x="100" y="600" font-size="200" font-family="Verdana" >
    Stretch to fit
  </text>
</svg>
```

**Rendered into viewport with width=300px, height=200px**

**Rendered into viewport with width=150px, height=200px**




***Example ViewBox***

*View this example as SVG (SVG-enabled browsers only)*

The effect of the **viewBox** attribute is that the user agent automatically supplies the appropriate transformation matrix to map the specified rectangle in user space to the bounds of a designated region (often, the viewport). To achieve the effect of the example on the left, with viewport dimensions of 300 by 200 pixels, the user agent needs to automatically insert a transformation which scales both X and Y by 0.2. The effect is equivalent to having a viewport of size 300px by 200px and the following supplemental transformation in the document, as follows:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300px" height="200px"
     xmlns="http://www.w3.org/2000/svg">

  <g transform="scale(0.2)">

    <!-- Rest of document goes here -->

  </g>
</svg>
```

To achieve the effect of the example on the right, with viewport dimensions of 150 by 200 pixels, the user agent needs to automatically insert a transformation which scales X by 0.1 and Y by 0.2. The effect is equivalent to having a viewport of size 150px by 200px and the following supplemental transformation in the document, as follows:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="150px" height="200px"
     xmlns="http://www.w3.org/2000/svg">

  <g transform="scale(0.1 0.2)">

    <!-- Rest of document goes here -->

  </g>
```

```
</svg>
```

(Note: in some cases the user agent will need to supply a **translate** transformation in addition to a **scale** transformation. For example, on an outermost **'svg'**, a **translate** transformation will be needed if the **viewBox** attributes specifies values other than zero for **<min-x>** or **<min-y>**.)

Unlike the **transform** attribute (see effect of the **transform** on sibling attributes), the automatic transformation that is created due to a **viewBox** does not affect the **x**, **y**, **width** and **height** attributes (or in the case of the **'marker'** element, the **markerWidth** and **markerHeight** attributes) on the element with the **viewBox** attribute. Thus, in the example above which shows an **'svg'** element which has attributes **width**, **height** and **viewBox**, the **width** and **height** attributes represent values in the coordinate system that exists *before* the **viewBox** transformation is applied. On the other hand, like the **transform** attribute, it does establish a new coordinate system for all other attributes and for descendant elements.

For the **viewBox** attribute:

*Animatable: yes.*

## 7.8 The preserveAspectRatio attribute

In some cases, typically when using the **viewBox** attribute, it is desirable that the graphics stretch to fit non-uniformly to take up the entire viewport. In other cases, it is desirable that uniform scaling be used for the purposes of preserving the aspect ratio of the graphics.

Attribute **preserveAspectRatio="<align> [<meetOrSlice>]"**, which is available for all elements that establish a new viewport (see elements that establish viewports), plus the **'image'**, **'marker'**, **'pattern'** and **'view'** elements, indicates whether or not to force uniform scaling.

For elements that establish a new viewport (see elements that establish viewports), plus the **'marker'**, **'pattern'** and **'view'** elements, **preserveAspectRatio** only applies when a value has been provided for **viewBox** on the same element. For these elements, if attribute **viewBox** is not provided, then **preserveAspectRatio** is ignored.

For **'image'** elements, **preserveAspectRatio** indicates how referenced images should be fitted with respect to the reference rectangle and whether the aspect ratio of the referenced image should be preserved with respect to the current user coordinate system.

The **<align>** parameter indicates whether to force uniform scaling and, if so, the alignment method to use in case the aspect ratio of the **viewBox** doesn't match the aspect ratio of the viewport. The **<align>** parameter must be one of the following strings:

- **none** - Do not force uniform scaling. Scale the graphic content of the given element non-uniformly if necessary such that the element's bounding box exactly matches the viewport rectangle.
  (Note: if **<align>** is **none**, then the optional **<meetOrSlice>** value is ignored.)
- **xMinYMin** - Force uniform scaling.
  Align the **<min-x>** of the element's **viewBox** with the smallest X value of the viewport.
  Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the viewport.

- **xMidYMin** - Force uniform scaling.
  Align the midpoint X value of the element's **viewBox** with the midpoint X value of the viewport.
  Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the viewport.
- **xMaxYMin** - Force uniform scaling.
  Align the **<min-x>+<width>** of the element's **viewBox** with the maximum X value of the viewport.
  Align the **<min-y>** of the element's **viewBox** with the smallest Y value of the viewport.
- **xMinYMid** - Force uniform scaling.
  Align the **<min-x>** of the element's **viewBox** with the smallest X value of the viewport.
  Align the midpoint Y value of the element's **viewBox** with the midpoint Y value of the viewport.
- **xMidYMid** (the default) - Force uniform scaling.
  Align the midpoint X value of the element's **viewBox** with the midpoint X value of the viewport.
  Align the midpoint Y value of the element's **viewBox** with the midpoint Y value of the viewport.
- **xMaxYMid** - Force uniform scaling.
  Align the **<min-x>+<width>** of the element's **viewBox** with the maximum X value of the viewport.
  Align the midpoint Y value of the element's **viewBox** with the midpoint Y value of the viewport.
- **xMinYMax** - Force uniform scaling.
  Align the **<min-x>** of the element's **viewBox** with the smallest X value of the viewport.
  Align the **<min-y>+<height>** of the element's **viewBox** with the maximum Y value of the viewport.
- **xMidYMax** - Force uniform scaling.
  Align the midpoint X value of the element's **viewBox** with the midpoint X value of the viewport.
  Align the **<min-y>+<height>** of the element's **viewBox** with the maximum Y value of the viewport.
- **xMaxYMax** - Force uniform scaling.
  Align the **<min-x>+<width>** of the element's **viewBox** with the maximum X value of the viewport.
  Align the **<min-y>+<height>** of the element's **viewBox** with the maximum Y value of the viewport.

The **<meetOrSlice>** parameter is optional and, if provided, is separated from the **<align>** value by one or more spaces and then must be one of the following strings:

- **meet** (the default) - Scale the graphic such that:
  - aspect ratio is preserved
  - the entire **viewBox** is visible within the viewport
  - the **viewBox** is scaled up as much as possible, while still meeting the other criteria

  In this case, if the aspect ratio of the graphic does not match the viewport, some of the viewport will extend beyond the bounds of the **viewBox** (i.e., the area into which the **viewBox** will draw will be smaller than the viewport).
- **slice** - Scale the graphic such that:
  - aspect ratio is preserved
  - the entire viewport is covered by the **viewBox**
  - the **viewBox** is scaled down as much as possible, while still meeting the other criteria

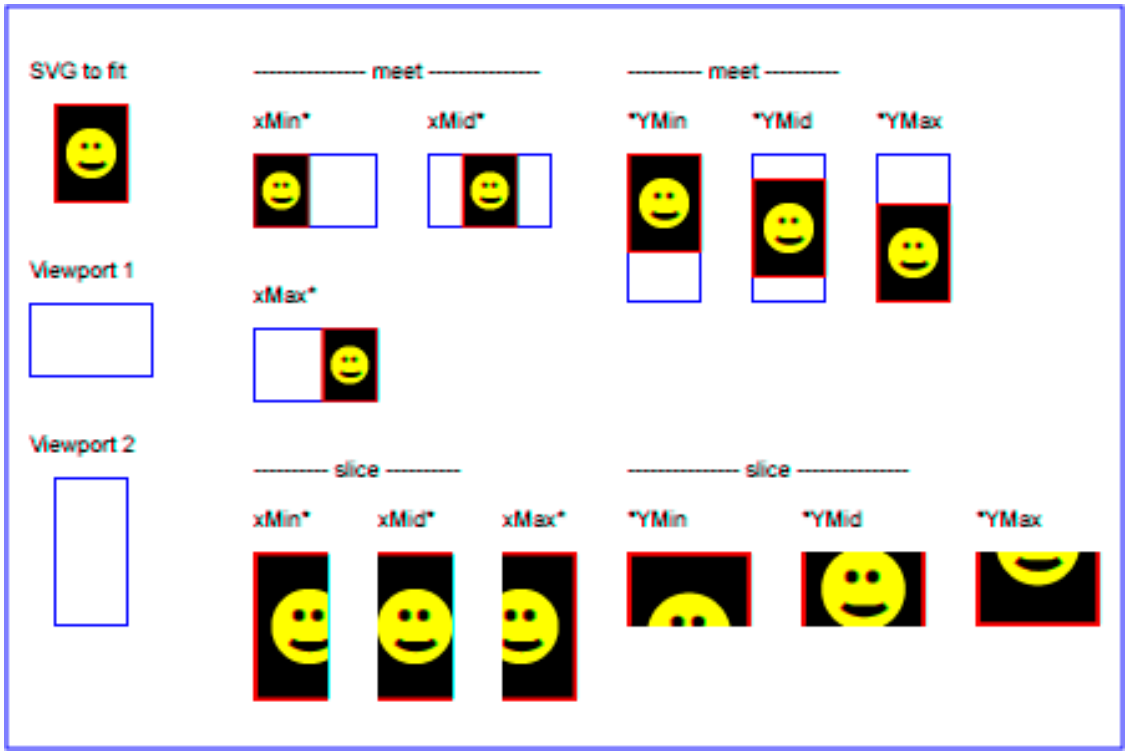
  In this case, if the aspect ratio of the **viewBox** does not match the viewport, some of the **viewBox** will extend beyond the bounds of the viewport (i.e., the area into which the **viewBox** will draw is larger than the viewport).

Example PreserveAspectRatio illustrates the various options on **preserveAspectRatio**. To save space, XML entities have been defined for the three repeated graphic objects, the rectangle with the smile inside

and the outlines of the two rectangles which have the same dimensions as the target viewports. The example creates several new viewports by including **'svg'** sub-elements embedded inside the outermost **'svg'** element (see Establishing a new viewport).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
[ <!ENTITY Smile "
<rect x='.5' y='.5' width='29' height='39' fill='black' stroke='red'/>
<g transform='translate(0, 5)'>
<circle cx='15' cy='15' r='10' fill='yellow'/>
<circle cx='12' cy='12' r='1.5' fill='black'/>
<circle cx='17' cy='12' r='1.5' fill='black'/>
<path d='M 10 19 A 8 8 0 0 0 20 19' stroke='black' stroke-width='2'/>
</g>
">
<!ENTITY Viewport1 "<rect x='.5' y='.5' width='49' height='29'
fill='none' stroke='blue'/>">
<!ENTITY Viewport2 "<rect x='.5' y='.5' width='29' height='59'
fill='none' stroke='blue'/>">
]>

<svg width="450px" height="300px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example PreserveAspectRatio - illustrates preserveAspectRatio attribute</desc>
  <rect x="1" y="1" width="448" height="298"
        fill="none" stroke="blue"/>
  <g font-size="9">
    <text x="10" y="30">SVG to fit</text>
    <g transform="translate(20,40)">&Smile;</g>
    <text x="10" y="110">Viewport 1</text>
    <g transform="translate(10,120)">&Viewport1;</g>
    <text x="10" y="180">Viewport 2</text>
    <g transform="translate(20,190)">&Viewport2;</g>

    <g id="meet-group-1" transform="translate(100, 60)">
      <text x="0" y="-30">--------------- meet ---------------</text>
      <g><text y="-10">xMin*</text>&Viewport1;
        <svg preserveAspectRatio="xMinYMin meet" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
      <g transform="translate(70,0)"><text y="-10">xMid*</text>&Viewport1;
        <svg preserveAspectRatio="xMidYMid meet" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
      <g transform="translate(0,70)"><text y="-10">xMax*</text>&Viewport1;
        <svg preserveAspectRatio="xMaxYMax meet" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
    </g>

    <g id="meet-group-2" transform="translate(250, 60)">
      <text x="0" y="-30">---------- meet ----------</text>
      <g><text y="-10">*YMin</text>&Viewport2;
        <svg preserveAspectRatio="xMinYMin meet" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
      <g transform="translate(50, 0)"><text y="-10">*YMid</text>&Viewport2;
        <svg preserveAspectRatio="xMidYMid meet" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
      <g transform="translate(100, 0)"><text y="-10">*YMax</text>&Viewport2;
        <svg preserveAspectRatio="xMaxYMax meet" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
    </g>

    <g id="slice-group-1" transform="translate(100, 220)">
      <text x="0" y="-30">---------- slice ----------</text>
      <g><text y="-10">xMin*</text>&Viewport2;
        <svg preserveAspectRatio="xMinYMin slice" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
      <g transform="translate(50,0)"><text y="-10">xMid*</text>&Viewport2;
        <svg preserveAspectRatio="xMidYMid slice" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
```

```
      <g transform="translate(100,0)"><text y="-10">xMax*</text>&Viewport2;
        <svg preserveAspectRatio="xMaxYMax slice" viewBox="0 0 30 40"
             width="30" height="60">&Smile;</svg></g>
    </g>

    <g id="slice-group-2" transform="translate(250, 220)">
      <text x="0" y="-30">--------------- slice ---------------</text>
      <g><text y="-10">*YMin</text>&Viewport1;
        <svg preserveAspectRatio="xMinYMin slice" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
      <g transform="translate(70,0)"><text y="-10">*YMid</text>&Viewport1;
        <svg preserveAspectRatio="xMidYMid slice" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
      <g transform="translate(140,0)"><text y="-10">*YMax</text>&Viewport1;
        <svg preserveAspectRatio="xMaxYMax slice" viewBox="0 0 30 40"
             width="50" height="30">&Smile;</svg></g>
    </g>
  </g>
</svg>
```




***Example PreserveAspectRatio***

*View this example as SVG (SVG-enabled browsers only)*

For the **preserveAspectRatio** attribute:

*Animatable: yes.*

## 7.9 Establishing a new viewport

At any point in an SVG drawing, you can establish a new viewport into which all contained graphics is drawn by including an **'svg'** element inside SVG content. By establishing a new viewport, you also implicitly establish a new viewport coordinate system, a new user coordinate system, and, potentially, a new clipping path (see the definition of the **'overflow'** property). Additionally, there is a new meaning for percentage units defined to be relative to the current viewport since a new viewport has been established (see Units)

The bounds of the new viewport are defined by the **x**, **y**, **width** and **height** attributes on the element establishing the new viewport, such as an **'svg'** element. Both the new viewport coordinate system and the new user coordinate system have their origins at (**x**, **y**), where **x** and **y** represent the value of the corresponding attributes on the element establishing the viewport. The orientation of the new viewport coordinate system and the new user coordinate system correspond to the orientation of the current user coordinate system for the element establishing the viewport. A single unit in the new viewport coordinate system and the new user coordinate system are the same size as a single unit in the current user coordinate system for the element establishing the viewport.

Here is an example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
     xmlns="http://www.w3.org/2000/svg">
  <desc>This SVG drawing embeds another one,
    thus establishing a new viewport
  </desc>
  <!-- The following statement establishing a new viewport
       and renders SVG drawing B into that viewport -->
  <svg x="25%" y="25%" width="50%" height="50%">
     <!-- drawing B goes here -->
  </svg>
</svg>
```

For an extensive example of creating new viewports, see Example PreserveAspectRatio.

The following elements establish new viewports:

- The **'svg'** element
- A **'symbol'** element define new viewports whenever they are instanced by a **'use'** element.
- An **'image'** element that references an SVG file will result in the establishment of a temporary new viewport since the referenced resource by definition will have an **'svg'** element.
- A **'foreignObject'** element creates a new viewport for rendering the content that is within the element.

Whether a new viewport also establishes a new additional clipping path is determined by the value of the **'overflow'** property on the element that establishes the new viewport. If a clipping path is created to correspond to the new viewport, the clipping path's geometry is determined by the value of the **'clip'** property. Also, see Clip to viewport vs. clip to **viewBox**.

## 7.10 Units

All coordinates and lengths in SVG can be specified with or without a **unit identifier**.

When a coordinate or length value is a number without a unit identifier (e.g., "25"), then the given coordinate or length is assumed to be in user units (i.e., a value in the current user coordinate system). For example:

```
<text style="font-size: 50">Text size is 50 user units</text>
```

Alternatively, a coordinate or length value can be expressed as a number following by a unit identifier (e.g., "25cm" or "15em"). The list of unit identifiers in SVG matches the list of unit identifiers in CSS: em, ex, px, pt, pc, cm, mm, in and percentages. The following describes how the various unit identifiers are processed:

- As in CSS, the *em* and *ex* unit identifiers are relative to the current font's *font-size* and *x-height*, respectively.

- One *px* unit is defined to be equal to one user unit. Thus, a length of "5px" is the same as a length of "5".

  Note that at initialization, a user unit in the the initial coordinate system is equivalenced to the parent environment's notion of a *px* unit. Thus, in the the initial coordinate system, because the user coordinate system aligns exactly with the parent's coordinate system, and because often the parent's coordinate system aligns with the device pixel grid, "5px" might actually map to 5 devices pixels. However, if there are any coordinate system transformation due to the use of **transform** or **viewBox** attributes, because "5px" maps to 5 user units and because the coordinate system transformations have resulted in a revised user coordinate system, "5px" likely will not map to 5 device pixels. As a result, in most circumstances, "px" units will not map to the device pixel grid.

- The other absolute unit identifiers from CSS (i.e., pt, pc, cm, mm, in) are all defined as an appropriate multiple of one *px* unit (which, according to the previous item, is defined to be equal to one user unit), based on what the SVG user agent determines is the size of a *px* unit (possibly passed from the parent processor or environment at initialization time). For example, suppose that the user agent can determine from its environment that "1px" corresponds to "0.2822222mm" (i.e., 90dpi). Then, for all processing of SVG content:

  - "1pt" equals "1.25px" (and therefore 1.25 user units)
  - "1pc" equals "15px" (and therefore 15 user units)
  - "1mm" would be "3.543307px" (3.543307 user units)
  - "1cm" equals "35.43307px" (and therefore 35.43307 user units)
  - "1in" equals "90px" (and therefore 90 user units)

Note that use of *px* units or any other absolute unit identifiers can cause inconsistent visual results on different viewing environments since the size of "1px" may map to a different number of user units on different systems; thus, absolute units identifiers are only recommended for the **width** and the **height** on outermost **'svg'** elements and situations where the content contains no transformations and it is desirable to specify values relative to the device pixel grid or to a particular real world unit size.

For percentage values that are defined to be relative to the size of viewport:

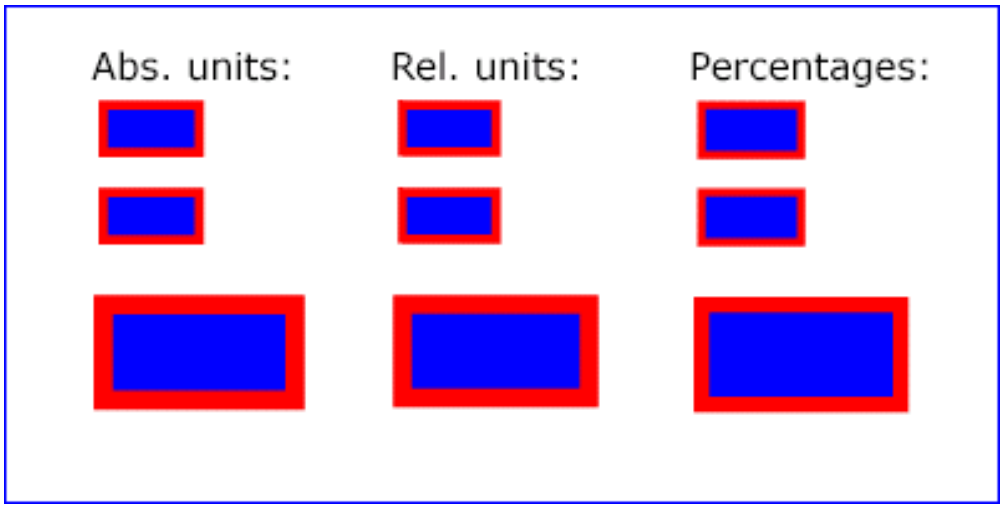
- For any x-coordinate value or width value expressed as a percentage of the viewport, the value to use is the specified percentage of the *actual-width* in user units for the nearest containing

viewport, where *actual-width* is the width dimension of the viewport element within the user coordinate system for the viewport element.

- For any y-coordinate value or height value expressed as a percentage of the viewport, the value to use is the specified percentage of the *actual-height* in user units for the nearest containing viewport, where *actual-height* is the height dimension of the viewport element within the user coordinate system for the viewport element.
- For any other length value expressed as a percentage of the viewport, the percentage is calculated as the specified percentage of `sqrt((`*`actual-width`*`)**2 + (`*`actual-height`*`)**2))/sqrt(2)`.

Example Units below illustrates some of the processing rules for different types of units.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="200px" viewBox="0 0 4000 2000"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example Units</title>
  <desc>Illustrates various units options</desc>

  <!-- Frame the picture -->
  <rect x="5" y="5" width="3990" height="1990"
        fill="none" stroke="blue" stroke-width="10"/>

  <g fill="blue" stroke="red" font-family="Verdana" font-size="150">
    <!-- Absolute unit specifiers -->
    <g transform="translate(400,0)">
      <text x="-50" y="300" fill="black" stroke="none">Abs. units:</text>
      <rect x="0" y="400" width="4in" height="2in" stroke-width=".4in"/>
      <rect x="0" y="750" width="384" height="192" stroke-width="38.4"/>
      <g transform="scale(2)">
        <rect x="0" y="600" width="4in" height="2in" stroke-width=".4in"/>
      </g>
    </g>

    <!-- Relative unit specifiers -->
    <g transform="translate(1600,0)">
      <text x="-50" y="300" fill="black" stroke="none">Rel. units:</text>
      <rect x="0" y="400" width="2.5em" height="1.25em" stroke-width=".25em"/>
      <rect x="0" y="750" width="375" height="187.5" stroke-width="37.5"/>
      <g transform="scale(2)">
        <rect x="0" y="600" width="2.5em" height="1.25em" stroke-width=".25em"/>
      </g>
    </g>

    <!-- Percentages -->
    <g transform="translate(2800,0)">
      <text x="-50" y="300" fill="black" stroke="none">Percentages:</text>
      <rect x="0" y="400" width="10%" height="10%" stroke-width="1%"/>
      <rect x="0" y="750" width="400" height="200" stroke-width="31.62"/>
      <g transform="scale(2)">
        <rect x="0" y="600" width="10%" height="10%" stroke-width="1%"/>
      </g>
    </g>
  </g>
</svg>
```




***Example Units***

*View this example as SVG (SVG-enabled browsers only)*

The three rectangles on the left demonstrate the use of one of the absolute unit identifiers, the "in" unit (inch). The reference image above was generated on a 96dpi system (i.e., 1 inch = 96 pixels). Therefore, the topmost rectangle, which is specified in inches, is exactly the same size as the middle rectangle, which is specified in user units such that there are 96 user units for each corresponding inch in the topmost rectangle. (Note: on systems with different screen resolutions, the top and middle rectangles will likely be rendered at different sizes.) The bottom rectangle of the group illustrates what happens when values specified in inches are scaled.

The three rectangles in the middle demonstrate the use of one of the relative unit identifiers, the "em" unit. Because the **'font-size'** property has been set to **150** on the outermost **'g'** element, each "em" unit is equal to 150 user units. The topmost rectangle, which is specified in "em" units, is exactly the same size as the middle rectangle, which is specified in user units such that there are 150 user units for each corresponding "em" unit in the topmost rectangle. The bottom rectangle of the group illustrates what happens when values specified in "em" units are scaled.

The three rectangles on the right demonstrate the use of percentages. Note that the width and height of the viewport in the user coordinate system for the viewport element (in this case, the outermost **'svg'** element) are 4000 and 2000, respectively, because processing the **viewBox** attribute results in a transformed user coordinate system. The topmost rectangle, which is specified in percentage units, is exactly the same size as the middle rectangle, which is specified in equivalent user units. In particular, note that the **'stroke-width'** property in the middle rectangle is set to 1% of the `sqrt((`*`actual-width`*`)**2 + (`*`actual-height`*`)**2)) / sqrt(2)`, which in this case is .01*sqrt(4000*4000+2000*2000)/sqrt(2), or 31.62. The bottom rectangle of the group illustrates what happens when values specified in percentage units are scaled.

## 7.11 Object bounding box units

The following elements offer the option of expressing coordinate values and lengths as fractions (and, in some cases, percentages) of the **bounding box** (via keyword **objectBoundingBox**) on a given element:

| Element | Attribute | Effect |
|---|---|---|
| 'linearGradient' | gradientUnits="**objectBoundingBox**" | Indicates that the attributes which specify the gradient vector (x1, y1, x2, y2) represent fractions or percentages of the bounding box of the element to which the gradient is applied. |
| 'radialGradient' | gradientUnits="**objectBoundingBox**" | Indicates that the attributes which specify the center (cx, cy), the radius (r) and focus (fx, fy) represent fractions or percentages of the bounding box of the element to which the gradient is applied. |
| 'pattern' | patternUnits="**objectBoundingBox**" | Indicates that the attributes which define how to tile the pattern (x, y, width, height) are established using the bounding box of the element to which the pattern is applied. |
| 'pattern' | patternContentUnits="**objectBoundingBox**" | Indicates that the user coordinate system for the contents of the pattern is established using the bounding box of the element to which the pattern is applied. |
| 'clipPath' | clipPathUnits="**objectBoundingBox**" | Indicates that the user coordinate system for the contents of the 'clipPath' element is established using the bounding box of the element to which the clipping path is applied. |
| 'mask' | maskUnits="**objectBoundingBox**" | Indicates that the attributes which define the masking region (x, y, width, height) is established using the bounding box of the element to which the mask is applied. |
| 'mask' | maskContentUnits="**objectBoundingBox**" | Indicates that the user coordinate system for the contents of the 'mask' element are established using the bounding box of the element to which the mask is applied. |
| 'filter' | filterUnits="**objectBoundingBox**" | Indicates that the attributes which define the filter effects region (x, y, width, height) represent fractions or percentages of the bounding box of the element to which the filter is applied. |

| | | |
|---|---|---|
| **'filter'** | **primitiveUnits**="**objectBoundingBox**" | Indicates that the various length values within the filter primitives represent fractions or percentages of the bounding box of the element to which the filter is applied. |

In the discussion that follows, the term **applicable element** is the element to which the given effect applies. For gradients and patterns, the applicable element is the graphics element which has its **'fill'** or **'stroke'** property referencing the given gradient or pattern. (See Inheritance of Painting Properties. For special rules concerning text elements, see the discussion of object bounding box units and text elements.) For clipping paths, masks and filters, the applicable element can be either a container element or a graphics element.

When keyword **objectBoundingBox** is used, then the effect is as if a supplemental transformation matrix were inserted into the list of nested transformation matrices to create a new user coordinate system.

First, the (**minx**,**miny**) and (**maxx**,**maxy**) coordinates are determined for the applicable element and all of its descendants. The values **minx**, **miny**, **maxx** and **maxy** are determined by computing the maximum extent of the shape of the element in X and Y with respect to the user coordinate system for the applicable element. The bounding box is the tightest fitting rectangle aligned with the axes of the applicable element's user coordinate system that entirely encloses the applicable element and its descendants. The bounding box is computed exclusive of any values for clipping, masking, filter effects, opacity and stroke-width. For curved shapes, the bounding box encloses all portions of the shape, not just end points. For **'text'** elements, for the purposes of the bounding box calculation, each glyph is treated as a separate graphics element. The calculations assume that all glyphs occupy the full glyph cell. For example, for horizontal text, the calculations assume that each glyph extends vertically to the full ascent and descent values for the font.

Then, coordinate (0,0) in the new user coordinate system is mapped to the (minx,miny) corner of the tight bounding box within the user coordinate system of the applicable element and coordinate (1,1) in the new user coordinate system is mapped to the (maxx,maxy) corner of the tight bounding box of the applicable element. In most situations, the following transformation matrix produces the correct effect:

```
[ (maxx-minx) 0 0 (maxy-miny) minx miny ]
```

When percentages are used with attributes that define the gradient vector, the pattern tile, the filter region or the masking region, a percentage represents the same value as the corresponding decimal value (e.g., 50% means the same as 0.5). If percentages are used within the content of a **'pattern'**, **'clipPath'**, **'mask'** or **'filter'** element, these values are treated according to the processing rules for percentages as defined in Units.

Any numeric value can be specified for values expressed as a fraction or percentage of object bounding box units. In particular, fractions less are zero or greater than one and percentages less than 0% or greater than 100% can be specified.

Keyword **objectBoundingBox** should not be used when the geometry of the applicable element has no width or no height, such as the case of a horizontal or vertical line, even when the line has actual

thickness when viewed due to having a non-zero stroke width since stroke width is ignored for bounding box calculations. When the geometry of the applicable element has no width or height and **objectBoundingBox** is specified, then the given effect (e.g., a gradient or a filter) will be ignored.

## 7.12 DOM interfaces

The following interfaces are defined below: SVGPoint, SVGPointList, SVGMatrix, SVGTransform, SVGTransformList, SVGAnimatedTransformList, SVGPreserveAspectRatio, SVGAnimatedPreserveAspectRatio.

### Interface SVGPoint

Many of the SVG DOM interfaces refer to objects of class **SVGPoint**. An **SVGPoint** is an (x,y) coordinate pair. When used in matrix operations, an **SVGPoint** is treated as a vector of the form:

```
[x]
[y]
[1]
```

**IDL Definition**

```
interface SVGPoint {

           attribute float x;
                       // raises DOMException on setting
           attribute float y;
                       // raises DOMException on setting

  SVGPoint matrixTransform ( in SVGMatrix matrix );
};
```

**Attributes**

**float x**

The x coordinate.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**float y**

The y coordinate.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**Methods**

**matrixTransform**

Applies a 2x3 matrix transformation on this SVGPoint object and returns a new,

transformed SVGPoint object:

```
newpoint = matrix * thispoint
```

**Parameters**

in SVGMatrix matrix  The matrix which is to be applied to this SVGPoint object.

**Return value**

SVGPoint  A new SVGPoint object.

**No Exceptions**

## Interface SVGPointList

This interface defines a list of **SVGPoint** objects.

**SVGPointList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

**IDL Definition**

```
interface SVGPointList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  SVGPoint initialize ( in SVGPoint newItem )
                 raises( DOMException, SVGException );
  SVGPoint getItem ( in unsigned long index )
                 raises( DOMException );
  SVGPoint insertItemBefore ( in SVGPoint newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGPoint replaceItem ( in SVGPoint newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGPoint removeItem ( in unsigned long index )
                 raises( DOMException );
  SVGPoint appendItem ( in SVGPoint newItem )
                 raises( DOMException, SVGException );
};
```

**Attributes**

**readonly unsigned long numberOfItems**

The number of items in the list.

**Methods**

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

DOMException  NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single

item specified by the parameter.

**Parameters**

| | |
|---|---|
| in SVGPoint newItem | The item which should become the only member of the list. |

**Return value**

SVGPoint The item being inserted into the list.

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**getItem**

Returns the specified item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item from the list which is to be returned. The first item is number 0. |

**Return value**

SVGPoint The selected item.

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGPoint newItem | The item which is to be inserted into the list. |
| in unsigned long index | The index of the item before which the new item is to be inserted. The first item is number 0.<br>If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list. |

**Return value**

SVGPoint The inserted item.

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGPoint newItem | The item which is to be inserted into the list. |
| in unsigned long index | The index of the item which is to be replaced. The first item is number 0. |

**Return value**

SVGPoint The inserted item.

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.<br>INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**removeItem**

Removes an existing item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item which is to be removed. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGPoint | The removed item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.<br>INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGPoint newItem | The item which is to be inserted into the list. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGPoint | The inserted item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

## Interface SVGMatrix

Many of SVG's graphics operations utilize 2x3 matrices of the form:

```
[a c e]
[b d f]
```

which, when expanded into a 3x3 matrix for the purposes of matrix arithmetic, become:

```
[a c e]
[b d f]
[0 0 1]
```

**IDL Definition**

```
interface SVGMatrix {

           attribute float a;
                       // raises DOMException on setting
           attribute float b;
                       // raises DOMException on setting
           attribute float c;
                       // raises DOMException on setting
           attribute float d;
                       // raises DOMException on setting
           attribute float e;
                       // raises DOMException on setting
           attribute float f;
                       // raises DOMException on setting

  SVGMatrix multiply ( in SVGMatrix secondMatrix );
  SVGMatrix inverse (  )
                raises( SVGException );
  SVGMatrix translate ( in float x, in float y );
  SVGMatrix scale ( in float scaleFactor );
  SVGMatrix scaleNonUniform ( in float scaleFactorX, in float scaleFactorY );
  SVGMatrix rotate ( in float angle );
  SVGMatrix rotateFromVector ( in float x, in float y )
                raises( SVGException );
  SVGMatrix flipX (  );
  SVGMatrix flipY (  );
  SVGMatrix skewX ( in float angle );
  SVGMatrix skewY ( in float angle );
};
```

**Attributes**

**float a**

The **a** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float b**

The **b** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float c**

The **c** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float d**

The **d** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float e**

The **e** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float f**

The **f** component of the matrix.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**Methods**

**multiply**

Performs matrix multiplication. This matrix is post-multiplied by another matrix, returning the resulting new matrix.

**Parameters**

in SVGMatrix secondMatrix The matrix which is post-multiplied to this matrix.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**inverse**

Returns the inverse matrix.

**No Parameters**

**Return value**

SVGMatrix The inverse matrix.

**Exceptions**

SVGException SVG_MATRIX_NOT_INVERTABLE: Raised if this matrix is not invertable.

**translate**

Post-multiplies a translation transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float x The distance to translate along the x-axis.

in float y The distance to translate along the y-axis.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**scale**

Post-multiplies a uniform scale transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float scaleFactor Scale factor in both X and Y.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**scaleNonUniform**

Post-multiplies a non-uniform scale transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float scaleFactorX Scale factor in X.

in float scaleFactorY Scale factor in Y.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**rotate**

Post-multiplies a rotation transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float angle Rotation angle.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**rotateFromVector**

Post-multiplies a rotation transformation on the current matrix and returns the resulting matrix. The rotation angle is determined by taking (+/-) atan(y/x). The direction of the vector (x,y) determines whether the positive or negative angle value is used.

**Parameters**

in float x The X coordinate of the vector (x,y). Must not be zero.

in float y The Y coordinate of the vector (x,y). Must not be zero.

**Return value**

SVGMatrix The resulting matrix.

**Exceptions**

SVGException SVG_INVALID_VALUE_ERR: Raised if one of the parameters has an invalid value.

**flipX**

Post-multiplies the transformation [-1 0 0 1 0 0] and returns the resulting matrix.

**No Parameters**

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**flipY**

Post-multiplies the transformation [1 0 0 -1 0 0] and returns the resulting matrix.

**No Parameters**

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**skewX**

Post-multiplies a skewX transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float angle Skew angle.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

**skewY**

Post-multiplies a skewY transformation on the current matrix and returns the resulting matrix.

**Parameters**

in float angle Skew angle.

**Return value**

SVGMatrix The resulting matrix.

**No Exceptions**

## Interface SVGTransform

**SVGTransform** is the interface for one of the component transformations within a **SVGTransformList**; thus, a **SVGTransform** object corresponds to a single component (e.g., "scale(..)" or "matrix(...)") within

a **transform** attribute specification.

### IDL Definition

```
interface SVGTransform {

  // Transform Types
  const unsigned short SVG_TRANSFORM_UNKNOWN   = 0;
  const unsigned short SVG_TRANSFORM_MATRIX    = 1;
  const unsigned short SVG_TRANSFORM_TRANSLATE = 2;
  const unsigned short SVG_TRANSFORM_SCALE     = 3;
  const unsigned short SVG_TRANSFORM_ROTATE    = 4;
  const unsigned short SVG_TRANSFORM_SKEWX     = 5;
  const unsigned short SVG_TRANSFORM_SKEWY     = 6;

  readonly attribute unsigned short type;
  readonly attribute SVGMatrix matrix;
  readonly attribute float angle;

  void setMatrix ( in SVGMatrix matrix );
  void setTranslate ( in float tx, in float ty );
  void setScale ( in float sx, in float sy );
  void setRotate ( in float angle, in float cx, in float cy );
  void setSkewX ( in float angle );
  void setSkewY ( in float angle );
};
```

### Definition group Transform Types

#### Defined constants

| | |
|---|---|
| SVG_TRANSFORM_UNKNOWN | The unit type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_TRANSFORM_MATRIX | A "matrix(...)" transformation. |
| SVG_TRANSFORM_TRANSLATE | A "translate(...)" transformation. |
| SVG_TRANSFORM_SCALE | A "scale(...)" transformation. |
| SVG_TRANSFORM_ROTATE | A "rotate(...)" transformation. |
| SVG_TRANSFORM_SKEWX | A "skewX(...)" transformation. |
| SVG_TRANSFORM_SKEWY | A "skewY(...)" transformation. |

### Attributes

**readonly unsigned short type**

The type of the value as specified by one of the constants specified above.

**readonly SVGMatrix matrix**

The matrix that represents this transformation.
For SVG_TRANSFORM_MATRIX, the matrix contains the a, b, c, d, e, f values supplied by the user.
For SVG_TRANSFORM_TRANSLATE, e and f represent the translation amounts (a=1,b=0,c=0,d=1).
For SVG_TRANSFORM_SCALE, a and d represent the scale amounts (b=0,c=0,e=0,f=0).
For SVG_TRANSFORM_ROTATE, SVG_TRANSFORM_SKEWX and SVG_TRANSFORM_SKEWY, a, b, c and d represent the matrix which will result in the given transformation (e=0,f=0).

**readonly float angle**

A convenience attribute for SVG_TRANSFORM_ROTATE, SVG_TRANSFORM_SKEWX and SVG_TRANSFORM_SKEWY. It holds the angle that was specified.

For SVG_TRANSFORM_MATRIX, SVG_TRANSFORM_TRANSLATE and SVG_TRANSFORM_SCALE, angle will be zero.

**Methods**

**setMatrix**

Sets the transform type to SVG_TRANSFORM_MATRIX, with parameter matrix defining the new transformation.

**Parameters**

in SVGMatrix matrix The new matrix for the transformation.

**No Return Value**

**No Exceptions**

**setTranslate**

Sets the transform type to SVG_TRANSFORM_TRANSLATE, with parameters tx and ty defining the translation amounts.

**Parameters**

in float tx The translation amount in X.

in float ty The translation amount in Y.

**No Return Value**

**No Exceptions**

**setScale**

Sets the transform type to SVG_TRANSFORM_SCALE, with parameters sx and sy defining the scale amounts.

**Parameters**

in float sx The scale factor in X.

in float sy The scale factor in Y.

**No Return Value**

**No Exceptions**

**setRotate**

Sets the transform type to SVG_TRANSFORM_ROTATE, with parameter angle defining the rotation angle and parameters cx and cy defining the optional centre of rotation.

**Parameters**

in float angle The rotation angle.

in float cx The x coordinate of centre of rotation.

in float cy The y coordinate of centre of rotation.

**No Return Value**

**No Exceptions**

**setSkewX**

Sets the transform type to SVG_TRANSFORM_SKEWX, with parameter angle defining the amount of skew.

**Parameters**

in float angle The skew angle.

**No Return Value**

**No Exceptions**

**setSkewY**

Sets the transform type to SVG_TRANSFORM_SKEWY, with parameter angle defining the amount of skew.

**Parameters**

in float angle The skew angle.

**No Return Value**

**No Exceptions**

## Interface SVGTransformList

This interface defines a list of **SVGTransform** objects.

The **SVGTransformList** and **SVGTransform** interfaces correspond to the various attributes which specify a set of transformations, such as the **transform** attribute which is available for many of SVG's elements.

**SVGTransformList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

### IDL Definition

```
interface SVGTransformList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  SVGTransform initialize ( in SVGTransform newItem )
                 raises( DOMException, SVGException );
  SVGTransform getItem ( in unsigned long index )
                 raises( DOMException );
  SVGTransform insertItemBefore ( in SVGTransform newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGTransform replaceItem ( in SVGTransform newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGTransform removeItem ( in unsigned long index )
                 raises( DOMException );
  SVGTransform appendItem ( in SVGTransform newItem )
                 raises( DOMException, SVGException );
  SVGTransform createSVGTransformFromMatrix ( in SVGMatrix matrix );
  SVGTransform consolidate (  );
};
```

### Attributes

**readonly unsigned long numberOfItems**

The number of items in the list.

### Methods

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter.

**Parameters**

| | |
|---|---|
| in SVGTransform newItem | The item which should become the only member of the list. |

**Return value**

| | |
|---|---|
| SVGTransform | The item being inserted into the list. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**getItem**

Returns the specified item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item from the list which is to be returned. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGTransform | The selected item. |

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGTransform newItem | The item which is to be inserted into the list. |
| in unsigned long index | The index of the item before which the new item is to be inserted. The first item is number 0.<br>If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list. |

**Return value**

| | |
|---|---|
| SVGTransform | The inserted item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGTransform newItem | The item which is to be inserted into the list. |
| in unsigned long index | The index of the item which is to be replaced. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGTransform | The inserted item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.<br>INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**removeItem**

Removes an existing item from the list.

**Parameters**

| | |
|---|---|
| in unsigned long index | The index of the item which is to be removed. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGTransform | The removed item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.<br>INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems. |

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

| | |
|---|---|
| in SVGTransform newItem | The item which is to be inserted into the list. The first item is number 0. |

**Return value**

| | |
|---|---|
| SVGTransform | The inserted item. |

**Exceptions**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified. |
| SVGException | SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list. |

**createSVGTransformFromMatrix**

Creates an **SVGTransform** object which is initialized to transform of type SVG_TRANSFORM_MATRIX and whose values are the given matrix.

**Parameters**

| | |
|---|---|
| in SVGMatrix matrix | The matrix which defines the transformation. |

**Return value**

| | |
|---|---|
| SVGTransform | The returned **SVGTransform** object. |

**No Exceptions**

**consolidate**

Consolidates the list of separate **SVGTransform** objects by multiplying the equivalent transformation matrices together to result in a list consisting of a single **SVGTransform** object of type SVG_TRANSFORM_MATRIX.

**No Parameters**

**Return value**

| | |
|---|---|
| SVGTransform | The resulting **SVGTransform** object which becomes single item in the list. If the list was empty, then a value of null is returned. |

**No Exceptions**

## Interface SVGAnimatedTransformList

Used for the various attributes which specify a set of transformations, such as the **transform** attribute which is available for many of SVG's elements, and which can be animated.

**IDL Definition**

```
interface SVGAnimatedTransformList {

  readonly attribute SVGTransformList baseVal;
  readonly attribute SVGTransformList animVal;
};
```

**Attributes**

**readonly SVGTransformList baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGTransformList animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'baseVal'.

## Interface SVGPreserveAspectRatio

The **SVGPreserveAspectRatio** interface corresponds to the **preserveAspectRatio** attribute, which is available for some of SVG's elements.

**IDL Definition**

```
interface SVGPreserveAspectRatio {

  // Alignment Types
  const unsigned short SVG_PRESERVEASPECTRATIO_UNKNOWN   = 0;
  const unsigned short SVG_PRESERVEASPECTRATIO_NONE     = 1;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMIN = 2;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMIN = 3;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMIN = 4;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMID = 5;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMID = 6;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMID = 7;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMAX = 8;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMAX = 9;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMAX = 10;
  // Meet-or-slice Types
  const unsigned short SVG_MEETORSLICE_UNKNOWN   = 0;
  const unsigned short SVG_MEETORSLICE_MEET  = 1;
  const unsigned short SVG_MEETORSLICE_SLICE = 2;

           attribute unsigned short align;
                      // raises DOMException on setting
           attribute unsigned short meetOrSlice;
                      // raises DOMException on setting
};
```

**Definition group Alignment Types**

**Defined constants**

| | |
|---|---|
| SVG_PRESERVEASPECTRATIO_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_PRESERVEASPECTRATIO_NONE | Corresponds to value 'none' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMINYMIN | Corresponds to value 'xMinYMin' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMIDYMIN | Corresponds to value 'xMidYMin' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMAXYMIN | Corresponds to value 'xMaxYMin' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMINYMID | Corresponds to value 'xMinYMid' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMIDYMID | Corresponds to value 'xMidYMid' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMAXYMID | Corresponds to value 'xMaxYMid' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMINYMAX | Corresponds to value 'xMinYMax' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMIDYMAX | Corresponds to value 'xMidYMax' for attribute **preserveAspectRatio**. |
| SVG_PRESERVEASPECTRATIO_XMAXYMAX | Corresponds to value 'xMaxYMax' for attribute **preserveAspectRatio**. |

**Definition group Meet-or-slice Types**

**Defined constants**

| | |
|---|---|
| SVG_MEETORSLICE_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_MEETORSLICE_MEET | Corresponds to value 'meet' for attribute **preserveAspectRatio**. |
| SVG_MEETORSLICE_SLICE | Corresponds to value 'slice' for attribute **preserveAspectRatio**. |

**Attributes**

**unsigned short align**

The type of the alignment value as specified by one of the constants specified above.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**unsigned short meetOrSlice**

The type of the meet-or-slice value as specified by one of the constants specified above.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Interface SVGAnimatedPreserveAspectRatio

Used for attributes of type SVGPreserveAspectRatio which can be animated.

### IDL Definition

```
interface SVGAnimatedPreserveAspectRatio {

  readonly attribute SVGPreserveAspectRatio baseVal;
  readonly attribute SVGPreserveAspectRatio animVal;
};
```

### Attributes

**readonly SVGPreserveAspectRatio baseVal**

The base value of the given attribute before applying any animations.

**readonly SVGPreserveAspectRatio animVal**

If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same

---

04 September 2001

# 8 Paths

## Contents



## 8.1 Introduction

Paths represent the outline of a shape which can be filled, stroked, used as a clipping path, or any combination of the three. (See Filling, Stroking and Paint Servers and Clipping, Masking and Compositing.)

A path is described using the concept of a current point. In an analogy with drawing on paper, the current point can be thought of as the location of the pen. The position of the pen can be changed, and the outline of a shape (open or closed) can be traced by dragging the pen in either straight lines or curves.

Paths represent the geometry of the outline of an object, defined in terms of *moveto* (set a new current point), *lineto* (draw a straight line), *curveto* (draw a curve using a cubic Bézier), *arc* (elliptical or circular arc) and *closepath* (close the current shape by drawing a line to the last *moveto*) elements. Compound paths (i.e., a path with multiple subpaths) are possible to allow effects such as "donut holes" in objects.

This chapter describes the syntax, behavior and DOM interfaces for SVG paths. Various implementation notes for SVG paths can be found in 'path' element implementation notes and Elliptical arc implementation notes.

A path is defined in SVG using the 'path' element.

## 8.2 The 'path' element

```
<!ENTITY % pathExt "" >
<!ELEMENT path (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
               %geExt;%pathExt;)*) >
<!ATTLIST path
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  d %PathData; #REQUIRED
  pathLength %Number; #IMPLIED >
```

*Attribute definitions:*

**d = "*path data*"**

The definition of the outline of a shape. See Path data.

*Animatable: yes.* Path data animation is only possible when each path data specification within an animation specification has exactly the same list of path data commands as the **d** attribute. If an animation is specified and the list of path data commands is not the same, then the animation specification is in error (see Error Processing). The animation engine interpolates each parameter to each path data command separately based on the attributes to the given animation element. Flags and booleans are interpolated as fractions between zero and one, with any non-zero value considered to be a value of one/true.

**pathLength = "<number>"**

The author's computation of the total length of the path, in user units. This value is used to calibrate the user agent's own distance-along-a-path calculations with that of the author. The user agent will scale all distance-along-a-path computations by the ratio of **pathLength** to the user agent's own computed value for total path length. **pathLength** potentially affects calculations for text on a path, motion animation and various stroke operations.

A negative value is an error (see Error processing).

*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color; %PresentationAttributes-FillStroke; %PresentationAttributes-Graphics; %PresentationAttributes-Markers;.**

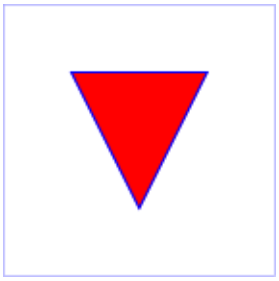
## 8.3 Path data

### 8.3.1 General information about path data

A path is defined by including a **'path'** element which contains a **d="(path data)"** attribute, where the **d** attribute contains the *moveto*, *line*, *curve* (both cubic and quadratic Béziers), *arc* and *closepath* instructions.

Example triangle01 specifies a path in the shape of a triangle. (The **M** indicates a *moveto*, the **L**'s indicate *lineto*'s, and the **z** indicates a *closepath*).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example triangle01- simple example of a 'path'</title>
  <desc>A path that draws a triangle</desc>
  <rect x="1" y="1" width="398" height="398"
        fill="none" stroke="blue" />
  <path d="M 100 100 L 300 100 L 200 300 z"
        fill="red" stroke="blue" stroke-width="3" />
</svg>
```



***Example triangle01***

*View this example as SVG (SVG-enabled browsers only)*

Path data can contain newline characters and thus can be broken up into multiple lines to improve readability. Because of line length limitations with certain related tools, it is recommended that SVG generators split long path data strings across multiple lines, with each line not exceeding 255 characters. Also note that newline characters are only allowed at certain places within path data.

The syntax of path data is concise in order to allow for minimal file size and efficient downloads, since many SVG files will be dominated by their path data. Some of the ways that SVG attempts to minimize the size of path data are as follows:

- All instructions are expressed as one character (e.g., a *moveto* is expressed as an **M**).
- Superfluous white space and separators such as commas can be eliminated (e.g., "M 100 100 L 200 200" contains unnecessary spaces and could be expressed more compactly as "M100 100L200 200").
- The command letter can be eliminated on subsequent commands if the same command is used multiple times in a row (e.g., you can drop the second "L" in "M 100 200 L 200 100 L -100 -200" and use "M 100 200 L 200 100 -100 -200" instead).
- Relative versions of all commands are available (uppercase means absolute coordinates, lowercase means relative coordinates).
- Alternate forms of *lineto* are available to optimize the special cases of horizontal and vertical lines (absolute and relative).
- Alternate forms of *curve* are available to optimize the special cases where some of the control points on the current segment can be determined automatically from the control points on the previous segment.

The path data syntax is a prefix notation (i.e., commands followed by parameters). The only allowable decimal point is a Unicode [UNICODE] FULL STOP (".") character (also referred to in Unicode as PERIOD, dot and decimal point) and no other delimiter characters are allowed. (For example, the following is an invalid numeric value in a path data stream: "13,000.56". Instead, say: "13000.56".)

For the relative versions of the commands, all coordinate values are relative to the current point at the start of the command.

In the tables below, the following notation is used:

- (): grouping of parameters
- +: 1 or more of the given parameter(s) is required

The following sections list the commands.

### 8.3.2 The "moveto" commands

The "moveto" commands (**M** or **m**) establish a new current point. The effect is as if the "pen" were lifted and moved to a new location. A path data segment must begin with a "moveto" command. Subsequent "moveto" commands (i.e., when the "moveto" is not the first command) represent the start of a new *subpath*:

| Command | Name | Parameters | Description |
|---|---|---|---|
| **M** (absolute)<br>**m** (relative) | moveto | (x y)+ | Start a new sub-path at the given (x,y) coordinate. **M** (uppercase) indicates that absolute coordinates will follow; **m** (lowercase) indicates that relative coordinates will follow. If a relative moveto (**m**) appears as the first element of the path, then it is treated as a pair of absolute coordinates. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands. |

### 8.3.3 The "closepath" command

The "closepath" (**Z** or **z**) ends the current subpath and causes an automatic straight line to be drawn from the current point to the initial point of

the current subpath. If a "closepath" is followed immediately by a "moveto", then the "moveto" identifies the start point of the next subpath. If a "closepath" is followed immediately by any other command, then the next subpath starts at the same initial point as the current subpath.

When a subpath ends in a "closepath," it differs in behavior from what happens when "manually" closing a subpath via a "lineto" command in how 'stroke-linejoin' and 'stroke-linecap' are implemented. With "closepath", the end of the final segment of the subpath is "joined" with the start of the initial segment of the subpath using the current value of 'stroke-linejoin'. If you instead "manually" close the subpath via a "lineto" command, the start of the first segment and the end of the last segment are not joined but instead are each capped using the current value of 'stroke-linecap'. At the end of the command, the new current point is set to the initial point of the current subpath.

| Command | Name | Parameters | Description |
|---|---|---|---|
| **Z** or **z** | closepath | (none) | Close the current subpath by drawing a straight line from the current point to current subpath's initial point. |

### 8.3.4 The "lineto" commands

The various "lineto" commands draw straight lines from the current point to a new point:

| Command | Name | Parameters | Description |
|---|---|---|---|
| **L** (absolute) **l** (relative) | lineto | (x y)+ | Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. **L** (uppercase) indicates that absolute coordinates will follow; **l** (lowercase) indicates that relative coordinates will follow. A number of coordinates pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final set of coordinates provided. |
| **H** (absolute) **h** (relative) | horizontal lineto | x+ | Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). **H** (uppercase) indicates that absolute coordinates will follow; **h** (lowercase) indicates that relative coordinates will follow. Multiple x values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (x, cpy) for the final value of x. |
| **V** (absolute) **v** (relative) | vertical lineto | y+ | Draws a vertical line from the current point (cpx, cpy) to (cpx, y). **V** (uppercase) indicates that absolute coordinates will follow; **v** (lowercase) indicates that relative coordinates will follow. Multiple y values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (cpx, y) for the final value of y. |

### 8.3.5 The curve commands

These three groups of commands draw curves:

- Cubic Bézier commands (**C**, **c**, **S** and **s**). A cubic Bézier segment is defined by a start point, an end point, and two control points.
- Quadratic Bézier commands (**Q**, **q**, **T** and **t**). A quadratic Bézier segment is defined by a start point, an end point, and one control point.
- Elliptical arc commands (**A** and **a**). An elliptical arc segment draws a segment of an ellipse.

### 8.3.6 The cubic Bézier curve commands

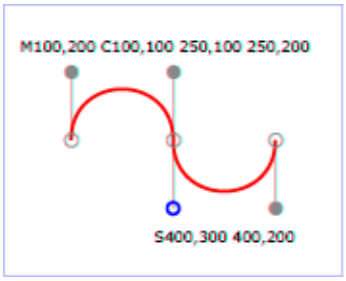
The cubic Bézier commands are as follows:

| Command | Name | Parameters | Description |
|---|---|---|---|
| **C** (absolute) **c** (relative) | curveto | (x1 y1 x2 y2 x y)+ | Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. **C** (uppercase) indicates that absolute coordinates will follow; **c** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier. |

| | | | |
|---|---|---|---|
| **S** (absolute)<br>**s** (relative) | shorthand/smooth curveto | (x2 y2 x y)+ | Draws a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not an C, c, S or s, assume the first control point is coincident with the current point.) (x2,y2) is the second control point (i.e., the control point at the end of the curve). **S** (uppercase) indicates that absolute coordinates will follow; **s** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier. |

Example cubic01 shows some simple uses of cubic Bézier commands within a path. The example uses an internal CSS style sheet to assign styling properties. Note that the control point for the "S" command is computed automatically as the reflection of the control point for the previous "C" command relative to the start point of the "S" command.

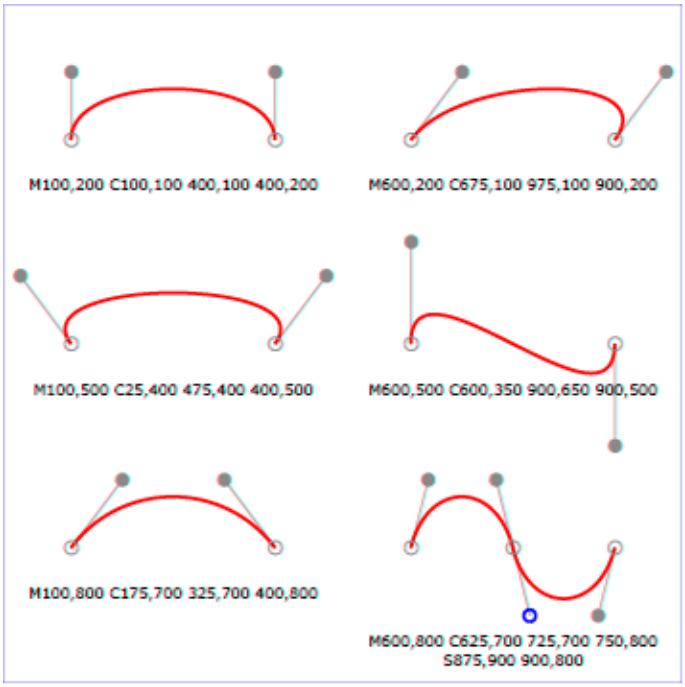
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="4cm" viewBox="0 0 500 400"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example cubic01- cubic Bézier commands in path data</title>
  <desc>Picture showing a simple example of path data
        using both a "C" and an "S" command,
        along with annotations showing the control points
        and end points</desc>
  <style type="text/css"><![CDATA[
    .Border { fill:none; stroke:blue; stroke-width:1 }
    .Connect { fill:none; stroke:#888888; stroke-width:2 }
    .SamplePath { fill:none; stroke:red; stroke-width:5 }
    .EndPoint { fill:none; stroke:#888888; stroke-width:2 }
    .CtlPoint { fill:#888888; stroke:none }
    .AutoCtlPoint { fill:none; stroke:blue; stroke-width:4 }
    .Label { font-size:22; font-family:Verdana }
  ]]></style>

  <rect class="Border" x="1" y="1" width="498" height="398" />

  <polyline class="Connect" points="100,200 100,100" />
  <polyline class="Connect" points="250,100 250,200" />
  <polyline class="Connect" points="250,200 250,300" />
  <polyline class="Connect" points="400,300 400,200" />
  <path class="SamplePath" d="M100,200 C100,100 250,100 250,200
                              S400,300 400,200" />
  <circle class="EndPoint" cx="100" cy="200" r="10" />
  <circle class="EndPoint" cx="250" cy="200" r="10" />
  <circle class="EndPoint" cx="400" cy="200" r="10" />
  <circle class="CtlPoint" cx="100" cy="100" r="10" />
  <circle class="CtlPoint" cx="250" cy="100" r="10" />
  <circle class="CtlPoint" cx="400" cy="300" r="10" />
  <circle class="AutoCtlPoint" cx="250" cy="300" r="9" />
  <text class="Label" x="25" y="70">M100,200 C100,100 250,100 250,200</text>
  <text class="Label" x="325" y="350"
        style="text-anchor:middle">S400,300 400,200</text>
</svg>
```




***Example cubic01***

*View this example as SVG (SVG- and CSS-enabled browsers only)*

The following picture shows some how cubic Bézier curves change their shape depending on the position of the control points. The first five examples illustrate a single cubic Bézier path segment. The example at the lower right shows a "C" command followed by an "S" command.




*View this example as SVG (SVG-enabled browsers only)*

### 8.3.7 The quadratic Bézier curve commands

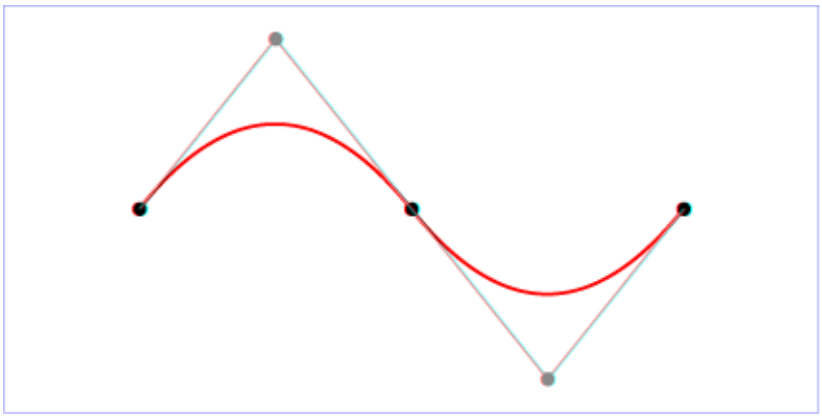
The quadratic Bézier commands are as follows:

| Command | Name | Parameters | Description |
|---|---|---|---|
| **Q** (absolute)<br>**q** (relative) | quadratic Bézier curveto | (x1 y1 x y)+ | Draws a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point. **Q** (uppercase) indicates that absolute coordinates will follow; **q** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier. |
| **T** (absolute)<br>**t** (relative) | Shorthand/smooth quadratic Bézier curveto | (x y)+ | Draws a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a Q, q, T or t, assume the control point is coincident with the current point.) **T** (uppercase) indicates that absolute coordinates will follow; **t** (lowercase) indicates that relative coordinates will follow. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier. |

Example quad01 shows some simple uses of quadratic Bézier commands within a path. Note that the control point for the "T" command is computed automatically as the reflection of the control point for the previous "Q" command relative to the start point of the "T" command.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="6cm" viewBox="0 0 1200 600"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example quad01 - quadratic Bezier commands in path data</title>
  <desc>Picture showing a "Q" a "T" command,
        along with annotations showing the control points
        and end points</desc>
  <rect x="1" y="1" width="1198" height="598"
        fill="none" stroke="blue" stroke-width="1" />

  <path d="M200,300 Q400,50 600,300 T1000,300"
```

```
        fill="none" stroke="red" stroke-width="5"  />
  <!-- End points -->
  <g fill="black" >
    <circle cx="200" cy="300" r="10"/>
    <circle cx="600" cy="300" r="10"/>
    <circle cx="1000" cy="300" r="10"/>
  </g>
  <!-- Control points and lines from end points to control points -->
  <g fill="#888888" >
    <circle cx="400" cy="50" r="10"/>
    <circle cx="800" cy="550" r="10"/>
  </g>
  <path d="M200,300 L400,50 L600,300
           L800,550 L1000,300"
        fill="none" stroke="#888888" stroke-width="2" />
</svg>
```



***Example quad01***

*View this example as SVG (SVG-enabled browsers only)*

### 8.3.8 The elliptical arc curve commands

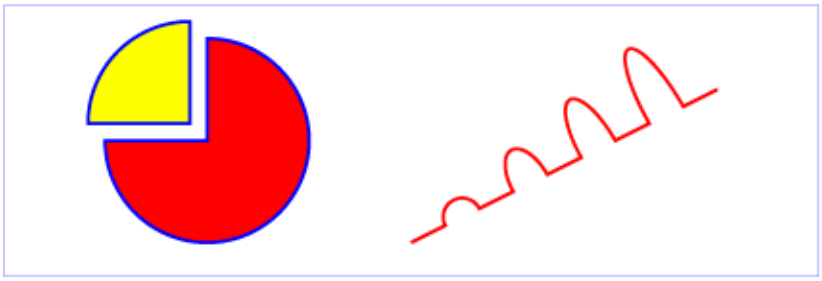
The elliptical arc commands are as follows:

| Command | Name | Parameters | Description |
|---|---|---|---|
| **A** (absolute)<br>**a** (relative) | elliptical arc | (rx ry x-axis-rotation large-arc-flag sweep-flag x y)+ | Draws an elliptical arc from the current point to (**x**, **y**). The size and orientation of the ellipse are defined by two radii (**rx**, **ry**) and an **x-axis-rotation**, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center (**cx**, **cy**) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. **large-arc-flag** and **sweep-flag** contribute to the automatic calculations and help determine how the arc is drawn. |

Example arcs01 shows some simple uses of arc commands within a path.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="5.25cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example arcs01 - arc commands in path data</title>
  <desc>Picture of a pie chart with two pie wedges and
        a picture of a line with arc blips</desc>
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="1" />

  <path d="M300,200 h-150 a150,150 0 1,0 150,-150 z"
        fill="red" stroke="blue" stroke-width="5" />
  <path d="M275,175 v-150 a150,150 0 0,0 -150,150 z"
        fill="yellow" stroke="blue" stroke-width="5" />
```

```
  <path d="M600,350 l 50,-25
           a25,25 -30 0,1 50,-25 l 50,-25
           a25,50 -30 0,1 50,-25 l 50,-25
           a25,75 -30 0,1 50,-25 l 50,-25
           a25,100 -30 0,1 50,-25 l 50,-25"
         fill="none" stroke="red" stroke-width="5"  />
</svg>
```



***Example arcs01***

*View this example as SVG (SVG-enabled browsers only)*

The elliptical arc command draws a section of an ellipse which meets the following constraints:

- the arc starts at the current point
- the arc ends at point (**x**, **y**)
- the ellipse has the two radii (**rx**, **ry**)
- the x-axis of the ellipse is rotated by **x-axis-rotation** relative to the x-axis of the current coordinate system.

For most situations, there are actually four different arcs (two different ellipses, each with two different arc sweeps) that satisfy these constraints. **large-arc-flag** and **sweep-flag** indicate which one of the four arcs are drawn, as follows:

- Of the four candidate arc sweeps, two will represent an arc sweep of greater than or equal to 180 degrees (the "large-arc"), and two will represent an arc sweep of less than or equal to 180 degrees (the "small-arc"). If **large-arc-flag** is '1', then one of the two larger arc sweeps will be chosen; otherwise, if **large-arc-flag** is '0', one of the smaller arc sweeps will be chosen,
- If **sweep-flag** is '1', then the arc will be drawn in a "positive-angle" direction (i.e., the ellipse formula x=**cx**+**rx***cos(theta) and y=**cy**+**ry***sin(theta) is evaluated such that theta starts at an angle corresponding to the current point and increases positively until the arc reaches (x,y)). A value of 0 causes the arc to be drawn in a "negative-angle" direction (i.e., theta starts at an angle value corresponding to the current point and decreases until the arc reaches (x,y)).

The following illustrates the four combinations of **large-arc-flag** and **sweep-flag** and the four different arcs that will be drawn based on the values of these flags. For each case, the following path data command was used:

```
<path d="M 125,75 a100,50 0 ?,? 100,50"
      style="fill:none; stroke:red; stroke-width:6"/>
```

where "?,?" is replaced by "0,0" "0,1" "1,0" and "1,1" to generate the four possible cases.


*View this example as SVG (SVG-enabled browsers only)*

Refer to Elliptical arc implementation notes for detailed implementation notes for the path data elliptical arc commands.

### 8.3.9 The grammar for path data

The following notation is used in the Backus-Naur Form (BNF) description of the grammar for path data:

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- (): grouping
- |: separates alternatives
- double quotes surround literals

The following is the BNF for SVG paths.

```
svg-path:
    wsp* moveto-drawto-command-groups? wsp*

moveto-drawto-command-groups:
    moveto-drawto-command-group
    | moveto-drawto-command-group wsp* moveto-drawto-command-groups

moveto-drawto-command-group:
    moveto wsp* drawto-commands?

drawto-commands:
    drawto-command
    | drawto-command wsp* drawto-commands

drawto-command:
    closepath
    | lineto
    | horizontal-lineto
    | vertical-lineto
    | curveto
    | smooth-curveto
    | quadratic-bezier-curveto
    | smooth-quadratic-bezier-curveto
    | elliptical-arc

moveto:
    ( "M" | "m" ) wsp* moveto-argument-sequence

moveto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? lineto-argument-sequence

closepath:
    ("Z" | "z")

lineto:
    ( "L" | "l" ) wsp* lineto-argument-sequence

lineto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? lineto-argument-sequence

horizontal-lineto:
    ( "H" | "h" ) wsp* horizontal-lineto-argument-sequence

horizontal-lineto-argument-sequence:
    coordinate
    | coordinate comma-wsp? horizontal-lineto-argument-sequence

vertical-lineto:
    ( "V" | "v" ) wsp* vertical-lineto-argument-sequence

vertical-lineto-argument-sequence:
    coordinate
    | coordinate comma-wsp? vertical-lineto-argument-sequence

curveto:
    ( "C" | "c" ) wsp* curveto-argument-sequence

curveto-argument-sequence:
    curveto-argument
    | curveto-argument comma-wsp? curveto-argument-sequence

curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair comma-wsp? coordinate-pair

smooth-curveto:
    ( "S" | "s" ) wsp* smooth-curveto-argument-sequence

smooth-curveto-argument-sequence:
    smooth-curveto-argument
    | smooth-curveto-argument comma-wsp? smooth-curveto-argument-sequence
```

```
smooth-curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair

quadratic-bezier-curveto:
    ( "Q" | "q" ) wsp* quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument-sequence:
    quadratic-bezier-curveto-argument
    | quadratic-bezier-curveto-argument comma-wsp?
        quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair

smooth-quadratic-bezier-curveto:
    ( "T" | "t" ) wsp* smooth-quadratic-bezier-curveto-argument-sequence

smooth-quadratic-bezier-curveto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? smooth-quadratic-bezier-curveto-argument-sequence

elliptical-arc:
    ( "A" | "a" ) wsp* elliptical-arc-argument-sequence

elliptical-arc-argument-sequence:
    elliptical-arc-argument
    | elliptical-arc-argument comma-wsp? elliptical-arc-argument-sequence

elliptical-arc-argument:
    nonnegative-number comma-wsp? nonnegative-number comma-wsp?
        number comma-wsp flag comma-wsp flag comma-wsp coordinate-pair

coordinate-pair:
    coordinate comma-wsp? coordinate

coordinate:
    number

nonnegative-number:
    integer-constant
    | floating-point-constant

number:
    sign? integer-constant
    | sign? floating-point-constant

flag:
    "0" | "1"

comma-wsp:
    (wsp+ comma? wsp*) | (comma wsp*)

comma:
    ","

integer-constant:
    digit-sequence

floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent

fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."

exponent:
    ( "e" | "E" ) sign? digit-sequence

sign:
    "+" | "-"

digit-sequence:
    digit
    | digit digit-sequence

digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

wsp:
    (#x20 | #x9 | #xD | #xA)
```

The processing of the BNF must consume as much of a given BNF production as possible, stopping at the point when a character is encountered which no longer satisfies the production. Thus, in the string "M 100-200", the first coordinate for the "moveto" consumes the characters "100" and stops upon encountering the minus sign because the minus sign cannot follow a digit in the production of a "coordinate". The result is that the first coordinate will be "100" and the second coordinate will be "-200".

Similarly, for the string "M 0.6.5", the first coordinate of the "moveto" consumes the characters "0.6" and stops upon encountering the second decimal point because the production of a "coordinate" only allows one decimal point. The result is that the first coordinate will be "0.6" and the second coordinate will be ".5".

## 8.4 Distance along a path

Various operations, including text on a path and motion animation and various stroke operations, require that the user agent compute the distance along the geometry of a graphics element, such as a **'path'**.

Exact mathematics exist for computing distance along a path, but the formulas are highly complex and require substantial computation. It is recommended that authoring products and user agents employ algorithms that produce as precise results as possible; however, to accommodate implementation differences and to help distance calculations produce results that approximate author intent, the **pathLength** attribute can be used to provide the author's computation of the total length of the path so that the user agent can scale distance-along-a-path computations by the ratio of **pathLength** to the user agent's own computed value for total path length.

A "moveto" operation within a **'path'** element is defined to have zero length. Only the various "lineto", "curveto" and "arcto" commands contribute to path length calculations.

## 8.5 DOM interfaces

The following interfaces are defined below: SVGPathSeg, SVGPathSegClosePath, SVGPathSegMovetoAbs, SVGPathSegMovetoRel, SVGPathSegLinetoAbs, SVGPathSegLinetoRel, SVGPathSegCurvetoCubicAbs, SVGPathSegCurvetoCubicRel, SVGPathSegCurvetoQuadraticAbs, SVGPathSegCurvetoQuadraticRel, SVGPathSegArcAbs, SVGPathSegArcRel, SVGPathSegLinetoHorizontalAbs, SVGPathSegLinetoHorizontalRel, SVGPathSegLinetoVerticalAbs, SVGPathSegLinetoVerticalRel, SVGPathSegCurvetoCubicSmoothAbs, SVGPathSegCurvetoCubicSmoothRel, SVGPathSegCurvetoQuadraticSmoothAbs, SVGPathSegCurvetoQuadraticSmoothRel, SVGPathSegList, SVGAnimatedPathData, SVGPathElement.

### Interface SVGPathSeg

The **SVGPathSeg** interface is a base interface that corresponds to a single command within a path data specification.

**IDL Definition**

```
interface SVGPathSeg {

  // Path Segment Types
  const unsigned short PATHSEG_UNKNOWN                    = 0;
  const unsigned short PATHSEG_CLOSEPATH                  = 1;
  const unsigned short PATHSEG_MOVETO_ABS                 = 2;
  const unsigned short PATHSEG_MOVETO_REL                 = 3;
  const unsigned short PATHSEG_LINETO_ABS                 = 4;
  const unsigned short PATHSEG_LINETO_REL                 = 5;
  const unsigned short PATHSEG_CURVETO_CUBIC_ABS          = 6;
  const unsigned short PATHSEG_CURVETO_CUBIC_REL          = 7;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_ABS      = 8;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_REL      = 9;
  const unsigned short PATHSEG_ARC_ABS                    = 10;
  const unsigned short PATHSEG_ARC_REL                    = 11;
  const unsigned short PATHSEG_LINETO_HORIZONTAL_ABS      = 12;
  const unsigned short PATHSEG_LINETO_HORIZONTAL_REL      = 13;
  const unsigned short PATHSEG_LINETO_VERTICAL_ABS        = 14;
  const unsigned short PATHSEG_LINETO_VERTICAL_REL        = 15;
  const unsigned short PATHSEG_CURVETO_CUBIC_SMOOTH_ABS   = 16;
  const unsigned short PATHSEG_CURVETO_CUBIC_SMOOTH_REL   = 17;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS = 18;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL = 19;

  readonly attribute unsigned short pathSegType;
  readonly attribute DOMString      pathSegTypeAsLetter;
};
```

**Definition group Path Segment Types**

**Defined constants**

| | |
|---|---|
| PATHSEG_UNKNOWN | The unit type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| PATHSEG_CLOSEPATH | Corresponds to a "closepath" (z) path data command. |
| PATHSEG_MOVETO_ABS | Corresponds to an "absolute moveto" (M) path data command. |
| PATHSEG_MOVETO_REL | Corresponds to a "relative moveto" (m) path data command. |
| PATHSEG_LINETO_ABS | Corresponds to an "absolute lineto" (L) path data command. |
| PATHSEG_LINETO_REL | Corresponds to a "relative lineto" (l) path data command. |
| PATHSEG_CURVETO_CUBIC_ABS | Corresponds to an "absolute cubic Bézier curveto" (C) path data command. |
| PATHSEG_CURVETO_CUBIC_REL | Corresponds to a "relative cubic Bézier curveto" (c) path data command. |

| | |
|---|---|
| PATHSEG_CURVETO_QUADRATIC_ABS | Corresponds to an "absolute quadratic Bézier curveto" (Q) path data command. |
| PATHSEG_CURVETO_QUADRATIC_REL | Corresponds to a "relative quadratic Bézier curveto" (q) path data command. |
| PATHSEG_ARC_ABS | Corresponds to an "absolute arcto" (A) path data command. |
| PATHSEG_ARC_REL | Corresponds to a "relative arcto" (a) path data command. |
| PATHSEG_LINETO_HORIZONTAL_ABS | Corresponds to an "absolute horizontal lineto" (H) path data command. |
| PATHSEG_LINETO_HORIZONTAL_REL | Corresponds to a "relative horizontal lineto" (h) path data command. |
| PATHSEG_LINETO_VERTICAL_ABS | Corresponds to an "absolute vertical lineto" (V) path data command. |
| PATHSEG_LINETO_VERTICAL_REL | Corresponds to a "relative vertical lineto" (v) path data command. |
| PATHSEG_CURVETO_CUBIC_SMOOTH_ABS | Corresponds to an "absolute smooth cubic curveto" (S) path data command. |
| PATHSEG_CURVETO_CUBIC_SMOOTH_REL | Corresponds to a "relative smooth cubic curveto" (s) path data command. |
| PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS | Corresponds to an "absolute smooth quadratic curveto" (T) path data command. |
| PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL | Corresponds to a "relative smooth quadratic curveto" (t) path data command. |

**Attributes**

**readonly unsigned short pathSegType**

The type of the path segment as specified by one of the constants specified above.

**readonly DOMString pathSegTypeAsLetter**

The type of the path segment, specified by the corresponding one character command name.

## Interface SVGPathSegClosePath

The **SVGPathSegClosePath** interface corresponds to a "closepath" (z) path data command.

**IDL Definition**

```
interface SVGPathSegClosePath : SVGPathSeg {};
```

## Interface SVGPathSegMovetoAbs

The **SVGPathSegMovetoAbs** interface corresponds to an "absolute moveto" (M) path data command.

**IDL Definition**

```
interface SVGPathSegMovetoAbs : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegMovetoRel

The **SVGPathSegMovetoRel** interface corresponds to an "relative moveto" (m) path data command.

**IDL Definition**

```
interface SVGPathSegMovetoRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoAbs

The **SVGPathSegLinetoAbs** interface corresponds to an "absolute lineto" (L) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoAbs : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoRel

The **SVGPathSegLinetoRel** interface corresponds to an "relative lineto" (l) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoCubicAbs

The **SVGPathSegCurvetoCubicAbs** interface corresponds to an "absolute cubic Bézier curveto" (C) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoCubicAbs : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   x1;
                      // raises DOMException on setting
           attribute float   y1;
                      // raises DOMException on setting
           attribute float   x2;
                      // raises DOMException on setting
           attribute float   y2;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x1**

The absolute X coordinate for the first control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y1**

The absolute Y coordinate for the first control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x2**

The absolute X coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y2**

The absolute Y coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoCubicRel

The **SVGPathSegCurvetoCubicRel** interface corresponds to a "relative cubic Bézier curveto" (c) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoCubicRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   x1;
                      // raises DOMException on setting
           attribute float   y1;
                      // raises DOMException on setting
           attribute float   x2;
                      // raises DOMException on setting
           attribute float   y2;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x1**

The relative X coordinate for the first control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y1**

The relative Y coordinate for the first control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x2**

The relative X coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y2**

The relative Y coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoQuadraticAbs

The **SVGPathSegCurvetoQuadraticAbs** interface corresponds to an "absolute quadratic Bézier curveto" (Q) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoQuadraticAbs : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   x1;
                      // raises DOMException on setting
           attribute float   y1;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x1**

The absolute X coordinate for the control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y1**

The absolute Y coordinate for the control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoQuadraticRel

The **SVGPathSegCurvetoQuadraticRel** interface corresponds to a "relative quadratic Bézier curveto" (q) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoQuadraticRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   x1;
                      // raises DOMException on setting
           attribute float   y1;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x1**

The relative X coordinate for the control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y1**

The relative Y coordinate for the control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegArcAbs

The **SVGPathSegArcAbs** interface corresponds to an "absolute arcto" (A) path data command.

**IDL Definition**

```
interface SVGPathSegArcAbs : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   r1;
                      // raises DOMException on setting
           attribute float   r2;
                      // raises DOMException on setting
           attribute float   angle;
                      // raises DOMException on setting
           attribute boolean largeArcFlag;
                      // raises DOMException on setting
           attribute boolean sweepFlag;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float r1**

The x-axis radius for the ellipse (i.e., r1).

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float r2**

The y-axis radius for the ellipse (i.e., r2).

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float angle**

The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**boolean largeArcFlag**

The value of the large-arc-flag parameter.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**boolean sweepFlag**

The value of the sweep-flag parameter.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegArcRel

The **SVGPathSegArcRel** interface corresponds to a "relative arcto" (a) path data command.

**IDL Definition**

```
interface SVGPathSegArcRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
           attribute float   r1;
                      // raises DOMException on setting
           attribute float   r2;
                      // raises DOMException on setting
           attribute float   angle;
                      // raises DOMException on setting
           attribute boolean largeArcFlag;
                      // raises DOMException on setting
           attribute boolean sweepFlag;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float r1**

The x-axis radius for the ellipse (i.e., r1).

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float r2**

The y-axis radius for the ellipse (i.e., r2).

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float angle**

The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**boolean largeArcFlag**

The value of the large-arc-flag parameter.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**boolean sweepFlag**

The value of the sweep-flag parameter.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoHorizontalAbs

The **SVGPathSegLinetoHorizontalAbs** interface corresponds to an "absolute horizontal lineto" (H) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoHorizontalAbs : SVGPathSeg {
          attribute float   x;
                    // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoHorizontalRel

The **SVGPathSegLinetoHorizontalRel** interface corresponds to a "relative horizontal lineto" (h) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoHorizontalRel : SVGPathSeg {
          attribute float   x;
                    // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoVerticalAbs

The **SVGPathSegLinetoVerticalAbs** interface corresponds to an "absolute vertical lineto" (V) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoVerticalAbs : SVGPathSeg {
          attribute float   y;
                    // raises DOMException on setting
};
```

**Attributes**

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegLinetoVerticalRel

The **SVGPathSegLinetoVerticalRel** interface corresponds to a "relative vertical lineto" (v) path data command.

**IDL Definition**

```
interface SVGPathSegLinetoVerticalRel : SVGPathSeg {
           attribute float   y;
                       // raises DOMException on setting
};
```

**Attributes**

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoCubicSmoothAbs

The **SVGPathSegCurvetoCubicSmoothAbs** interface corresponds to an "absolute smooth cubic curveto" (S) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoCubicSmoothAbs : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
           attribute float   x2;
                       // raises DOMException on setting
           attribute float   y2;
                       // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x2**

The absolute X coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y2**

The absolute Y coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoCubicSmoothRel

The **SVGPathSegCurvetoCubicSmoothRel** interface corresponds to a "relative smooth cubic curveto" (s) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoCubicSmoothRel : SVGPathSeg {
           attribute float   x;
                     // raises DOMException on setting
           attribute float   y;
                     // raises DOMException on setting
           attribute float   x2;
                     // raises DOMException on setting
           attribute float   y2;
                     // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x2**

The relative X coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y2**

The relative Y coordinate for the second control point.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoQuadraticSmoothAbs

The **SVGPathSegCurvetoQuadraticSmoothAbs** interface corresponds to an "absolute smooth quadratic curveto" (T) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoQuadraticSmoothAbs : SVGPathSeg {
           attribute float   x;
                     // raises DOMException on setting
           attribute float   y;
                     // raises DOMException on setting
};
```

**Attributes**

**float x**

The absolute X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The absolute Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegCurvetoQuadraticSmoothRel

The **SVGPathSegCurvetoQuadraticSmoothRel** interface corresponds to a "relative smooth quadratic curveto" (t) path data command.

**IDL Definition**

```
interface SVGPathSegCurvetoQuadraticSmoothRel : SVGPathSeg {
           attribute float   x;
                      // raises DOMException on setting
           attribute float   y;
                      // raises DOMException on setting
};
```

**Attributes**

**float x**

The relative X coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

The relative Y coordinate for the end point of this path segment.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGPathSegList

This interface defines a list of **SVGPathSeg** objects.

**SVGPathSegList** has the same attributes and methods as other SVGxxxList interfaces. Implementers may consider using a single base class to implement the various SVGxxxList interfaces.

**IDL Definition**

```
interface SVGPathSegList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                 raises( DOMException );
  SVGPathSeg initialize ( in SVGPathSeg newItem )
                 raises( DOMException, SVGException );
  SVGPathSeg getItem ( in unsigned long index )
                 raises( DOMException );
  SVGPathSeg insertItemBefore ( in SVGPathSeg newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGPathSeg replaceItem ( in SVGPathSeg newItem, in unsigned long index )
                 raises( DOMException, SVGException );
  SVGPathSeg removeItem ( in unsigned long index )
                 raises( DOMException );
  SVGPathSeg appendItem ( in SVGPathSeg newItem )
                 raises( DOMException, SVGException );
};
```

**Attributes**

**readonly unsigned long numberOfItems**

The number of items in the list.

**Methods**

**clear**

Clears all existing current items from the list, with the result being an empty list.

**No Parameters**

**No Return Value**

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

**initialize**

Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter.

**Parameters**

in SVGPathSeg newItem The item which should become the only member of the list.

**Return value**

SVGPathSeg The item being inserted into the list.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**getItem**

Returns the specified item from the list.

**Parameters**

in unsigned long index The index of the item from the list which is to be returned. The first item is number 0.

**Return value**

SVGPathSeg The selected item.

**Exceptions**

DOMException INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

**insertItemBefore**

Inserts a new item into the list at the specified position. The first item is number 0. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGPathSeg newItem The item which is to be inserted into the list.

in unsigned long index The index of the item before which the new item is to be inserted. The first item is number 0. If the index is equal to 0, then the new item is inserted at the front of the list. If the index is greater than or equal to numberOfItems, then the new item is appended to the end of the list.

**Return value**

SVGPathSeg The inserted item.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**replaceItem**

Replaces an existing item in the list with a new item. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGPathSeg newItem The item which is to be inserted into the list.

in unsigned long index The index of the item which is to be replaced. The first item is number 0.

**Return value**

SVGPathSeg The inserted item.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

SVGException SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

**removeItem**

Removes an existing item from the list.

**Parameters**

in unsigned long index The index of the item which is to be removed. The first item is number 0.

**Return value**

SVGPathSeg The removed item.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.
INDEX_SIZE_ERR: Raised if the index number is negative or greater than or equal to numberOfItems.

**appendItem**

Inserts a new item at the end of the list. If newItem is already in a list, it is removed from its previous list before it is inserted into this list.

**Parameters**

in SVGPathSeg newItem The item which is to be inserted into the list. The first item is number 0.

**Return value**

SVGPathSeg The inserted item.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised when the list cannot be modified.

SVGException SVG_WRONG_TYPE_ERR: Raised if parameter newItem is the wrong type of object for the given list.

## Interface SVGAnimatedPathData

The **SVGAnimatedPathData** interface supports elements which have a 'd' attribute which holds SVG path data, and supports the ability to animate that attribute.

The **SVGAnimatedPathData** interface provides two lists to access and modify the base (i.e., static) contents of the **d** attribute:

- DOM attribute pathSegList provides access to the static/base contents of the **d** attribute in a form which matches one-for-one with SVG's

syntax.
- DOM attribute normalizedPathSegList provides normalized access to the static/base contents of the **d** attribute where all path data commands are expressed in terms of the following subset of SVGPathSeg types: SVG_PATHSEG_MOVETO_ABS (M), SVG_PATHSEG_LINETO_ABS (L), SVG_PATHSEG_CURVETO_CUBIC_ABS (C) and SVG_PATHSEG_CLOSEPATH (z).

and two lists to access the current animated values of the **d** attribute:

- DOM attribute animatedPathSegList provides access to the current animated contents of the **d** attribute in a form which matches one-for-one with SVG's syntax.
- DOM attribute animatedNormalizedPathSegList provides normalized access to the current animated contents of the **d** attribute where all path data commands are expressed in terms of the following subset of SVGPathSeg types: SVG_PATHSEG_MOVETO_ABS (M), SVG_PATHSEG_LINETO_ABS (L), SVG_PATHSEG_CURVETO_CUBIC_ABS (C) and SVG_PATHSEG_CLOSEPATH (z).

Each of the two lists are always kept synchronized. Modifications to one list will immediately cause the corresponding list to be modified. Modifications to normalizedPathSegList might cause entries in pathSegList to be broken into a set of normalized path segments.

Additionally, the 'd' attribute on the 'path' element accessed via the XML DOM (e.g., using the getAttribute() method call) will reflect any changes made to pathSegList or normalizedPathSegList.

**IDL Definition**

```
interface SVGAnimatedPathData {

  readonly attribute SVGPathSegList   pathSegList;
  readonly attribute SVGPathSegList   normalizedPathSegList;
  readonly attribute SVGPathSegList   animatedPathSegList;
  readonly attribute SVGPathSegList   animatedNormalizedPathSegList;
};
```

**Attributes**

**readonly SVGPathSegList pathSegList**

Provides access to the base (i.e., static) contents of the **d** attribute in a form which matches one-for-one with SVG's syntax. Thus, if the **d** attribute has an "absolute moveto (M)" and an "absolute arcto (A)" command, then pathSegList will have two entries: a SVG_PATHSEG_MOVETO_ABS and a SVG_PATHSEG_ARC_ABS.

**readonly SVGPathSegList normalizedPathSegList**

Provides access to the base (i.e., static) contents of the **d** attribute in a form where all path data commands are expressed in terms of the following subset of SVGPathSeg types: SVG_PATHSEG_MOVETO_ABS (M), SVG_PATHSEG_LINETO_ABS (L), SVG_PATHSEG_CURVETO_CUBIC_ABS (C) and SVG_PATHSEG_CLOSEPATH (z). Thus, if the **d** attribute has an "absolute moveto (M)" and an "absolute arcto (A)" command, then pathSegList will have one SVG_PATHSEG_MOVETO_ABS entry followed by a series of SVG_PATHSEG_ARC_ABS entries which approximate the arc. This alternate representation is available to provide a simpler interface to developers who would benefit from a more limited set of commands.

The only valid **SVGPathSeg** types are SVG_PATHSEG_MOVETO_ABS (M), SVG_PATHSEG_LINETO_ABS (L), SVG_PATHSEG_CURVETO_CUBIC_ABS (C) and SVG_PATHSEG_CLOSEPATH (z).

**readonly SVGPathSegList animatedPathSegList**

Provides access to the current animated contents of the **d** attribute in a form which matches one-for-one with SVG's syntax. If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'pathSegList'.

**readonly SVGPathSegList animatedNormalizedPathSegList**

Provides access to the current animated contents of the **d** attribute in a form where all path data commands are expressed in terms of the following subset of SVGPathSeg types: SVG_PATHSEG_MOVETO_ABS (M), SVG_PATHSEG_LINETO_ABS (L), SVG_PATHSEG_CURVETO_CUBIC_ABS (C) and SVG_PATHSEG_CLOSEPATH (z). If the given attribute or property is being animated, contains the current animated value of the attribute or property, and both the object itself and its contents are readonly. If the given attribute or property is not currently being animated, contains the same value as 'normalizedPathSegList'.

## Interface SVGPathElement

The **SVGPathElement** interface corresponds to the **'path'** element.

### IDL Definition

```
interface SVGPathElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget,
                SVGAnimatedPathData {

  readonly attribute SVGAnimatedNumber pathLength;

  float         getTotalLength (  );
  SVGPoint      getPointAtLength ( in float distance );
  unsigned long getPathSegAtLength ( in float distance );
  SVGPathSegClosePath    createSVGPathSegClosePath (  );
  SVGPathSegMovetoAbs    createSVGPathSegMovetoAbs ( in float x, in float y );
  SVGPathSegMovetoRel    createSVGPathSegMovetoRel ( in float x, in float y );
  SVGPathSegLinetoAbs    createSVGPathSegLinetoAbs ( in float x, in float y );
  SVGPathSegLinetoRel    createSVGPathSegLinetoRel ( in float x, in float y );
  SVGPathSegCurvetoCubicAbs    createSVGPathSegCurvetoCubicAbs ( in float x, in float y, in float x1, in float y1, in float x2, in float y2 );
  SVGPathSegCurvetoCubicRel    createSVGPathSegCurvetoCubicRel ( in float x, in float y, in float x1, in float y1, in float x2, in float y2 );
  SVGPathSegCurvetoQuadraticAbs    createSVGPathSegCurvetoQuadraticAbs ( in float x, in float y, in float x1, in float y1 );
  SVGPathSegCurvetoQuadraticRel    createSVGPathSegCurvetoQuadraticRel ( in float x, in float y, in float x1, in float y1 );
  SVGPathSegArcAbs    createSVGPathSegArcAbs ( in float x, in float y, in float r1, in float r2, in float angle, in boolean largeArcFlag, in boolean sweepFlag );
  SVGPathSegArcRel    createSVGPathSegArcRel ( in float x, in float y, in float r1, in float r2, in float angle, in boolean largeArcFlag, in boolean sweepFlag );
  SVGPathSegLinetoHorizontalAbs    createSVGPathSegLinetoHorizontalAbs ( in float x );
  SVGPathSegLinetoHorizontalRel    createSVGPathSegLinetoHorizontalRel ( in float x );
  SVGPathSegLinetoVerticalAbs    createSVGPathSegLinetoVerticalAbs ( in float y );
  SVGPathSegLinetoVerticalRel    createSVGPathSegLinetoVerticalRel ( in float y );
  SVGPathSegCurvetoCubicSmoothAbs    createSVGPathSegCurvetoCubicSmoothAbs ( in float x, in float y, in float x2, in float y2 );
  SVGPathSegCurvetoCubicSmoothRel    createSVGPathSegCurvetoCubicSmoothRel ( in float x, in float y, in float x2, in float y2 );
  SVGPathSegCurvetoQuadraticSmoothAbs    createSVGPathSegCurvetoQuadraticSmoothAbs ( in float x, in float y );
  SVGPathSegCurvetoQuadraticSmoothRel    createSVGPathSegCurvetoQuadraticSmoothRel ( in float x, in float y );
};
```

### Attributes

**readonly SVGAnimatedNumber pathLength**

Corresponds to attribute **pathLength** on the given **'path'** element.

### Methods

**getTotalLength**

Returns the user agent's computed value for the total length of the path using the user agent's distance-along-a-path algorithm, as a distance in the current user coordinate system.

**No Parameters**

**Return value**

float The total length of the path.

**No Exceptions**

**getPointAtLength**

Returns the (x,y) coordinate in user space which is `distance` units along the path, utilizing the user agent's distance-along-a-path algorithm.

**Parameters**

in float distance The distance along the path, relative to the start of the path, as a distance in the current user coordinate system.

**Return value**

SVGPoint The returned point in user space.

**No Exceptions**

**getPathSegAtLength**

Returns the index into pathSegList which is `distance` units along the path, utilizing the user agent's distance-along-a-path algorithm.

**Parameters**

in float distance The distance along the path, relative to the start of the path, as a distance in the current user coordinate system.

**Return value**

unsigned long The index of the path segment, where the first path segment is number 0.

**No Exceptions**

**createSVGPathSegClosePath**

Returns a stand-alone, parentless SVGPathSegClosePath object.

**No Parameters**

**Return value**

SVGPathSegClosePath A stand-alone, parentless SVGPathSegClosePath object.

**No Exceptions**

**createSVGPathSegMovetoAbs**

Returns a stand-alone, parentless SVGPathSegMovetoAbs object.

**Parameters**

in float `x` The absolute X coordinate for the end point of this path segment.

in float y The absolute Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegMovetoAbs A stand-alone, parentless SVGPathSegMovetoAbs object.

**No Exceptions**

**createSVGPathSegMovetoRel**

Returns a stand-alone, parentless SVGPathSegMovetoRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

in float y The relative Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegMovetoRel A stand-alone, parentless SVGPathSegMovetoRel object.

**No Exceptions**

**createSVGPathSegLinetoAbs**

Returns a stand-alone, parentless SVGPathSegLinetoAbs object.

**Parameters**

in float x The absolute X coordinate for the end point of this path segment.

in float y The absolute Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoAbs A stand-alone, parentless SVGPathSegLinetoAbs object.

**No Exceptions**

**createSVGPathSegLinetoRel**

Returns a stand-alone, parentless SVGPathSegLinetoRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

in float y The relative Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoRel A stand-alone, parentless SVGPathSegLinetoRel object.

**No Exceptions**

**createSVGPathSegCurvetoCubicAbs**

Returns a stand-alone, parentless SVGPathSegCurvetoCubicAbs object.

**Parameters**

in float x The absolute X coordinate for the end point of this path segment.

in float y The absolute Y coordinate for the end point of this path segment.

in float x1 The absolute X coordinate for the first control point.

in float y1 The absolute Y coordinate for the first control point.

in float x2 The absolute X coordinate for the second control point.

in float y2 The absolute Y coordinate for the second control point.

**Return value**

SVGPathSegCurvetoCubicAbs A stand-alone, parentless SVGPathSegCurvetoCubicAbs object.

**No Exceptions**

**createSVGPathSegCurvetoCubicRel**

Returns a stand-alone, parentless SVGPathSegCurvetoCubicRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

in float y The relative Y coordinate for the end point of this path segment.

in float x1 The relative X coordinate for the first control point.

in float y1 The relative Y coordinate for the first control point.

in float x2 The relative X coordinate for the second control point.

in float y2 The relative Y coordinate for the second control point.

**Return value**

SVGPathSegCurvetoCubicRel A stand-alone, parentless SVGPathSegCurvetoCubicRel object.

**No Exceptions**

**createSVGPathSegCurvetoQuadraticAbs**

Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticAbs object.

**Parameters**

in float x The absolute X coordinate for the end point of this path segment.

in float y The absolute Y coordinate for the end point of this path segment.

in float x1 The absolute X coordinate for the control point.

in float y1 The absolute Y coordinate for the control point.

**Return value**

SVGPathSegCurvetoQuadraticAbs A stand-alone, parentless SVGPathSegCurvetoQuadraticAbs object.

**No Exceptions**

**createSVGPathSegCurvetoQuadraticRel**

Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

in float y The relative Y coordinate for the end point of this path segment.

in float x1 The relative X coordinate for the control point.

in float y1 The relative Y coordinate for the control point.

**Return value**

SVGPathSegCurvetoQuadraticRel A stand-alone, parentless SVGPathSegCurvetoQuadraticRel object.

**No Exceptions**

**createSVGPathSegArcAbs**

Returns a stand-alone, parentless SVGPathSegArcAbs object.

**Parameters**

in float x The absolute X coordinate for the end point of this path segment.

in float y The absolute Y coordinate for the end point of this path segment.

in float r1 The x-axis radius for the ellipse (i.e., r1).

in float r2 The y-axis radius for the ellipse (i.e., r2).

in float angle The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.

in boolean largeArcFlag The value for the large-arc-flag parameter.

in boolean sweepFlag The value for the sweep-flag parameter.

**Return value**

SVGPathSegArcAbs A stand-alone, parentless SVGPathSegArcAbs object.

**No Exceptions**

**createSVGPathSegArcRel**

Returns a stand-alone, parentless SVGPathSegArcRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

in float y The relative Y coordinate for the end point of this path segment.

in float r1 The x-axis radius for the ellipse (i.e., r1).

in float r2 The y-axis radius for the ellipse (i.e., r2).

in float angle The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.

in boolean largeArcFlag The value for the large-arc-flag parameter.

in boolean sweepFlag The value for the sweep-flag parameter.

**Return value**

SVGPathSegArcRel A stand-alone, parentless SVGPathSegArcRel object.

**No Exceptions**

**createSVGPathSegLinetoHorizontalAbs**

Returns a stand-alone, parentless SVGPathSegLinetoHorizontalAbs object.

**Parameters**

in float x The absolute X coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoHorizontalAbs A stand-alone, parentless SVGPathSegLinetoHorizontalAbs object.

**No Exceptions**

**createSVGPathSegLinetoHorizontalRel**

Returns a stand-alone, parentless SVGPathSegLinetoHorizontalRel object.

**Parameters**

in float x The relative X coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoHorizontalRel A stand-alone, parentless SVGPathSegLinetoHorizontalRel object.

**No Exceptions**

**createSVGPathSegLinetoVerticalAbs**

Returns a stand-alone, parentless SVGPathSegLinetoVerticalAbs object.

**Parameters**

in float y The absolute Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoVerticalAbs A stand-alone, parentless SVGPathSegLinetoVerticalAbs object.

**No Exceptions**

**createSVGPathSegLinetoVerticalRel**

Returns a stand-alone, parentless SVGPathSegLinetoVerticalRel object.

**Parameters**

in float y The relative Y coordinate for the end point of this path segment.

**Return value**

SVGPathSegLinetoVerticalRel A stand-alone, parentless SVGPathSegLinetoVerticalRel object.

**No Exceptions**

**createSVGPathSegCurvetoCubicSmoothAbs**

Returns a stand-alone, parentless SVGPathSegCurvetoCubicSmoothAbs object.

**Parameters**

| | |
|---|---|
| in float x | The absolute X coordinate for the end point of this path segment. |
| in float y | The absolute Y coordinate for the end point of this path segment. |
| in float x2 | The absolute X coordinate for the second control point. |
| in float y2 | The absolute Y coordinate for the second control point. |

**Return value**

| | |
|---|---|
| SVGPathSegCurvetoCubicSmoothAbs | A stand-alone, parentless SVGPathSegCurvetoCubicSmoothAbs object. |

**No Exceptions**

**createSVGPathSegCurvetoCubicSmoothRel**

Returns a stand-alone, parentless SVGPathSegCurvetoCubicSmoothRel object.

**Parameters**

| | |
|---|---|
| in float x | The relative X coordinate for the end point of this path segment. |
| in float y | The relative Y coordinate for the end point of this path segment. |
| in float x2 | The relative X coordinate for the second control point. |
| in float y2 | The relative Y coordinate for the second control point. |

**Return value**

| | |
|---|---|
| SVGPathSegCurvetoCubicSmoothRel | A stand-alone, parentless SVGPathSegCurvetoCubicSmoothRel object. |

**No Exceptions**

**createSVGPathSegCurvetoQuadraticSmoothAbs**

Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothAbs object.

**Parameters**

| | |
|---|---|
| in float x | The absolute X coordinate for the end point of this path segment. |
| in float y | The absolute Y coordinate for the end point of this path segment. |

**Return value**

| | |
|---|---|
| SVGPathSegCurvetoQuadraticSmoothAbs | A stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothAbs object. |

**No Exceptions**

**createSVGPathSegCurvetoQuadraticSmoothRel**

Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothRel object.

**Parameters**

| | |
|---|---|
| in float x | The relative X coordinate for the end point of this path segment. |
| in float y | The relative Y coordinate for the end point of this path segment. |

**Return value**

| | |
|---|---|
| SVGPathSegCurvetoQuadraticSmoothRel | A stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothRel object. |

**No Exceptions**

---

04 September 2001

# 9 Basic Shapes

## Contents



## 9.1 Introduction

SVG contains the following set of basic shape elements:

- **rectangles** (rectangle, including optional rounded corners)
- **circles**
- **ellipses**
- **lines**
- **polylines**
- **polygons**

Mathematically, these shape elements are equivalent to a **'path'** element that would construct the same shape. The basic shapes may be stroked, filled and used as clip paths. All of the properties available for **'path'** elements also apply to the basic shapes.

## 9.2 The 'rect' element

The **'rect'** element defines a rectangle which is axis-aligned with the current user coordinate system. Rounded rectangles can be achieved by setting appropriate values for attributes **rx** and **ry**.

```
<!ENTITY % rectExt "" >
<!ELEMENT rect (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%rectExt;)*) >
<!ATTLIST rect
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED
  rx %Length; #IMPLIED
  ry %Length; #IMPLIED >
```

*Attribute definitions:*

**x = "<coordinate>"**
: The x-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value in the current user coordinate system.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>"**
: The y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value in the current user coordinate system.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**width = "<length>"**
: The width of the rectangle.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

**height = "<length>"**
: The height of the rectangle.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

**rx = "<length>"**
: For rounded rectangles, the x-axis radius of the ellipse used to round off the corners of the rectangle.
A negative value is an error (see Error processing).
See the notes below about what happens if the attribute is not specified.
*Animatable: yes.*

**ry = "<length>"**
: For rounded rectangles, the y-axis radius of the ellipse used to round off the corners of the

rectangle.
A negative value is an error (see Error processing).
See the notes below about what happens if the attribute is not specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

If a properly specified value is provided for **rx** but not for **ry**, then the user agent processes the **'rect'** element with the effective value for **ry** as equal to **rx**. If a properly specified value is provided for **ry** but not for **rx**, then the user agent processes the **'rect'** element with the effective value for **rx** as equal to **ry**. If neither **rx** nor **ry** has a properly specified value, then the user agent processes the **'rect'** element as if no rounding had been specified, resulting in square corners. If **rx** is greater than half of the width of the rectangle, then the user agent processes the **'rect'** element with the effective value for **rx** as half of the width of the rectangle. If **ry** is greater than half of the height of the rectangle, then the user agent processes the **'rect'** element with the effective value for **ry** as half of the height of the rectangle.

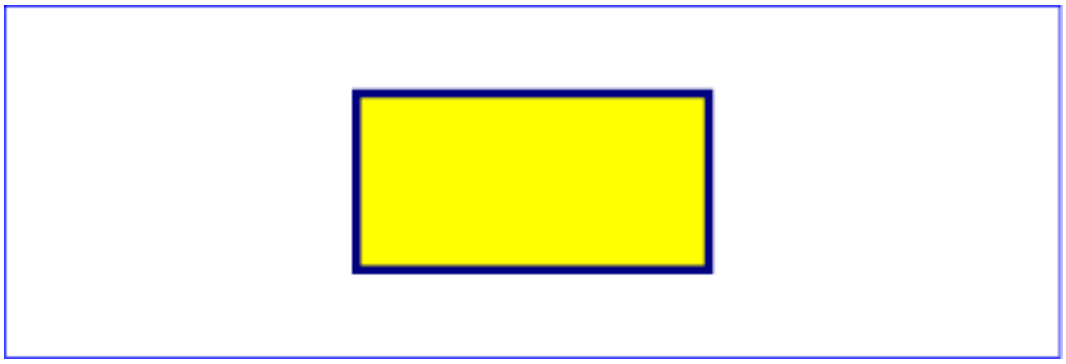
Mathematically, a **'rect'** element can be mapped to an equivalent **'path'** element as follows: (Note: all coordinate and length values are first converted into user space coordinates according to Units.)

- perform an absolute moveto operation to location (*x*+*rx*,*y*), where *x* is the value of the **'rect'** element's **x** attribute converted to user space, *rx* is the effective value of the **rx** attribute converted to user space and *y* is the value of the **y** attribute converted to user space
- perform an absolute horizontal lineto operation to location (*x*+*width*-*rx*,*y*), where *width* is the **'rect'** element's **width** attribute converted to user space
- perform an absolute elliptical arc operation to coordinate (*x*+*width*,*y*+*ry*), where the effective values for the **rx** and **ry** attributes on the **'rect'** element converted to user space are used as the *rx* and *ry* attributes on the elliptical arc command, respectively, the *x-axis-rotation* is set to zero, the *large-arc-flag* is set to zero, and the *sweep-flag* is set to one
- perform a absolute vertical *lineto* to location (*x*+*width*,*y*+*height*-*ry*), where *height* is the **'rect'** element's **height** attribute converted to user space
- perform an absolute elliptical arc operation to coordinate (*x*+*width*-*rx*,*y*+*height*)
- perform an absolute horizontal *lineto* to location (*x*+*rx*,*y*+*height*)
- perform an absolute elliptical arc operation to coordinate (*x*,*y*+*height*-*ry*)
- perform an absolute absolute vertical *lineto* to location (*x*,*y*+*ry*)
- perform an absolute elliptical arc operation to coordinate (*x*+*rx*,*y*)

Example rect01 shows a rectangle with sharp corners. The **'rect'** element is filled with yellow and stroked with navy.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example rect01 - rectangle with sharp corners</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
```

```
        fill="none" stroke="blue" stroke-width="2"/>

  <rect x="400" y="100" width="400" height="200"
        fill="yellow" stroke="navy" stroke-width="10"  />
</svg>
```

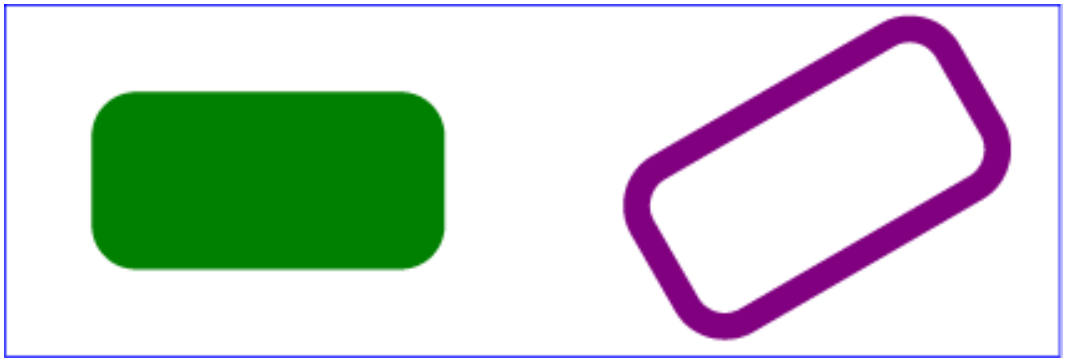


***Example rect01***

*View this example as SVG (SVG-enabled browsers only)*

Example rect02 shows two rounded rectangles. The **rx** specifies how to round the corners of the rectangles. Note that since no value has been specified for the **ry** attribute, it will be assigned the same value as the **rx** attribute.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example rect02 - rounded rectangles</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>

  <rect x="100" y="100" width="400" height="200" rx="50"
        fill="green" />

  <g transform="translate(700 210) rotate(-30)">
    <rect x="0" y="0" width="400" height="200" rx="50"
          fill="none" stroke="purple" stroke-width="30" />
  </g>
</svg>
```



*Example rect02*

*View this example as SVG (SVG-enabled browsers only)*

## 9.3 The 'circle' element

The **'circle'** element defines a circle based on a center point and a radius.

```
<!ENTITY % circleExt "" >
<!ELEMENT circle (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%circleExt;)*) >
<!ATTLIST circle
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  r %Length; #REQUIRED >
```

*Attribute definitions:*

**cx = "<coordinate>"**
: The x-axis coordinate of the center of the circle.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**cy = "<coordinate>"**
: The y-axis coordinate of the center of the circle.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**r = "<length>"**

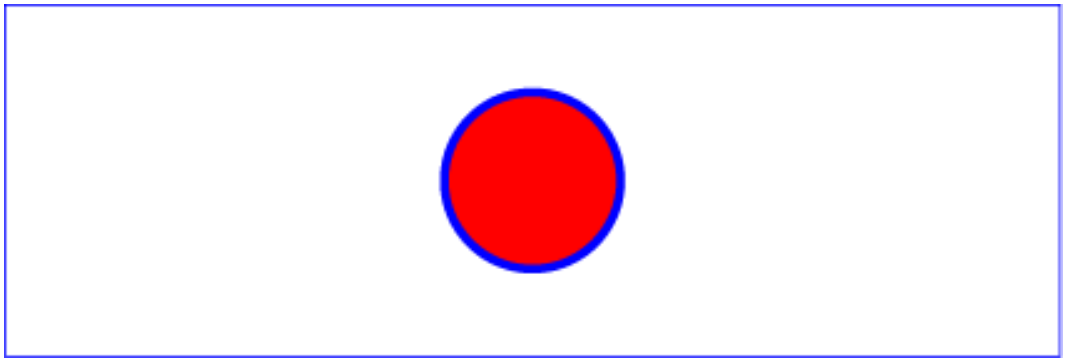
The radius of the circle.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

Example circle01 consists of a **'circle'** element that is filled with red and stroked with blue.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example circle01 - circle filled with red and stroked with blue</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>

  <circle cx="600" cy="200" r="100"
        fill="red" stroke="blue" stroke-width="10"  />
</svg>
```



***Example circle01***

*View this example as SVG (SVG-enabled browsers only)*

## 9.4 The 'ellipse' element

The **'ellipse'** element defines an ellipse which is axis-aligned with the current user coordinate system based on a center point and two radii.

```
<!ENTITY % ellipseExt "" >
<!ELEMENT ellipse (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%ellipseExt;)*) >
<!ATTLIST ellipse
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  rx %Length; #REQUIRED
  ry %Length; #REQUIRED >
```

*Attribute definitions:*

**cx = "<coordinate>"**
: The x-axis coordinate of the center of the ellipse.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**cy = "<coordinate>"**
: The y-axis coordinate of the center of the ellipse.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**rx = "<length>"**
: The x-axis radius of the ellipse.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

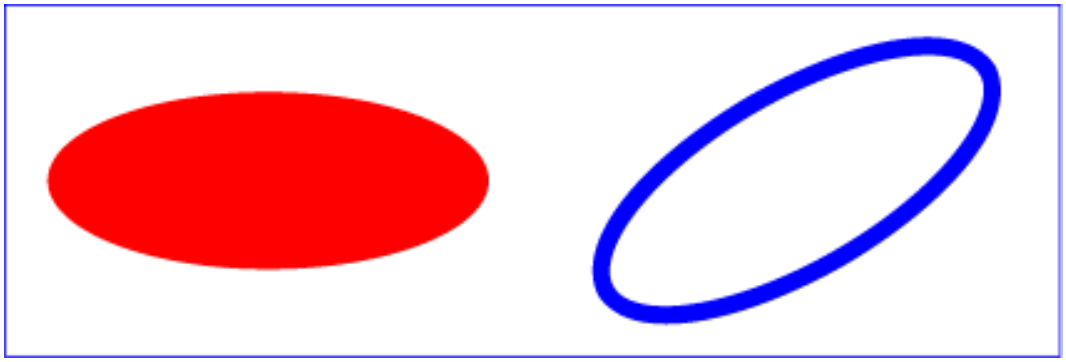
**ry = "<length>"**
: The y-axis radius of the ellipse.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

***Attributes defined elsewhere:***
: **%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

Example ellipse01 below specifies the coordinates of the two ellipses in the user coordinate system established by the **viewBox** attribute on the **'svg'** element and the **transform** attribute on the **'g'** and **'ellipse'** elements. Both ellipses use the default values of zero for the **cx** and **cy** attributes (the center of

the ellipse). The second ellipse is rotated.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example ellipse01 - examples of ellipses</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />

  <g transform="translate(300 200)">
    <ellipse rx="250" ry="100"
          fill="red"  />
  </g>

  <ellipse transform="translate(900 200) rotate(-30)"
        rx="250" ry="100"
        fill="none" stroke="blue" stroke-width="20"  />

</svg>
```



***Example ellipse01***

*View this example as SVG (SVG-enabled browsers only)*

## 9.5 The 'line' element

The **'line'** element defines a line segment that starts at one point and ends at another.

```
<!ENTITY % lineExt "" >
<!ELEMENT line (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
               %geExt;%lineExt;)*) >
<!ATTLIST line
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x1 %Coordinate; #IMPLIED
  y1 %Coordinate; #IMPLIED
  x2 %Coordinate; #IMPLIED
  y2 %Coordinate; #IMPLIED >
```

*Attribute definitions:*

**x1 = "<coordinate>"**
: The x-axis coordinate of the start of the line.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y1 = "<coordinate>"**
: The y-axis coordinate of the start of the line.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**x2 = "<coordinate>"**
: The x-axis coordinate of the end of the line.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y2 = "<coordinate>"**
: The y-axis coordinate of the end of the line.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
: **%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

Mathematically, a **'line'** element can be mapped to an equivalent **'path'** element as follows: (Note: all coordinate and length values are first converted into user space coordinates according to Units.)

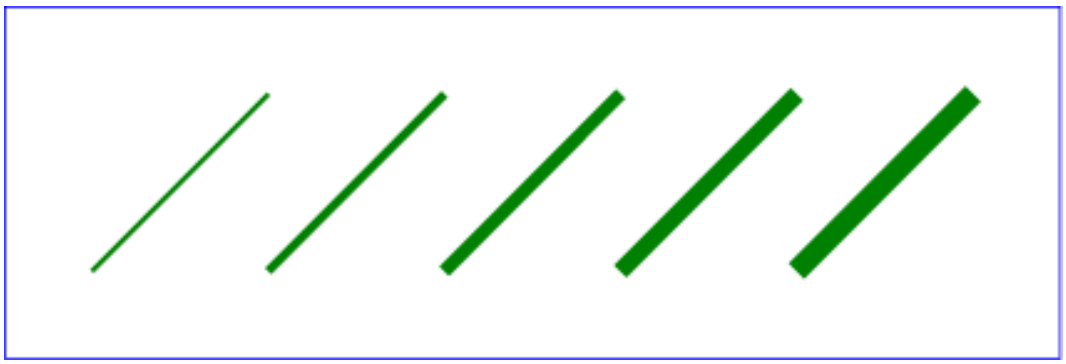
- perform an absolute moveto operation to absolute location (*x1*,*y1*), where *x1* and *y1* are the

values of the **'line'** element's **x1** and **y1** attributes converted to user space, respectively
- perform an absolute lineto operation to absolute location (*x2*,*y2*), where *x2* and *y2* are the values of the **'line'** element's **x2** and **y2** attributes converted to user space, respectively

Because **'line'** elements are single lines and thus are geometrically one-dimensional, they have no interior; thus, **'line'** elements are never filled (see the **'fill'** property).

Example line01 below specifies the coordinates of the five lines in the user coordinate system established by the **viewBox** attribute on the **'svg'** element. The lines have different thicknesses.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example line01 - lines expressed in user coordinates</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />

  <g stroke="green" >
    <line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5"  />
    <line x1="300" y1="300" x2="500" y2="100"
          stroke-width="10"  />
    <line x1="500" y1="300" x2="700" y2="100"
          stroke-width="15"  />
    <line x1="700" y1="300" x2="900" y2="100"
          stroke-width="20"  />
    <line x1="900" y1="300" x2="1100" y2="100"
          stroke-width="25"  />
  </g>
</svg>
```



***Example line01***

*View this example as SVG (SVG-enabled browsers only)*

## 9.6 The 'polyline' element

The **'polyline'** element defines a set of connected straight line segments. Typically, **'polyline'** elements

define open shapes.

```
<!ENTITY % polylineExt "" >
<!ELEMENT polyline (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
              %geExt;%polylineExt;)*) >
<!ATTLIST polyline
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  points %Points; #REQUIRED >
```

*Attribute definitions:*

**`points` = "<list-of-points>"**
: The points that make up the polyline. All coordinate values are in the user coordinate system.
*Animatable: yes.*

***Attributes defined elsewhere:***
: **%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

If an odd number of coordinates is provided, then the element is in error, with the same user agent behavior as occurs with an incorrectly specified **'path'** element.

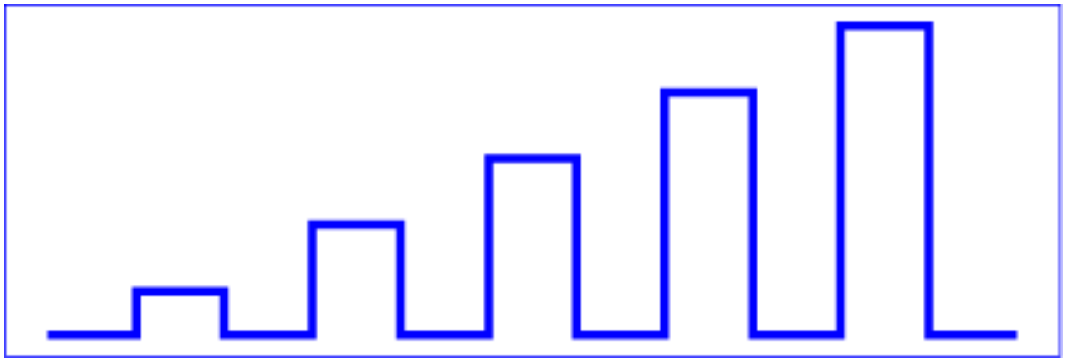
Mathematically, a **'polyline'** element can be mapped to an equivalent **'path'** element as follows:

- perform an absolute moveto operation to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute lineto operation to that coordinate pair.

Example polyline01 below specifies a polyline in the user coordinate system established by the **viewBox** attribute on the **'svg'** element.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example polyline01 - increasingly larger bars</desc>
```

```
    <!-- Show outline of canvas using 'rect' element -->
    <rect x="1" y="1" width="1198" height="398"
          fill="none" stroke="blue" stroke-width="2" />

    <polyline fill="none" stroke="blue" stroke-width="10"
              points="50,375
                      150,375 150,325 250,325 250,375
                      350,375 350,250 450,250 450,375
                      550,375 550,175 650,175 650,375
                      750,375 750,100 850,100 850,375
                      950,375 950,25 1050,25 1050,375
                      1150,375" />
</svg>
```



***Example polyline01***

*View this example as SVG (SVG-enabled browsers only)*

## 9.7 The 'polygon' element

The **'polygon'** element defines a closed shape consisting of a set of connected straight line segments.

```
<!ENTITY % polygonExt "" >
<!ELEMENT polygon (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
              %geExt;%polygonExt;)*) >
<!ATTLIST polygon
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  points %Points; #REQUIRED >
```

*Attribute definitions:*

**`points` = "<list-of-points>"**

The points that make up the polygon. All coordinate values are in the user coordinate system.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-Graphics;.**

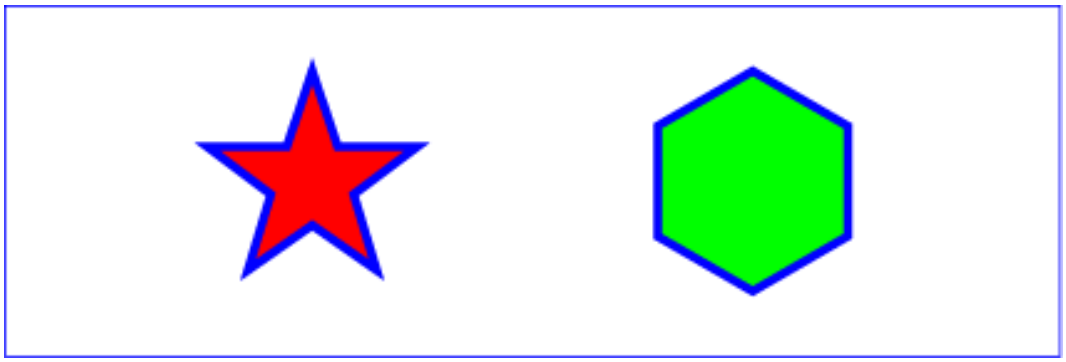
If an odd number of coordinates is provided, then the element is in error, with the same user agent behavior as occurs with an incorrectly specified **'path'** element.

Mathematically, a **'polygon'** element can be mapped to an equivalent **'path'** element as follows:

- perform an absolute moveto operation to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute lineto operation to that coordinate pair
- perform a closepath command

Example polygon01 below specifies two polygons (a star and a hexagon) in the user coordinate system established by the **viewBox** attribute on the **'svg'** element.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example polygon01 - star and hexagon</desc>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />

  <polygon fill="red" stroke="blue" stroke-width="10"
           points="350,75  379,161 469,161 397,215
                   423,301 350,250 277,301 303,215
                   231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10"
           points="850,75  958,137.5 958,262.5
                   850,325 742,262.6 742,137.5" />
</svg>
```



*Example polygon01*

*View this example as SVG (SVG-enabled browsers only)*

## 9.8 The grammar for points specifications in 'polyline' and 'polygon' elements

The following is the Backus-Naur Form (BNF) for points specifications in **'polyline'** and **'polygon'** elements. The following notation is used:

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- (): grouping
- |: separates alternatives
- double quotes surround literals

```
list-of-points:
    wsp* coordinate-pairs? wsp*

coordinate-pairs:
    coordinate-pair
    | coordinate-pair comma-wsp coordinate-pairs

coordinate-pair:
    coordinate comma-wsp coordinate

coordinate:
    number

number:
    sign? integer-constant
    | sign? floating-point-constant

comma-wsp:
    (wsp+ comma? wsp*) | (comma wsp*)

comma:
    ","

integer-constant:
    digit-sequence

floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent
```

```
fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."

exponent:
    ( "e" | "E" ) sign? digit-sequence

sign:
    "+" | "-"

digit-sequence:
    digit
    | digit digit-sequence

digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

wsp:
    (#x20 | #x9 | #xD | #xA)+
```

## 9.9 DOM interfaces

The following interfaces are defined below: SVGRectElement, SVGCircleElement, SVGEllipseElement, SVGLineElement, SVGAnimatedPoints, SVGPolylineElement, SVGPolygonElement.

### Interface SVGRectElement

The **SVGRectElement** interface corresponds to the **'rect'** element.

**IDL Definition**

```
interface SVGRectElement :
               SVGElement,
               SVGTests,
               SVGLangSpace,
               SVGExternalResourcesRequired,
               SVGStylable,
               SVGTransformable,
               events::EventTarget {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};
```

**Attributes**

**readonly SVGAnimatedLength x**

Corresponds to attribute **x** on the given **'rect'** element.

**readonly SVGAnimatedLength y**

Corresponds to attribute **y** on the given **'rect'** element.

**readonly SVGAnimatedLength width**

Corresponds to attribute **width** on the given **'rect'** element.

**readonly SVGAnimatedLength height**

Corresponds to attribute **height** on the given **'rect'** element.

**readonly SVGAnimatedLength rx**

Corresponds to attribute **rx** on the given **'rect'** element.

**readonly SVGAnimatedLength ry**

Corresponds to attribute **ry** on the given **'rect'** element.

## Interface SVGCircleElement

The **SVGCircleElement** interface corresponds to the **'rect'** element.

**IDL Definition**

```
interface SVGCircleElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
};
```

**Attributes**

**readonly SVGAnimatedLength cx**

Corresponds to attribute **cx** on the given **'circle'** element.

**readonly SVGAnimatedLength cy**

Corresponds to attribute **cy** on the given **'circle'** element.

**readonly SVGAnimatedLength r**

Corresponds to attribute **r** on the given **'circle'** element.

## Interface SVGEllipseElement

The **SVGEllipseElement** interface corresponds to the **'ellipse'** element.

**IDL Definition**

```
interface SVGEllipseElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
```

```
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};
```

**Attributes**

**readonly SVGAnimatedLength cx**
Corresponds to attribute **cx** on the given **'ellipse'** element.

**readonly SVGAnimatedLength cy**
Corresponds to attribute **cy** on the given **'ellipse'** element.

**readonly SVGAnimatedLength rx**
Corresponds to attribute **rx** on the given **'ellipse'** element.

**readonly SVGAnimatedLength ry**
Corresponds to attribute **ry** on the given **'ellipse'** element.

## Interface SVGLineElement

The **SVGLineElement** interface corresponds to the **'line'** element.

**IDL Definition**

```
interface SVGLineElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};
```

**Attributes**

**readonly SVGAnimatedLength x1**
Corresponds to attribute **x1** on the given **'line'** element.

**readonly SVGAnimatedLength y1**
Corresponds to attribute **y1** on the given **'line'** element.

**readonly SVGAnimatedLength x2**
Corresponds to attribute **x2** on the given **'line'** element.

**readonly SVGAnimatedLength y2**
Corresponds to attribute **y2** on the given **'line'** element.

## Interface SVGAnimatedPoints

The **SVGAnimatedPoints** interface supports elements which have a 'points' attribute which holds a list of coordinate values and which support the ability to animate that attribute.

Additionally, the 'points' attribute on the original element accessed via the XML DOM (e.g., using the getAttribute() method call) will reflect any changes made to points.

### IDL Definition

```
interface SVGAnimatedPoints {

  readonly attribute SVGPointList   points;
  readonly attribute SVGPointList   animatedPoints;
};
```

### Attributes

**readonly SVGPointList points**

Provides access to the base (i.e., static) contents of the **points** attribute.

**readonly SVGPointList animatedPoints**

Provides access to the current animated contents of the **points** attribute. If the given attribute or property is being animated, contains the current animated value of the attribute or property. If the given attribute or property is not currently being animated, contains the same value as 'points'.

## Interface SVGPolylineElement

The **SVGPolylineElement** interface corresponds to the **'polyline'** element.

### IDL Definition

```
interface SVGPolylineElement :
               SVGElement,
               SVGTests,
               SVGLangSpace,
               SVGExternalResourcesRequired,
               SVGStylable,
               SVGTransformable,
               events::EventTarget,
               SVGAnimatedPoints {};
```

## Interface SVGPolygonElement

The **SVGPolygonElement** interface corresponds to the **'polygon'** element.

### IDL Definition

```
interface SVGPolygonElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget,
                SVGAnimatedPoints {};
```

---

04 September 2001

# 10 Text

## Contents



## 10.1 Introduction

Text that is to be rendered as part of an SVG document fragment is specified using the **'text'** element. The characters to be drawn are expressed as XML character data [XML10] inside the **'text'** element.

SVG's **'text'** elements are rendered like other graphics elements. Thus, coordinate system transformations, painting, clipping and masking features apply to **'text'** elements in the same way as they apply to shapes such as paths and rectangles.

Each **'text'** element causes a single string of text to be rendered. SVG performs no automatic line breaking or word wrapping. To achieve the effect of multiple lines of text, use one of the following methods:

- The author or authoring package needs to pre-compute the line breaks and use multiple **'text'** elements (one for each line of text).
- The author or authoring package needs to pre-compute the line breaks and use a single **'text'** element with one or more **'tspan'** child elements with appropriate values for attributes **x**, **y**, **dx** and **dy** to set new start positions for those characters which start new lines. (This approach allows user text selection across multiple lines of text -- see Text selection and clipboard operations.)
- Express the text to be rendered in another XML namespace such as XHTML [XHTML] embedded inline within a **'foreignObject'** element. (Note: the exact semantics of this approach are not completely defined at this time.)

The text strings within **'text'** elements can be rendered in a straight line or rendered along the outline of a **'path'** element. SVG supports the following international text processing features for both straight line text and text on a path:

- horizontal and vertical orientation of text
- left-to-right or bidirectional text (i.e., languages which intermix right-to-left and left-to-right text, such as Arabic and Hebrew)
- when SVG fonts are used, automatic selection of the correct glyph corresponding to the current form for Arabic and Han text

(The layout rules for straight line text are described in Text layout. The layout rules for text on a path are described in Text on a path layout rules.)

Because SVG text is packaged as XML character data [XML10]:

- Text data in SVG content is readily accessible to the visually impaired (see Accessibility Support)
- In many viewing scenarios, the user will be able to search for and select text strings and copy selected text strings to the system clipboard (see Text selection and clipboard operations)
- XML-compatible Web search engines will find text strings in SVG content with no additional effort over what they need to do to find text strings in other XML documents

Multi-language SVG content is possible by substituting different text strings based on the user's preferred language.

For accessibility reasons, it is recommended that text which is included in a document have appropriate semantic markup to indicate its function. See SVG accessibility guidelines for more information.

## 10.2 Characters and their corresponding glyphs

In XML [XML10], textual content is defined in terms of a sequence of XML **characters**, where each character is defined by a particular Unicode code point [UNICODE]. Fonts, on the other hand, consists of a

collection of **glyphs** and other associated information, such as font tables. A glyph is a presentable form of one or more characters (or a part of a character in some cases). Each glyph consists of some sort of identifier (in some cases a string, in other cases a number) along with drawing instructions for rendering that particular glyph.

In many cases, there is a one-to-one mapping of Unicode characters (i.e., Unicode code points) to glyphs in a font. For example, it is common for a font designed for Latin languages (where the term *Latin* is used for European languages such as English with alphabets similar to and/or derivative to the Latin language) to contain a single glyph for each of the standard ASCII characters (i.e., A-to-Z, a-to-z, 0-to-9, plus the various punctuation characters found in ASCII). Thus, in most situations, the string "XML", which consists of three Unicode characters, would be rendered by the three glyphs corresponding to "X", "M" and "L", respectively.

In various other cases, however, there is not a strict one-to-one mapping of Unicode characters to glyphs. Some of the circumstances when the mapping is not one-to-one:

- Ligatures - For best looking typesetting, it is often desirable that particular sequences of characters are rendered as a single glyph. An example is the word "office". Many fonts will define an "ffi" ligature. When the word "office" is rendered, sometimes the user agent will render the glyph for the "ffi" ligature instead of rendering distinct glyphs (i.e., "f", "f" and "i") for each of the three characters. Thus, for ligatures, multiple Unicode characters map to a single glyph. (Note that for proper rendering of some languages, ligatures are required for certain character combinations.)
- Composite characters - In various situations, commonly used adornments such as diacritical marks will be stored once in a font as a particular glyph and then composed with one or more other glyphs to result in the desired character. For example, it is possible that a font engine might render the **é** character by first rendering the glyph for **e** and then rendering the glyph for ´ (the accent mark) such that the accent mark will appear over the **e**. In this situation, a single Unicode character maps to multiple glyphs.
- Glyph substitution - Some typography systems examine the nature of the textual content and utilize different glyphs in different circumstances. For example, in Arabic, the same Unicode character might render as any of four different glyphs, depending on such factors as whether the character appears at the start, the end or the middle of a sequence of cursively joined characters. Different glyphs might be used for a punctuation character depending on inline-progression-direction (e.g., horizontal vs. vertical). In these situations, a single Unicode character might map to one of several alternative glyphs.
- In some languages, particular sequences of characters will be converted into multiple glyphs such that parts of a particular character are in one glyph and the remainder of that character is in another glyph.
- Alternative glyph specification - SVG contains a facility for the author to explicitly specify that a particular sequence of Unicode characters is to be rendered using a particular glyph. (See Alternate glyphs.) When this facility is used, multiple Unicode characters map to a single glyph.

In many situations, the algorithms for mapping from characters to glyphs are system-dependent, resulting in the possibility that the rendering of text might be (usually slightly) different when viewed in different user environments. If the author of SVG content requires precise selection of fonts and glyphs, then the recommendation is that the necessary fonts (potentially subsetted to include only the glyphs needed for the given document) be available either as SVG fonts embedded within the SVG content or as WebFonts posted at the same Web location as the SVG content.

Throughout this chapter, the term **character** shall be equivalent to the definition of a character in XML [XML10].

## 10.3 Fonts, font tables and baselines

A font consists of a collection of glyphs together with the information (the font tables) necessary to use those glyphs to present characters on some medium. The combination of the collection of glyphs and the font tables is called the *font data*. The font tables include the information necessary to map characters to glyphs, to determine the size of glyph areas and to position the glyph area. Each font table consists of one or more font characteristics, such as the font-weight and font-style.

The geometric font characteristics are expressed in a coordinate system based on the EM box. (The EM is a relative measure of the height of the glyphs in the font; see CSS2 em square.) The box 1 EM high and 1 EM wide is called the *design space*. This space is given a geometric coordinates by sub-dividing the EM into a number of units-per-em.

Note: Units-per-em is a font characteristic. A typical value for units-per-EM is 1000 or 2048.

The coordinate space of the EM box is called the *design space coordinate system*. For scalable fonts, the curves and lines that are used to draw a glyph are represented using this coordinate system.

Note: Most often, the (0,0) point in this coordinate system is positioned on the left edge of the EM box, but not at the bottom left corner. The Y coordinate of the bottom of a roman capital letter is usually zero. And the descenders on lowercase roman letters have negative coordinate values.

SVG assumes that the font tables will provide at least three font characteristics: an ascent, a descent and a set of baseline-tables. The ascent is the distance to the top of the EM box from the (0,0) point of the font; the descent is the distance to the bottom of the EM box from the (0.0) point of the font. The baseline-table is explained below.

Note: Within an OpenType font, for horizontal writing-modes, the ascent and descent are given by the sTypoAscender and sTypoDescender entries in the OS/2 table. For vertical writing-modes, the descent (the distance, in this case from the (0,0) point to the left edge of the glyph) is normally zero because the (0,0) point is on the left edge. The ascent for vertical writing-modes is either 1 em or is specified by the ideographic top baseline value in the OpenType Base table for vertical writing-modes.

In horizontal writing-modes, the glyphs of a given script are positioned so that a particular point on each glyph, the *alignment-point*, is aligned with the alignment-points of the other glyphs in that script. The glyphs of different scripts, for example, Western, Northern Indic and Far-Eastern scripts, are typically aligned at different points on the glyph. For example, Western glyphs are aligned on the bottoms of the capital letters, northern indic glyphs are aligned at the top of a horizontal stroke near the top of the glyphs and far-eastern glyphs are aligned either at the bottom or center of the glyph. Within a script and within a line of text having a single font-size, the sequence of alignment-points defines, in the inline- progression-direction, a geometric line called a *baseline*. Western and most other alphabetic and syllabic glyphs are aligned to an "alphabetic" baseline, the northern indic glyphs are aligned to a "hanging" baseline and the far-eastern glyphs are aligned to an "ideographic" baseline.

A *baseline-table* specifies the position of one or more baselines in the design space coordinate system. The function of the baseline table is to facilitate the alignment of different scripts with respect to each other when they are mixed on the same text line. Because the desired relative alignments may depend on which script is dominant in a line (or block), there may be a different baseline table for each script. In addition, different alignment positions are needed for horizontal and vertical writing modes. Therefore, the font may have a set of baseline tables: typically, one or more for horizontal writing-modes and zero or more for vertical writing-modes.

Note: Some fonts may not have values for the baseline tables. Heuristics are suggested for approximating the baseline tables when a given font does not supply baseline tables.

SVG further assumes that for each glyph in the font data for a font, there are two width values, two alignment-baselines and two alignment-points, one each for horizontal writing-modes and the other for vertical writing-modes. (Even though it is specified as a width, for vertical writing-modes the width is used in the vertical direction.) The script to which a glyph belongs determines an alignment-baseline to which the glyph is to be aligned. The inline-progression-direction position of the alignment-point is on the start-edge of the glyph.

Properties related to baselines are described below under Baseline alignment properties.

In addition to the font characteristics required above, a font may also supply substitution and positioning tables that can be used by a formatter to re-order, combine and position a sequence of glyphs to make one or more composite glyphs. The combination may be as simple as a ligature, or as complex as an indic syllable which combines, usually with some re-ordering, multiple consonants and vowel glyphs.

## 10.4 The 'text' element

The **'text'** element defines a graphics element consisting of text. The XML [XML10] character data within the **'text'** element, along with relevant attributes and properties and character-to-glyph mapping tables within the font itself, define the glyphs to be rendered. (See Characters and their corresponding glyphs.) The attributes and properties on the **'text'** element indicate such things as the writing direction, font specification and painting attributes which describe how exactly to render the characters. Subsequent sections of this chapter describe the relevant text-specific attributes and properties, particular text layout and bidirectionality.

Since **'text'** elements are rendered using the same rendering methods as other graphics elements, all of the same coordinate system transformations, painting, clipping and masking features that apply to shapes such as paths and rectangles also apply to **'text'** elements.

It is possible to apply a gradient, pattern, clipping path, mask or filter to text. When one of these facilities is applied to text and keyword **objectBoundingBox** is used (see Object bounding box units) to specify a graphical effect relative to the "object bounding box", then the object bounding box units are computed relative to the entire **'text'** element in all cases, even when different effects are applied to different **'tspan'** elements within the same **'text'** element.

The **'text'** element renders its first glyph (after bidirectionality reordering) at the initial current text position, which is established by the **x** and **y** attributes on the **'text'** element (with possible adjustments due to the value of the **'text-anchor'** property, the presence of a **'textPath'** element containing the first character, and/or an **x**, **y**, **dx** or **dy** attributes on a **'tspan'**, **'tref'** or **'altGlyph'** element which contains the first character). After the glyph(s) corresponding to the given character is(are) rendered, the current text position is updated for the next character. In the simplest case, the new current text position is the previous current text position plus the glyphs' advance value (horizontal or vertical). See text layout for a description of glyph placement and glyph advance.

```
<!ENTITY % textExt "" >
<!ELEMENT text (#PCDATA|desc|title|metadata|
                tspan|tref|textPath|altGlyph|a|animate|set|
                animateMotion|animateColor|animateTransform
                %geExt;%textExt;)* >
<!ATTLIST text
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %PresentationAttributes-TextElements;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >
```

*Attribute definitions:*

**x = "<coordinate>+"**

If a single <coordinate> is provided, then the value represents the new absolute X coordinate for the current text position for rendering the glyphs that correspond to the first character within this element or any of its descendants.
If a comma- or space-separated list of <n> <coordinate>s is provided, then the values represent new absolute X coordinates for the current text position for rendering the glyphs corresponding to each of the first <n> characters within this element or any of its descendants.
For additional processing rules, refer to the description of the **x** attribute on the **'tspan'** element.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>+"**

The corresponding list of absolute Y coordinates for the glyphs corresponding to the characters within this element. The processing rules for the **'y'** attribute parallel the processing rules for the **'x'** attribute.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**dx = "<length>+"**

Shifts in the current text position along the x-axis for the characters within this element or any of its descendants.
Refer to the description of the **dx** attribute on the **'tspan'** element.
If the attribute is not specified on this element or any of its descendants, no supplemental shifts along the x-axis will occur.
*Animatable: yes.*

**dy = "<length>+"**

Shifts in the current text position along the y-axis for the characters within this element or any of its descendants.
Refer to the description of the **dy** attribute on the **'tspan'** element.
If the attribute is not specified on this element or any of its descendants, no supplemental shifts along the y-axis will occur.
*Animatable: yes.*

**`rotate` = "<number>+"**

The supplemental rotation about the current text position that will be applied to all of the glyphs corresponding to each character within this element.
Refer to the description of the **rotate** attribute on the **'tspan'** element.
If the attribute is not specified on this element or any of its descendants, no supplemental rotations will occur.
*Animatable: yes (non-additive, 'set' and 'animate' elements only).*

**`textLength` = "<length>"**

The author's computation of the total sum of all of the advance values that correspond to character data within this element, including the advance value on the glyph (horizontal or vertical), the effect of properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **dx** and **dy** on **'tspan'** elements. This value is used to calibrate the user agent's own calculations with that of the author.
The purpose of this attribute is to allow the author to achieve exact alignment, in visual rendering order after any bidirectional reordering, for the first and last rendered glyphs that correspond to this element; thus, for the last rendered character (in visual rendering order after any bidirectional reordering), any supplemental inter-character spacing beyond normal glyph advances are ignored (in most cases) when the user agent determines the appropriate amount to expand/compress the text string to fit within a length of **textLength**.
A negative value is an error (see Error processing).
If the attribute is not specified, the effect is as if the author's computation exactly matched the value calculated by the user agent; thus, no advance adjustments are made.
*Animatable: yes.*

**`lengthAdjust` = "spacing|spacingAndGlyphs"**

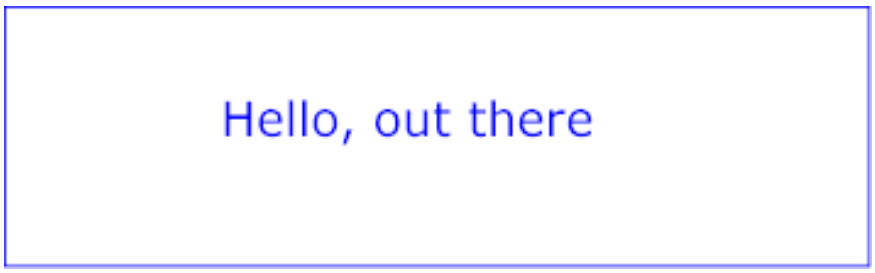
Indicates the type of adjustments which the user agent shall make to make the rendered length of the text match the value specified on the **textLength** attribute.
**spacing** indicates that only the advance values are adjusted. The glyphs themselves are not stretched or compressed.
**spacingAndGlyphs** indicates that the advance values are adjusted and the glyphs themselves stretched or compressed in one axis (i.e., a direction parallel to the inline-progression-direction).
The user agent is required to achieve correct start and end positions for the text strings, but the locations of intermediate glyphs are not predictable because user agents might employ advanced algorithms to stretch or compress text strings in order to balance correct start and end positioning with optimal typography.
Note that, for a text string that contains <n> characters, the adjustments to the advance values often occur only for <n-1> characters (see description of attribute **textLength**), whereas stretching or compressing of the glyphs will be applied to all <n> characters.
If the attribute is not specified, the effect is as a value of **spacing** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %testAttrs;, %langSpaceAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements;, %PresentationAttributes-TextElements;, transform, %graphicsElementEvents;.**

Example text01 below contains the text string "Hello, out there" which will be rendered onto the canvas using the Verdana font family with the glyphs filled with the color blue.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example text01 - 'Hello, out there' in blue</desc>

  <text x="250" y="150"
        font-family="Verdana" font-size="55" fill="blue" >
    Hello, out there
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```




***Example text01***

*View this example as SVG (SVG-enabled browsers only)*

## 10.5 The 'tspan' element

Within a **'text'** element, text and font properties and the current text position can be adjusted with absolute or relative coordinate values by including a **'tspan'** element.

```
<!ENTITY % tspanExt "" >
<!ELEMENT tspan (#PCDATA|desc|title|metadata|tspan|tref|altGlyph|a|animate|set|animateColor
                %tspanExt;)* >
<!ATTLIST tspan
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
```

```
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >
```

*Attribute definitions:*

**x = "<coordinate>+"**

If a single <coordinate> is provided, then the value represents the new absolute X coordinate for the current text position for rendering the glyphs that correspond to the first character within this element or any of its descendants.
If a comma- or space-separated list of <n> <coordinate>s is provided, then the values represent new absolute X coordinates for the current text position for rendering the glyphs corresponding to each of the first <n> characters within this element or any of its descendants.
If more <coordinate>s are provided than characters, then the extra <coordinate>s will have no effect on glyph positioning.
If more characters exist than <coordinate>s, then for each of these extra characters: (a) if an ancestor **'text'** or **'tspan'** element specifies an absolute X coordinate for the given character via an **x** attribute, then that absolute X coordinate is used as the starting X coordinate for that character (nearest ancestor has precedence), else (b) the starting X coordinate for rendering the glyphs corresponding to the given character is the X coordinate of the resulting current text position from the most recently rendered glyph for the current **'text'** element.
If the attribute is not specified: (a) if an ancestor **'text'** or **'tspan'** element specifies an absolute X coordinate for a given character via an **x** attribute, then that absolute X coordinate is used (nearest ancestor has precedence), else (b) the starting X coordinate for rendering the glyphs corresponding to a given character is the X coordinate of the resulting current text position from the most recently rendered glyph for the current **'text'** element.
*Animatable: yes.*

**y = "<coordinate>+"**

The corresponding list of absolute Y coordinates for the glyphs corresponding to the characters within this element. The processing rules for the **'y'** attribute parallel the processing rules for the **'x'** attribute.
*Animatable: yes.*

**dx = "<length>+"**

If a single <length> is provided, this value represents the new relative X coordinate for the current text position for rendering the glyphs corresponding to the first character within this element or any of its descendants. The current text position is shifted along the x-axis of the current user coordinate system by <length> before the first character's glyphs are rendered.
If a comma- or space-separated list of <n> <length>s is provided, then the values represent incremental shifts along the x-axis for the current text position before rendering the glyphs corresponding to the first <n> characters within this element or any of its descendants. Thus, before the glyphs are rendered corresponding to each character, the current text position resulting from drawing the glyphs for the previous character within the current **'text'** element is shifted along the X axis of the current user coordinate system by <length>.
If more <length>s are provided than characters, then any extra <length>s will have no effect on glyph positioning.
If more characters exist than <length>s, then for each of these extra characters: (a) if an ancestor **'text'** or **'tspan'** element specifies a relative X coordinate for the given character via a **dx** attribute,

then the current text position is shifted along the x-axis of the current user coordinate system by that amount (nearest ancestor has precedence), else (b) no extra shift along the x-axis occurs.
If the attribute is not specified: (a) if an ancestor **'text'** or **'tspan'** element specifies a relative X coordinate for a given character via a **dx** attribute, then the current text position is shifted along the x-axis of the current user coordinate system by that amount (nearest ancestor has precedence), else (b) no extra shift along the x-axis occurs.
*Animatable: yes.*

**dy = "<length>+"**

The corresponding list of relative Y coordinates for the characters within the **'tspan'** element. The processing rules for the **'dy'** attribute parallel the processing rules for the **'dx'** attribute.
*Animatable: yes.*

**rotate = "<number>+"**

The supplemental rotation about the current text position that will be applied to all of the glyphs corresponding to each character within this element.
If a comma- or space-separated list of <number>s is provided, then the first <number> represents the supplemental rotation for the glyphs corresponding to the first character within this element or any of its descendants, the second <number> represents the supplemental rotation for the glyphs that correspond to the second character, and so on.
If more <number>s are provided than there are characters, then the extra <number>s will be ignored.
If more characters are provided than <number>s, then for each of these extra characters: (a) if an ancestor **'text'** or **'tspan'** element specifies a supplemental rotation for the given character via a **rotate** attribute, then the given supplemental rotation is applied to the given character, else (b) no supplemental rotation occurs.
If the attribute is not specified: (a) if an ancestor **'text'** or **'tspan'** element specifies a supplemental rotation for a given character via a **rotate** attribute, then the given supplemental rotation is applied to the given character (nearest ancestor has precedence), else (b) no supplemental rotation occurs.
This supplemental rotation has no impact on the rules by which current text position is modified as glyphs get rendered and is supplemental to any rotation due to text on a path and to **'glyph-orientation-horizontal'** or **'glyph-orientation-vertical'**.
*Animatable: yes (non-additive, 'set' and 'animate' elements only).*

**textLength = "<length>"**

The author's computation of the total sum of all of the advance values that correspond to character data within this element, including the advance value on the glyph (horizontal or vertical), the effect of properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **dx** and **dy** on this **'tspan'** element or any descendants. This value is used to calibrate the user agent's own calculations with that of the author.
The purpose of this attribute is to allow the author to achieve exact alignment, in visual rendering order after any bidirectional reordering, for the first and last rendered glyphs that correspond to this element; thus, for the last rendered character (in visual rendering order after any bidirectional reordering), any supplemental inter-character spacing beyond normal glyph advances are ignored (in most cases) when the user agent determines the appropriate amount to expand/compress the text string to fit within a length of **textLength**.
If attribute **textLength** is specified on a given element and also specified on an ancestor, the adjustments on all character data within this element are controlled by the value of **textLength** on this element exclusively, with the possible side-effect that the adjustment ratio for the contents of this element might be different than the adjustment ratio used for other content that shares the same ancestor. The user agent must assume that the total advance values for the other content within that ancestor is the difference between the advance value on that ancestor and the advance value for this element.
A negative value is an error (see Error processing).
If the attribute is not specified anywhere within a **'text'** element, the effect is as if the author's

computation exactly matched the value calculated by the user agent; thus, no advance adjustments are made.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements;, lengthAdjust.**

The **x**, **y**, **dx**, **dy** and **rotate** on the **'tspan'** element are useful in high-end typography scenarios where individual glyphs require exact placement. These attributes are useful for minor positioning adjustments between characters or for major positioning adjustments, such as moving the current text position to a new location to achieve the visual effect of a new line of text. Multi-line **'text'** elements are possible by defining different **'tspan'** elements for each line of text, with attributes **x**, **y**, **dx** and/or **dy** defining the position of each **'tspan'**. (An advantage of such an approach is that users will be able to perform multi-line text selection.)

In situations where micro-level positioning adjustment are necessary for advanced typographic control, the SVG content designer needs to ensure that the necessary font will be available for all viewers of the document (e.g., package up the necessary font data in the form of an SVG font or an alternative WebFont format which is stored at the same Web site as the SVG content) and that the viewing software will process the font in the expected way (the capabilities, characteristics and font layout mechanisms vary greatly from system to system). If the SVG content contains **x**, **y**, **dx** or **dy** attribute values which are meant to correspond to a particular font processed by a particular set of viewing software and either of these requirements is not met, then the text might display with poor quality.

The following additional rules apply to attributes **x**, **y**, **dx**, **dy** and **rotate** when they contain a list of numbers:

- When a single XML character maps to a single glyph - In this case, the **i**-th value for the **x**, **y**, **dx**, **dy** and **rotate** attributes is applied to the glyph that corresponds to the **i**-th character.
- When a single XML character maps to multiple glyphs (e.g., when an accent glyph is placed on top of a base glyph) - In this case, the **i**-th value for the **x**, **y**, **dx** and **dy** values are applied (i.e., the current text position is adjusted) before rendering the first glyph. The rotation transformation corresponding to the **i**-th **rotate** value is applied to the glyphs and to the inter-glyph advance values corresponding to this character on a group basis (i.e., the rotation value creates a temporary new rotated coordinate system, and the glyphs corresponding to the character are rendered into this rotated coordinate system).
- When multiple XML characters map to a single glyph (e.g., when a ligature is used) - Suppose that the **i**-th and **(i+1)**-th XML characters map to a single glyph. In this case, the **i**-th value for the **x**, **y**, **dx**, **dy** and **rotate** attributes all apply when rendering the glyph. The **(i+1)**-th values, however, for **x**, **y** and **rotate** are ignored (exception: the final **rotate** value in the list would still apply to subsequent characters), whereas the **dx** and **dy** are applied to the subsequent XML character (i.e., the **(i+2)**-th character), if one exists, by translating the current text position by the given amounts before rendering the first glyph associated with that character.
- When there is a many-to-many mapping of characters to glyphs (e.g., when three characters map to two glyphs, such as when the first glyph expresses the first character and half of the second character, and the second glyph expresses the other half of the second character plus the third character) - Suppose that the **i**-th, **(i+1)**-th and **(i+2)**-th XML characters map to two glyphs. In this case, the **i**-th value for the **x**, **y**, **dx** and **dy** values are applied (i.e., the current text position is

adjusted) before rendering the first glyph. The rotation transformation corresponding to the **i**-th **rotate** value is applied to both the two glyphs and the glyph advance values for the first glyph on a group basis (i.e., the rotation value creates a temporary new rotated coordinate system, and the two glyphs are rendered into the temporary rotated coordinate system). The **(i+1)**-th and **(i+2)**-th values, however, for the **x**, **y** and **rotate** attributes are not applied (exception: the final **rotate** value in the list would still apply to subsequent characters), whereas the **(i+1)**-th and **(i+2)**-th values for the **dx** and **dy** attributes are applied to the subsequent XML character (i.e., the **(i+3)**-th character), if one exists, by translating the current text position by the given amounts before rendering the first glyph associated with that character.

- Relationship to bidirectionality - As described below in the discussion on bidirectionality, text is laid out in a two-step process, where any bidirectional text is first re-ordered into a left-to-right string, and then text layout occurs with the re-ordered text string. Whenever the character data within a **'tspan'** element is re-ordered, the corresponding elements within the **x**, **y**, **dx**, **dy** and **rotate** are also re-ordered to maintain the correspondence. For example, suppose that you have the following **'tspan'** element:

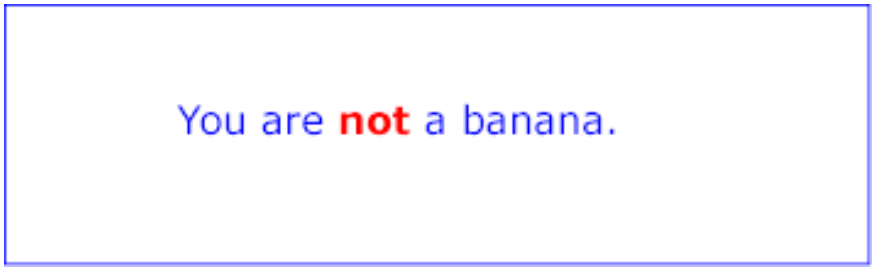
```
<tspan dx="11 12 13 14 15 0 21 22 23 0 31 32 33 34 35 36">Latin and Hebrew</span>
```

and that the word "Hebrew" will be drawn right-to-left. First, the character data and the corresponding values in the **dx** list will be reordered, such that the text string will be "Latin and werbeH" and the list of values for the **dx** attribute will be "11 12 13 14 15 0 21 22 23 0 36 35 34 33 32 31". After this re-ordering, the glyphs corresponding to the characters will be positioned using standard left-to-right layout rules.

The following examples show basic use of the **'tspan'** element.

Example tspan01 uses a **'tspan'** element to indicate that the word "not" is to use a bold font and have red fill.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example tspan01 - using tspan to change visual attributes</desc>

  <g font-family="Verdana" font-size="45" >
    <text x="200" y="150" fill="blue" >
      You are
        <tspan font-weight="bold" fill="red" >not</tspan>
      a banana.
    </text>
  </g>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```

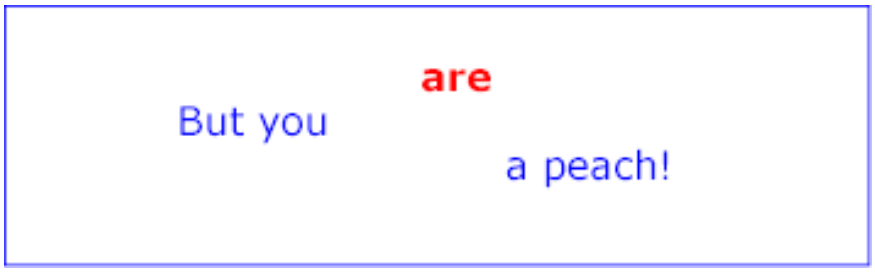


***Example tspan01***

*View this example as SVG (SVG-enabled browsers only)*

Example tspan02 uses the **dx** and **dy** attributes on the **'tspan'** element to adjust the current text position horizontally and vertically for particular text strings within a **'text'** element.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example tspan02 - using tspan's dx and dy attributes
        for incremental positioning adjustments</desc>

  <g font-family="Verdana" font-size="45" >
    <text x="200" y="150" fill="blue" >
      But you
        <tspan dx="2em" dy="-50" font-weight="bold" fill="red" >
          are
        </tspan>
        <tspan dy="100">
           a peach!
        </tspan>
    </text>
  </g>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```




***Example tspan02***

*View this example as SVG (SVG-enabled browsers only)*

Example tspan03 uses the **x** and **y** attributes on the **'tspan'** element to establish a new absolute current text position for each glyph to be rendered. The example shows two lines of text within a single **'text'** element. Because both lines of text are within the same **'text'** element, the user will be able to select through both lines of text and copy the text to the system clipboard in user agents that support text selection and clipboard operations,

```
<?xml version="1.0" standalone="no"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example tspan03 - using tspan's x and y attributes
        for multiline text and precise glyph positioning</desc>

  <g font-family="Verdana" font-size="45" >
    <text fill="rgb(255,164,0)" >
      <tspan x="300 350 400 450 500 550 600 650" y="100">
        Cute and
      </tspan>
      <tspan x="375 425 475 525 575" y="200">
         fuzzy
      </tspan>
    </text>
  </g>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```


***Example tspan03***

*View this example as SVG (SVG-enabled browsers only)*

## 10.6 The 'tref' element

The textual content for a **'text'** can be either character data directly embedded within the **'text'** element or the character data content of a referenced element, where the referencing is specified with a **'tref'** element.

```
<!ENTITY % trefExt "" >
<!ELEMENT tref (desc|title|metadata|animate|set|animateColor
               %trefExt;)* >
<!ATTLIST tref
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**

A URI reference to an element/fragment within an SVG document fragment whose character data content shall be used as character data for this **'tref'** element.
*Animatable: yes.*

***Attributes defined elsewhere:***

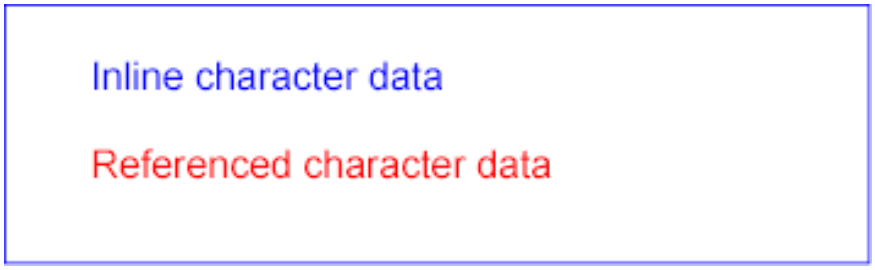
**%stdAttrs;, %xlinkRefAttrs;, %testAttrs;, %langSpaceAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements;, %graphicsElementEvents;, x, y, dx, dy, rotate, textLength, lengthAdjust.**

All character data within the referenced element, including character data enclosed within additional markup, will be rendered.

The **x**, **y**, **dx**, **dy** and **rotate** attributes have the same meanings as for the **'tspan'** element. The attributes are applied as if the **'tref'** element was replaced by a **'tspan'** with the referenced character data (stripped of all supplemental markup) embedded within the hypothetical **'tspan'** element.

Example tref01 shows how to use character data from a different element as the character data for a given **'tspan'** element. The first **'text'** element (with id="ReferencedText") will not draw because it is part of a **'defs'** element. The second **'text'** element draws the string "Inline character data". The third **'text'** element draws the string "Reference character data" because it includes a **'tref'** element which is a reference to element "ReferencedText", and that element's character data is "Referenced character data".

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <text id="ReferencedText">
      Referenced character data
    </text>
  </defs>
  <desc>Example tref01 - inline vs reference text content</desc>

  <text x="100" y="100" font-size="45" fill="blue" >
    Inline character data
  </text>
  <text x="100" y="200" font-size="45" fill="red" >
    <tref xlink:href="#ReferencedText"/>
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```




***Example tref01***

*View this example as SVG (SVG-enabled browsers only)*

## 10.7 Text layout

### 10.7.1 Text layout introduction

This section describes the text layout features supported by SVG, which includes support for various international writing directions, such as left-to-right (e.g., Latin scripts) and bidirectional (e.g., Hebrew or Arabic) and vertical (e.g., Asian scripts). The descriptions in this section assume straight line text (i.e., text that is either strictly horizontal or vertical with respect to the current user coordinate system). Subsequent sections describe the supplemental layout rules for text on a path.

SVG does not provide for automatic line breaks or word wrapping, which makes internationalized text layout for SVG relatively simpler than it is for languages which support formatting of multi-line text blocks.

For each **'text'** element, the SVG user agent determines the current **reference orientation**. For standard horizontal or vertical text (i.e., no text-on-a-path), the reference orientation is the vector pointing towards negative infinity in Y within the current user coordinate system. (Note: in the initial coordinate system, the reference orientation is up.) For text on a path, the reference orientation is reset with each character.

Based on the reference orientation and the value for property **'writing-mode'**, the SVG user agent determines the current **inline-progression-direction**. For left-to-right text, the inline-progression-direction points 90 degrees clockwise from the reference orientation vector. For right-to-left text, the inline progression points 90 degrees counter-clockwise from the reference orientation vector. For top-to-bottom text, the inline-progression-direction points 180 degrees from the reference orientation vector.

Based on the reference orientation and the value for property **'writing-mode'**, the SVG user agent determines the current **block-progression-direction**. For left-to-right and right-to-left text, the block-progression-direction points 180 degrees from the reference orientation vector because the only available horizontal **'writing-mode'**s are **lr-tb** and **rl-tb**. For top-to-bottom text, the block-progression-direction always points 90 degrees counter-clockwise from the reference orientation vector because the only available top-to-bottom **'writing-mode'** is **tb-rl**.

The **shift direction** is the direction towards which the baseline table moves due to positive values for property **'baseline-shift'**. The shift direction is such that a positive value shifts the baseline table towards the topmost entry in the parent's baseline table.

In processing a given **'text'** element, the SVG user agent keeps track of the **current text position**. The initial current text position is established by the **x** and **y** attributes on the **'text'** element.

The current text position is adjusted after each glyph to establish a new current text position at which the next glyph shall be rendered. The adjustment to the current text position is based on the current inline-progression-direction, glyph-specific advance values corresponding to the glyph orientation of the glyph just rendered, kerning tables in the font and the current values of various attributes and properties, such as the spacing properties and any **x**, **y**, **dx** and **dy** attributes on **'text'**, **'tspan'**, **'tref'** or **'altGlyph'** elements. If a glyph does not provide explicit advance values corresponding to the current glyph orientation, then an appropriate approximation should be used. For vertical text, a suggested approximation is the sum of the ascent and descent values for the glyph. Another suggested approximation for an advance value for both horizontal and vertical text is the size of an *em* (see units-per-em).

For each glyph to be rendered, the SVG user agent determines an appropriate **alignment-point** on the glyph which will be placed exactly at the current text position. The alignment-point is determined based on glyph cell metrics in the glyph itself, the current inline-progression-direction and the glyph orientation relative to the inline-progression-direction. For most uses of Latin text (i.e., **'writing-mode:lr'**, **'text-anchor:start'**, and **'alignment-baseline:baseline'**) the alignment-point in the glyph will be the intersection of left edge of the glyph cell (or some other glyph-specific x-axis coordinate indicating a left-side origin point) with the Latin baseline of the glyph. For many cases with top-to-bottom vertical text layout, the reference point will be either a glyph-specific origin point based on the set of vertical baselines for the font or the intersection of the center of the glyph with its *top line* (see [CSS2-topline] for a definition of *top line*). If a glyph does not provide explicit origin points corresponding to the current glyph orientation, then an appropriate approximation should be used, such as the intersection of the left edge of the glyph with the appropriate horizontal baseline for the glyph or intersection of the top edge of the glyph with the appropriate vertical baseline. If baseline tables are not available, user agents should establish baseline tables that reflect common practice.

Adjustments to the current text position are either **absolute position adjustments** or **relative position adjustments**. An absolute position adjustment occurs in the following circumstances:

- At the start of a **'text'** element

- At the start of each **'textPath'** element
- For each character within a **'text'**, **'tspan'**, **'tref'** and **'altGlyph'** element which has an **x** or **y** attribute value assigned to it explicitly

All other position adjustments to the current text position are relative position adjustments.

Each absolute position adjustment defines a new **text chunk**. Absolute position adjustments impact text layout in the following ways:

- Ligatures only occur when a set of characters which might map to a ligature are all in the same text chunk.
- Each text chunk represents a separate block of text for alignment due to **'text-anchor'** property values.
- Reordering of characters due to bidirectionality only occurs within a text chunk. Reordering does *not* happen across text chunks.

The following additional rules apply to ligature formation:

- As in [CSS2-spacing], when the resultant space between two characters is not the same as the default space, user agents should not use ligatures; thus, if there are non-default values for properties **'kerning'** or **'letter-spacing'**, the user agent should not use ligatures.
- Ligature formation should not be enabled for the glyphs corresponding to characters within different DOM text nodes; thus, characters separated by markup should not use ligatures.
- As mentioned above, ligature formation should not be enabled for the glyphs corresponding to characters within different text chunks.

## 10.7.2 Setting the inline-progression-direction

The **'writing-mode'** property specifies whether the initial inline-progression-direction for a **'text'** element shall be left-to-right, right-to-left, or top-to-bottom. The **'writing-mode'** property applies only to **'text'** elements; the property is ignored for **'tspan'**, **'tref'**, **'altGlyph'** and **'textPath'** sub-elements. (Note that the inline-progression-direction can change within a **'text'** element due to the Unicode bidirectional algorithm and properties **'direction'** and **'unicode-bidi'**. For more on bidirectional text, see Relationship with bidirectionality.)

**'writing-mode'**

| | |
|---|---|
| *Value:* | lr-tb \| rl-tb \| tb-rl \| lr \| rl \| tb \| inherit |
| *Initial:* | lr-tb |
| *Applies to:* | **'text'** elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | no |

**lr-tb | lr**

Sets the initial inline-progression-direction to left-to-right, as is common in most Latin-based documents. For most characters, the *current text position* is advanced from left to right after each glyph is rendered. (When the character data includes characters which are subject to the Unicode bidirectional algorithm, the text advance rules are more complex. See Relationship with bidirectionality).

**rl-tb | rl**

Sets the initial inline-progression-direction to right-to-left, as is common in Arabic or Hebrew scripts. (See Relationship with bidirectionality.)

**tb-rl | tb**

Sets the initial inline-progression-direction to top-to-bottom, as is common in some Asian scripts, such as Chinese and Japanese. Though hardly as frequent as horizontal, this type of vertical layout also occurs in Latin based documents, particularly in table column or row labels. In most cases, the vertical baselines running through the middle of each glyph are aligned.

## 10.7.3 Glyph orientation within a text run

In some cases, it is required to alter the orientation of a sequence of characters relative to the inline-progression-direction. The requirement is particularly applicable to vertical layouts of East Asian documents, where sometimes narrow-cell Latin text is to be displayed horizontally and other times vertically.

Two properties control the glyph orientation relative to the reference orientation for each of the two possible inline-progression-directions. **'glyph-orientation-vertical'** controls glyph orientation when the inline-progression-direction is vertical. **'glyph-orientation-horizontal'** controls glyph orientation when the inline-progression-direction is horizontal.

**'glyph-orientation-vertical'**

| | |
|---|---|
| *Value:* | auto \| <angle> \| inherit |
| *Initial:* | auto |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | no |

**auto**

- Fullwidth ideographic and fullwidth Latin text will be set with a glyph-orientation of 0-degrees.

  Ideographic punctuation and other ideographic characters having alternate horizontal and vertical forms will use the vertical form of the glyph.

- Text which is not fullwidth will be set with a glyph-orientation of 90-degrees.

  This reorientation rule applies only to the first-level non-ideographic text. All further embedding of writing-modes or bidi processing will be based on the first-level rotation.

  **NOTE:**

  - This is equivalent to having set the non-ideographic text string horizontally honoring the bidi-rule, then rotating the resultant sequence of inline-areas (one area for each change of glyph direction) 90-degrees clockwise.

    It should be noted that text set in this "rotated" manner may contain ligatures or other glyph combining and reordering common to the language and script. (This "rotated" presentation form does not disable auto-ligature formation or similar context-driven

variations.)

- The determination of which characters should be auto-rotated may vary across user agents. The determination is based on a complex interaction between country, language, script, character properties, font, and character context. It is suggested that one consult the Unicode TR 11 and the various JIS or other national standards.

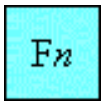
**<angle>**

The value of the angle is restricted to 0, 90, 180, and 270 degrees. The user agent shall round the value of the angle to the closest of the permitted values.
A value of **0deg** indicates that all glyphs are set with the top of the glyphs oriented towards the reference orientation. A value of **90deg** indicates an orientation of 90 degrees clockwise from the reference orientation.

This property is applied only to text written in a vertical **'writing-mode'**.

The glyph orientation affects the amount that the current text position advances as each glyph is rendered. When the inline-progression-direction is vertical and the **'glyph-orientation-vertical'** results in an orientation angle that is a multiple of 180 degrees, then the current text position is incremented according to the vertical metrics of the glyph. Otherwise, if the **'glyph-orientation-vertical'** results in an orientation angle that is not a multiple of 180 degrees, then the current text position is incremented according to the horizontal metrics of the glyph.

The text layout diagrams in this section use the following symbols:

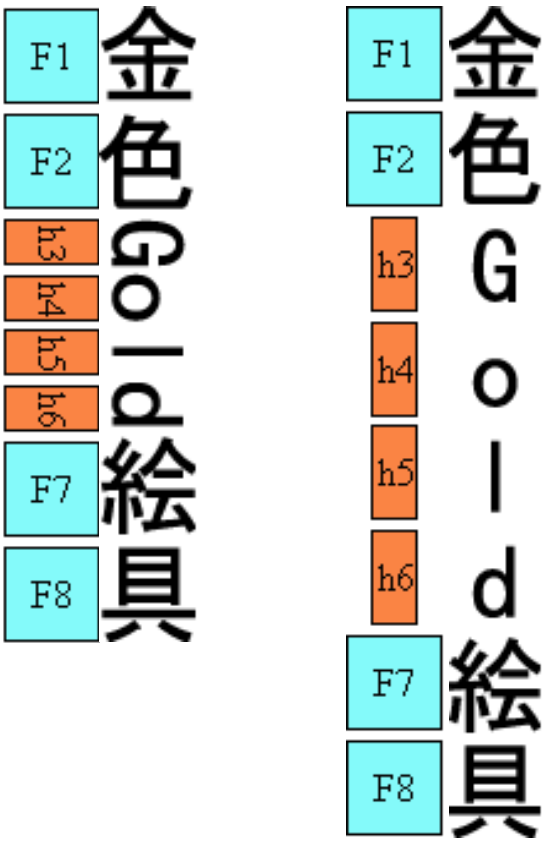


- wide-cell glyph (e.g. Han) which is the *n*-th glyph in the text run

- narrow-cell glyph (e.g. Latin) which is the *n*-th glyph in the text run

The orientation which the above symbols assume in the diagrams corresponds to the orientation that the Unicode characters they represent are intended to assume when rendered in the user agent. Spacing between the glyphs in the diagrams is usually symbolic, unless intentionally changed to make a point.

The diagrams below illustrate different uses of **'glyph-orientation-vertical'**. The diagram on the left shows the result of the mixing of full-width ideographic glyphs with narrow-cell Latin glyphs when **'glyph-orientation-vertical'** for the Latin characters is either **auto** or **90**. The diagram on the right show the result of mixing full-width ideographic glyphs with narrow-cell Latin glyphs when Latin glyphs are specified to have a **'glyph-orientation-vertical'** of **0**.




**'glyph-orientation-horizontal'**

| | |
|---|---|
| *Value:* | <angle> \| inherit |
| *Initial:* | 0deg |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | no |

**<angle>**

The value of the angle is restricted to 0, 90, 180, and 270 degrees. The user agent shall round the value of the angle to the closest of the permitted values.
A value of **0deg** indicates that all glyphs are set with the top of the glyphs oriented towards the reference orientation. A value of **90deg** indicates an orientation of 90 degrees clockwise from the reference orientation.

This property is applied only to text written in a horizontal **'writing-mode'**.

The glyph orientation affects the amount that the current text position advances as each glyph is rendered. When the reference orientation direction is horizontal and the **'glyph-orientation-horizontal'** results in an orientation angle that is a multiple of 180 degrees, then the current text position is incremented according to the horizontal metrics of the glyph. Otherwise, if the **'glyph-orientation-horizontal'** results in an orientation angle that is not a multiple of 180 degrees, then the current text position is incremented according to the vertical metrics of the glyph.

## 10.7.4 Relationship with bidirectionality

The characters in certain scripts are written from right to left. In some documents, in particular those written with the Arabic or Hebrew script, and in some mixed-language contexts, text in a single line may appear with mixed directionality. This phenomenon is called bidirectionality, or "bidi" for short.

The Unicode standard ([UNICODE], section 3.11) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. The SVG user agent applies this bidirectional algorithm when determining the layout of characters within a **'text'** element. The **'direction'** and **'unicode-bidi'** properties allow authors to override the inherent directionality of the content characters and thus explicitly control how the elements and attributes of a document language map to this algorithm. These two properties are applicable to all characters who glyphs are perpendicular to the inline-progression-direction.

In most cases, the bidirectional algorithm from [UNICODE] produces the desired result automatically, and overriding this algorithm properly is usually quite complex. Therefore, in most cases, authors are discouraged from assigning values to these properties.

A more complete discussion of bidirectionality can be found in the "Cascading Style Sheets (CSS) level 2" specification [CSS2-direction].

The processing model for bidirectional text is as follows. The user agent processes the characters which are provided in **logical order** (i.e., the order the characters appear in the original document, either via direct inclusion or via indirect reference due a **'tref'** element). The user agent determines the set of independent blocks within each of which it should apply the Unicode bidirectional algorithm. Each text chunk represents an independent block of text. Additionally, any change in glyph orientation due to processing of properties **'glyph-orientation-horizontal'** or **'glyph-orientation-vertical'** will subdivide the independent blocks of text further. After processing the Unicode bidirectional algorithm and properties **'direction'** and **'unicode-bidi'** on each of the independent text blocks, the user agent will have a potentially re-ordered list of characters which are now in left-to-right rendering order. Simultaneous with re-ordering of the characters, the **dx**, **dy** and **rotate** attributes on the **'tspan'** and **'tref'** elements are also re-ordered to maintain the original correspondence between characters and attribute values. While kerning or ligature processing might be font-specific, the preferred model is that kerning and ligature processing occurs between combinations of characters or glyphs after the characters have been re-ordered.

**'direction'**

| | |
|---|---|
| *Value:* | ltr \| rtl \| inherit |
| *Initial:* | ltr |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | no |

This property specifies the base writing direction of text and the direction of embeddings and overrides (see **'unicode-bidi'**) for the Unicode bidirectional algorithm. For the **'direction'** property to have any effect, the **'unicode-bidi'** property's value must be 'embed' or 'bidi-override'.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-direction].

The **'direction'** property applies only to glyphs oriented perpendicular to the inline-progression-direction,

which includes the usual case of horizontally-oriented Latin or Arabic text and the case of narrow-cell Latin or Arabic characters rotated 90 degrees clockwise relative to a top-to-bottom inline-progression-direction.

**'unicode-bidi'**

| | |
|---|---|
| *Value:* | normal \| embed \| bidi-override \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | no |

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-unicode-bidi].

## 10.8 Text rendering order

The glyphs associated with the characters within a **'text'** element are rendered in the logical order of the characters in the original document, independent of any re-ordering necessary to implement bidirectionality. Thus, for text that goes right-to-left visually, the glyphs associated with the rightmost character are rendered before the glyphs associated with the other characters.

Additionally, each distinct glyph is rendered in its entirety (i.e., it is filled and stroked as specified by the **'fill'** and **'stroke'** properties) before the next glyph gets rendered.

## 10.9 Alignment properties

### 10.9.1 Text alignment properties

The **'text-anchor'** property is used to align (start-, middle- or end-alignment) a string of text relative to a given point.

The **'text-anchor'** property is applied to each individual text chunk within a given **'text'** element. Each text chunk has an initial current text position, which represents the point in the user coordinate system resulting from (depending on context) application of the **x** and **y** attributes on the **'text'** element, any **x** or **y** attribute values on a **'tspan'**, **'tref'** or **'altGlyph'** element assigned explicitly to the first rendered character in a text chunk, or determination of the initial current text position for a **'textPath'** element.

**'text-anchor'**

| | |
|---|---|
| *Value:* | start \| middle \| end \| inherit |
| *Initial:* | start |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

Values have the following meanings:

**start**
: The rendered characters are aligned such that the start of the text string is at the initial current text position. For Latin or Arabic, which is usually rendered horizontally, this is comparable to left alignment. For Asian text with a vertical primary text direction, this is comparable to top alignment.

**middle**
: The rendered characters are aligned such that the middle of the text string is at the current text position. (For text on a path, conceptually the text string is first laid out in a straight line. The midpoint between the start of the text string and the end of the text string is determined. Then, the text string is mapped onto the path with this midpoint placed at the current text position.)

**end**
: The rendered characters are aligned such that the end of the text string is at the initial current text position. For Latin text in its usual orientation, this is comparable to right alignment.

### 10.9.2 Baseline alignment properties

An overview of baseline alignment and baseline tables can be found above in Fonts, font tables and baselines.

One of the characteristics of international text is that there are different baselines (different alignment points) for glyphs in different scripts. For example, in horizontal writing, ideographic scripts, such as Han Ideographs, Katakana, Hiragana, and Hangul, alignment occurs with a baseline near the bottoms of the glyphs; alphabetic based scripts, such as Latin, Cyrillic, Hebrew, Arabic, align a point that is the bottom of most glyphs, but some glyphs descend below the baseline; and Indic based scripts are aligned at a point that is near the top of the glyphs.

When different scripts are mixed on a line of text, an adjustment must be made to ensure that the glyphs in the different scripts are aligned correctly with one another. OpenType [OPENTYPE] fonts have a Baseline table (BASE) [OPENTYPE-BASETABLE] that specifies the offsets of the alternative baselines from the current baseline.

SVG uses a similar baseline table model that assumes one script (at one font-size) is the "dominant run" during processing of a **'text'** element; that is, all other baselines are defined in relation to this dominant run. The baseline of the script with the dominant run is called the **dominant baseline**. So, for example, if the dominant baseline is the alphabetic baseline, there will be offsets in the baseline table for the alternate baselines, such as the ideographic baseline and the Indic baseline. There will also be an offset for the math baseline which is used for some math fonts. Note that there are separate baseline tables for horizontal and vertical writing-modes. The offsets in these tables may be different for horizontal and vertical writing.

The baseline table established at the start of processing of a **'text'** element is called the **dominant baseline table**.

Because the value of the **'font-family'** property is a list of fonts, to insure a consistent choice of baseline table we define the *nominal font* in a font list as the first font in the list for which a glyph is available. This is the first font that could contain a glyph for each character encountered. (For this definition, glyph data is assumed to be present if a font substitution is made or if the font is synthesized.) This definition insures a content independent determination of the font and baseline table that is to be used.

The value of the **'font-size'** property on the **'text'** element establishes the **dominant baseline table font size**.

The model assumes that each glyph has a 'alignment-baseline' value which specifies the baseline with

which the glyph is to be aligned. (The 'alignment-baseline' is called the "Baseline Tag" in the OpenType baseline table description.) The initial value of the **'alignment-baseline'** property uses the baseline identifier associated with the given glyph. Alternate values for **'alignment-baseline'** can be useful for glyphs such as a "*" which are ambiguous with respect to script membership.

The model assumes that the font from which the glyph is drawn also has a baseline table, the **font baseline table**. This baseline table has offsets in units-per-em from the (0,0) point to each of the baselines the font knows about. In particular, it has the offset from the glyph's (0,0) point to the baseline identified by the 'alignment-baseline'.

The offset values in the baseline table are in "design units" which means fractional units of the EM. CSS calls these "units-per-em" [CSS2-UNITSPEREM]. Thus, the current **'font-size'** is used to determine the actual offset from the dominant baseline to the alternate baselines.

The glyph is aligned so that its baseline identified by its 'alignment-baseline' is aligned with the baseline with the same name from the dominant baseline table.

The offset from the dominant baseline of the parent to the baseline identified by the 'alignment-baseline' is computed using the dominant baseline table and dominant baseline table font size. The font baseline table and font size applicable to the glyph are used to compute the offset from the identified baseline to the (0,0) point of the glyph. This second offset is subtracted from the first offset to get the position of the (0,0) point in the shift direction. Both offsets are computed by multiplying the baseline value from the baseline table times the appropriate font size value.

If the 'alignment-baseline' identifies the dominant baseline, then the first offset is zero and the glyph is aligned with the dominant baseline; otherwise, the glyph is aligned with the chosen alternate baseline.

The baseline-identifiers below are used in this specification. Some of these are determined by baseline-tables contained in a font as described in **[XSL description of Fonts and Font Data]**. Others are computed from other font characteristics as described below.

**alphabetic**

> This identifies the baseline used by most alphabetic and syllabic scripts. These include, but are not limited to, many Western, Southern Indic, Southeast Asian (non-ideographic) scripts.

**ideographic**

> This identifies the baseline used by ideographic scripts. For historical reasons, this baseline is at the bottom of the ideographic EM box and not in the center of the ideographic EM box. See the "central" baseline. The ideographic scripts include Chinese, Japanese, Korean, and Vietnamese Chu Nom.

**hanging**

> This identifies the baseline used by certain Indic scripts. These scripts include Devanagari, Gurmukhi and Bengali.

**mathematical**

This identifies the baseline used by mathematical symbols.

**central**

This identifies a computed baseline that is at the center of the EM box. This baseline lies halfway between the text-before-edge and text-after-edge baselines.

**NOTE:**

For ideographic fonts, this baseline is often used to align the glyphs; it is an alternative to the ideographic baseline.

**middle**

This identifies a baseline that is offset from the alphabetic baseline in the **shift-direction** by 1/2 the value of the x-height font characteristic. The position of this baseline may be obtained from the font data or, for fonts that have a font characteristic for "x-height", it may be computed using 1/2 the "x-height". Lacking either of these pieces of information, the position of this baseline may be approximated by the "central" baseline.

**text-before-edge**

This identifies the before-edge of the EM box. The position of this baseline may be specified in the baseline-table or it may be calculated.

**NOTE:**

The position of this baseline is normally around or at the top of the ascenders, but it may not encompass all accents that can appear above a glyph. For these fonts the value of the "ascent" font characteristic is used. For ideographic fonts, the position of this baseline is normally 1 EM in the **shift-direction** from the "ideographic" baseline. However, some ideographic fonts have a reduced width in the inline-progression-direction to allow tighter setting. When such a font, designed only for vertical writing-modes, is used in a horizontal writing-mode, the "text-before-edge" baseline may be less than 1 EM from the text-after-edge.

**text-after-edge**

This identifies the after-edge of the EM box. The position of this baseline may be specified in the baseline-table or it may be calculated.

**NOTE:**

For fonts with descenders, the position of this baseline is normally around or at the bottom of the descenders. For these fonts the value of the "descent" font characteristic is used. For ideographic fonts, the position of this baseline is normally at the "ideographic" baseline.

There are, in addition, two computed baselines that are only defined for line areas. Since SVG does not support the notion of computations based on line areas, the two computed baselines are mapped as follows:

**before-edge**
For SVG, this is equivalent to **text-before-edge**.
**after-edge**
For SVG, this is equivalent to **text-after-edge**.

There are also four baselines that are defined only for horizontal writing-modes.

**top**

This baseline is the same as the "before-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

**text-top**

This baseline is the same as the "text-before-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

**bottom**

This baseline is the same as the "after-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

**text-bottom**

This baseline is the same as the "text-after-edge" baseline in a horizontal writing-mode and is undefined in a vertical writing mode.

The baseline-alignment properties follow.

**'dominant-baseline'**

| | |
|---|---|
| *Value:* | auto \| use-script \| no-change \| reset-size \| alphabetic \| hanging \| ideographic \| mathematical \| central \| middle \| text-after-edge \| text-before-edge \| text-top \| text-bottom \| inherit |
| *Initial:* | auto |
| *Applies to:* | text content elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The "dominant-baseline" property is used to determine or re-determine a scaled-baseline-table. A scaled-baseline-table is a compound value with three components: a baseline-identifier for the dominant-baseline, a baseline-table and a baseline-table font-size. Some values of the property re-determine all three values; other only re-establish the baseline-table font-size. When the initial value, "auto", would give an undesired result, this property can be used to explicitly set the desire scaled-baseline-table.

Values for the property have the following meaning:

**auto**

If this property occurs on a **'text'** element, then the computed value depends on the value of the **'writing-mode'** property. If the 'writing-mode' is horizontal, then the value of the dominant-baseline component is 'alphabetic', else if the 'writing-mode' is vertical, then the value of the dominant-baseline component is 'central'.

If this property occurs on a **'tspan'**, **'tref'**, **'altGlyph'** or **'textPath'** element, then the dominant-baseline and the baseline-table components remain the same as those of the parent text content element. If the computed 'baseline-shift' value actually shifts the baseline, then the baseline-table font-size component is set to the value of the **'font-size'** property on the element on which the **'dominant-baseline'** property occurs, otherwise the baseline-table font-size remains the same as that of the element. If there is no parent text content element, the scaled-baseline-table value is constructed as above for **'text'** elements.

**use-script**
The dominant-baseline and the baseline-table components are set by determining the predominant script of the character data content. The 'writing-mode', whether horizontal or vertical, is used to select the appropriate set of baseline-tables and the dominant baseline is used to select the baseline-table that corresponds to that baseline. The baseline-table font-size component is set to the value of the **'font-size'** property on the element on which the **'dominant-baseline'** property occurs.

**no-change**
The dominant-baseline, the baseline-table, and the baseline-table font-size remain the same as that of the parent text content element.

**reset-size**
The dominant-baseline and the baseline-table remain the same, but the baseline-table font-size is changed to the value of the **'font-size'** property on this element. This re-scales the baseline-table for the current **'font-size'**.

**alphabetic**
The baseline-identifier for the dominant-baseline is set to be 'alphabetic', the derived baseline-table is constructed using the 'alphabetic' baseline-table in the nominal font, and the baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**hanging**
The baseline-identifier for the dominant-baseline is set to be 'hanging', the derived baseline-table is constructed using the 'hanging' baseline-table in the nominal font, and the baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**ideographic**
The baseline-identifier for the dominant-baseline is set to be 'ideographic', the derived baseline-table is constructed using the 'ideographic' baseline-table in the nominal font, and the baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**mathematical**
The baseline-identifier for the dominant-baseline is set to be 'mathematical', the derived baseline-table is constructed using the 'mathematical' baseline-table in the nominal font, and the baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**central**
The baseline-identifier for the dominant-baseline is set to be 'central'. The derived baseline-table is constructed from the defined baselines in a baseline-table in the nominal font. That font baseline-table is chosen using the following priority order of baseline-table names: 'ideographic', 'alphabetic', 'hanging', 'mathematical'. The baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**middle**

The baseline-identifier for the dominant-baseline is set to be 'middle'. The derived baseline-table is constructed from the defined baselines in a baseline-table in the nominal font. That font baseline - table is chosen using the following priority order of baseline-table names: 'alphabetic', 'ideographic', 'hanging', 'mathematical'. The baseline-table font-size is changed to the value of the **'font-size'** property on this element.

**text-after-edge**

The baseline-identifier for the dominant-baseline is set to be 'text-after-edge'. The derived baseline-table is constructed from the defined baselines in a baseline-table in the nominal font. The choice of which font baseline-table to use from the baseline-tables in the nominal font is implementation defined. The baseline-table font-size is changed to the value of the **'font-size'** property on this element.

NOTE: using the following priority order of baseline-table names: 'alphabetic', 'ideographic', 'hanging', 'mathematical' is probably a reasonable strategy for determining which font baseline-table to use.

**text-before-edge**

The baseline-identifier for the dominant-baseline is set to be 'text-before-edge'. The derived baseline-table is constructed from the defined baselines in a baseline-table in the nominal font. The choice of which baseline-table to use from the baseline-tables in the nominal font is implementation defined. The baseline-table font-size is changed to the value of the **'font-size'** property on this element.

NOTE: Using the following priority order of baseline-table names: 'alphabetic', 'ideographic', 'hanging', 'mathematical' is probably a reasonable strategy for determining which font baseline-table to use.

**text-top**

See definition of 'text-top' baseline.

**text-bottom**

See definition of 'text-bottom' baseline.

If there is no baseline table in the nominal font or if the baseline table lacks an entry for the desired baseline, then the user agent may use heuristics to determine the position of the desired baseline.

**'alignment-baseline'**

| | |
|---|---|
| *Value:* | auto \| baseline \| before-edge \| text-before-edge \| middle \| after-edge \| text-after-edge \| ideographic \| alphabetic \| hanging \| mathematical \| inherit |
| *Initial:* | auto |
| *Applies to:* | **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property specifies how an object is aligned with respect to its parent. This property specifies which baseline of this element is to be aligned with the corresponding baseline of the parent. For example, this allows alphabetic baselines in Roman text to stay aligned across font size changes. It defaults to the baseline with the same name as the computed value of the alignment-baseline property. That is, the position of "ideographic" alignment-point in the **block-progression-direction** is the position of the "ideographic" baseline in the baseline-table of the object being aligned.

Values have the following meanings:

**auto**
The value is the dominant-baseline of the script to which the character belongs - i.e., use the dominant-baseline of the parent.

**baseline**
The alignment-point of the object being aligned is aligned with the dominant-baseline of the parent text content element.

**before-edge**
The alignment-point of the object being aligned is aligned with the "before-edge" baseline of the parent text content element.

**text-before-edge**
The alignment-point of the object being aligned is aligned with the "text-before-edge" baseline of the parent text content element.

**central**
The alignment-point of the object being aligned is aligned with the "central" baseline of the parent text content element.

**middle**
The alignment-point of the object being aligned is aligned with the "middle" baseline of the parent text content element.

**after-edge**
The alignment-point of the object being aligned is aligned with the "after-edge" baseline of the parent text content element.

**text-after-edge**
The alignment-point of the object being aligned is aligned with the "text-after-edge" baseline of the parent text content element.

**ideographic**
The alignment-point of the object being aligned is aligned with the "ideographic" baseline of the parent text content element.

**alphabetic**
The alignment-point of the object being aligned is aligned with the "alphabetic" baseline of the parent text content element.

**hanging**
The alignment-point of the object being aligned is aligned with the "hanging" baseline of the parent text content element.

**mathematical**
The alignment-point of the object being aligned is aligned with the "mathematical" baseline of the parent text content element.

**top**
The alignment-point of the object being aligned is aligned with the "top" baseline of the parent text content element if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

**bottom**
The alignment-point of the object being aligned is aligned with the "bottom" baseline of the parent text content element if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

**text-top**
The alignment-point of the object being aligned is aligned with the "text-top" baseline of the parent text content element if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

**text-bottom**
The alignment-point of the object being aligned is aligned with the "text-bottom" baseline of the parent text content element if the writing-mode is horizontal. Otherwise, the dominant-baseline is used.

**'baseline-shift'**

| | |
|---|---|
| *Value:* | baseline \| sub \| super \| <percentage> \| <length> \| inherit |
| *Initial:* | baseline |
| *Applies to:* | **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** elements |
| *Inherited:* | no |
| *Percentages:* | refers to the "line-height" of the **'text'** element, which in the case of SVG is defined to be equal to the **'font-size'** |
| *Media:* | visual |
| *Animatable:* | yes |

The **'baseline-shift'** property allows repositioning of the dominant-baseline relative to the dominant-baseline of the parent text content element. The shifted object might be a sub- or superscript. Within the shifted object, the whole baseline-table is offset; not just a single baseline. The amount of the shift is determined from information from the parent text content element, the sub- or superscript offset from the nominal font of the parent text content element, percent of the "line-height" of the parent text content element or an absolute value.

In SVG, the **'baseline-shift'** property represents a supplemental adjustment to the baseline tables. The **'baseline-shift'** property shifts the baseline tables for each glyph to temporary new positions, for example to lift the glyph into superscript or subscript position, but it does not effect the current text position. When the current text position is adjusted after rendering a glyph to take into account glyph advance values, the adjustment happens as if there were no baseline shift.

**'baseline-shift'** properties can nest. Each nested **'baseline-shift'** is added to previous baseline shift values.

Values for the property have the following meaning:

**baseline**
: There is no baseline shift; the dominant-baseline remains in its original position.

**sub**
: The dominant-baseline is shifted to the default position for subscripts. The offset to this position is determined using the font data for the nominal font. Because in most fonts the subscript position is normally given relative to the "alphabetic" baseline, the user agent may compute the effective position for subscripts for superscripts when some other baseline is dominant. The suggested computation is to subtract the difference between the position of the dominant baseline and the position of the "alphabetic" baseline from the position of the subscript. The resulting offset is determined by multiplying the effective subscript position by the dominant baseline-table font-size. If there is no applicable font data the user agent may use heuristics to determine the offset.

**super**
: The dominant-baseline is shifted to the default position for superscripts. The offset to this position is determined using the font data for the nominal font. Because in most fonts the superscript position is normally given relative to the "alphabetic" baseline, the user agent may compute the effective position for superscripts when some other baseline is dominant. The suggested computation is to subtract the difference between the position of the dominant baseline and the position of the "alphabetic" baseline from the position of the superscript. The resulting offset is determined by multiplying the effective superscript position by the dominant baseline-table font-size. If there is no applicable font data the user agent may use heuristics to determine the offset.

**<percentage>**
: The computed value of the property is this percentage multiplied by the computed "line-height" of the **'text'** element. The dominant-baseline is shifted in the shift direction (positive value) or opposite to the shift direction (negative value) of the parent text content element by the computed value. A value of "0%" is equivalent to "baseline".

**<length>**

The dominant-baseline is shifted in the shift direction (positive value) or opposite to the shift direction (negative value) of the parent text content element by the <length> value. A value of "0cm" is equivalent to "baseline".

## 10.10 Font selection properties

SVG uses the following font specification properties. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-fonts]. Any SVG-specific notes about these properties are contained in the descriptions below.

**'font-family'**

| | |
|---|---|
| *Value:* | [[ <family-name> \| <generic-family> ],]* [<family-name> \| <generic-family>] \| inherit |
| *Initial:* | depends on user agent |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property indicates which font family is to be used to render the text, specified as a prioritized list of font family names and/or generic family names. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-family]. The rules for expressing the syntax of CSS property values can be found at [CSS2-propdef].

**'font-style'**

| | |
|---|---|
| *Value:* | normal \| italic \| oblique \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property specifies whether the text is to be rendered using a normal, italic or oblique face. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-style].

**'font-variant'**

| | |
|---|---|
| *Value:* | normal \| small-caps \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property indicates whether the text is to be rendered using the normal glyphs for lowercase characters or using small-caps glyphs for lowercase characters. Except for any additional information provided in this

specification, the normative definition of the property is in [CSS2-font-variant].

**'font-weight'**

| | |
|---|---|
| *Value:* | normal \| bold \| bolder \| lighter \| 100 \| 200 \| 300 \| 400 \| 500 \| 600 \| 700 \| 800 \| 900 \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property refers to the boldness or lightness of the glyphs used to render the text, relative to other fonts in the same font family. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-weight].

**'font-stretch'**

| | |
|---|---|
| *Value:* | normal \| wider \| narrower \| ultra-condensed \| extra-condensed \| condensed \| semi-condensed \| semi-expanded \| expanded \| extra-expanded \| ultra-expanded \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property indicates the desired amount of condensing or expansion in the glyphs used to render the text. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-stretch].

**'font-size'**

| | |
|---|---|
| *Value:* | <absolute-size> \| <relative-size> \| <length> \| <percentage> \| inherit |
| *Initial:* | medium |
| *Applies to:* | text content elements |
| *Inherited:* | yes, the computed value is inherited |
| *Percentages:* | refer to parent element's font size |
| *Media:* | visual |
| *Animatable:* | yes |

This property refers to the size of the font from baseline to baseline when multiple lines of text are set solid in a multiline layout environment. For SVG, if a **<length>** is provided without a unit identifier (e.g., an unqualified number such as **128**), the SVG user agent processes the **<length>** as a height value in the current user coordinate system.

If a **<length>** is provided with one of the unit identifiers (e.g., **12pt** or **10%**), then the SVG user agent converts the **<length>** into a corresponding value in the current user coordinate system by applying the rules described in Units.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-size].

**'font-size-adjust'**

| | |
|---|---|
| *Value:* | <number> \| none \| inherit |
| *Initial:* | none |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes (non-additive, 'set' and 'animate' elements only) |

This property allows authors to specify an aspect value for an element that will preserve the x-height of the first choice font in a substitute font. Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font-size-adjust].

**'font'**

| | |
|---|---|
| *Value:* | [ [ <'font-style'> \|\| <'font-variant'> \|\| <'font-weight'> ]? <'font-size'> [ / <'line-height'> ]? <'font-family'> ] \| caption \| icon \| menu \| message-box\| small-caption \| status-bar \| inherit |
| *Initial:* | see individual properties |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | allowed on 'font-size' and 'line-height' (Note: for the purposes of processing the 'font' property in SVG, 'line-height' is assumed to be equal the value for property 'font-size') |
| *Media:* | visual |
| *Animatable:* | yes (non-additive, 'set' and 'animate' elements only) |

Shorthand property for setting 'font-style', 'font-variant', 'font-weight', 'font-size', 'line-height' and 'font-family'. The 'line-height' property has no effect on text layout in SVG. For the purposes of the **'font'** property, 'line-height' is assumed to be equal to the value of the **'font-size'** property. Conforming SVG Viewers are not required to support the various system font options (caption, icon, menu, message-box, small-caption and status-bar) and can use a system font or one of the generic fonts instead.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-font]. The rules for expressing the syntax of CSS property values can be found at [CSS2-propdef].

## 10.11 Spacing properties

Three properties affect the space between characters and words:

- **'kerning'** indicates whether the user agent should adjust inter-glyph spacing based on kerning tables that are included in the relevant font (i.e., enable auto-kerning) or instead disable auto-kerning and instead set inter-character spacing to a specific length (typically, zero).
- **'letter-spacing'** indicates an amount of space that is to be added between text characters supplemental to any spacing due to the **'kerning'** property.
- **'word-spacing'** indicates the spacing behavior between words.

**'kerning'**

| | |
|---|---|
| *Value:* | auto \| <length> \| inherit |
| *Initial:* | auto |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The value of **auto** indicates that the user agent should adjust inter-glyph spacing based on kerning tables that are included in the font that will be used (i.e., enable auto-kerning).

If a **<length>** is provided, then auto-kerning is disabled. Instead, inter-character spacing is set to the given **<length>**. The most common scenario, other than **auto**, is to set **'kerning'** to a value of **0** so that auto-kerning is disabled.

If a **<length>** is provided without a unit identifier (e.g., an unqualified number such as **128**), the SVG user agent processes the **<length>** as a width value in the current user coordinate system.

If a **<length>** is provided with one of the unit identifiers (e.g., **.25em** or **1%**), then the SVG user agent converts the **<length>** into a corresponding value in the current user coordinate system by applying the rules described in Units.

When a **<length>** is provided, its value is added to the inter-character spacing value specified by the **'letter-spacing'** property.

**'letter-spacing'**

| | |
|---|---|
| *Value:* | normal \| <length> \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property specifies spacing behavior between text characters supplemental to any spacing due to the **'kerning'** property.

For SVG, if a **<length>** is provided without a unit identifier (e.g., an unqualified number such as **128**), the SVG user agent processes the **<length>** as a width value in the current user coordinate system.

If a **<length>** is provided with one of the unit identifiers (e.g., **.25em** or **1%**), then the SVG user agent converts the **<length>** into a corresponding value in the current user coordinate system by applying the rules described in Units.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-letter-spacing].

**'word-spacing'**

| | |
|---|---|
| *Value:* | normal \| <length> \| inherit |
| *Initial:* | normal |
| *Applies to:* | text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property specifies spacing behavior between words. For SVG, if a **<length>** is provided without a unit identifier (e.g., an unqualified number such as **128**), the SVG user agent processes the **<length>** as a width value in the current user coordinate system.

If a **<length>** is provided with one of the unit identifiers (e.g., **.25em** or **1%**), then the SVG user agent converts the **<length>** into a corresponding value in the current user coordinate system by applying the rules described in Units.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-word-spacing].

## 10.12 Text decoration

**'text-decoration'**

| | |
|---|---|
| *Value:* | none \| [ underline \|\| overline \|\| line-through \|\| blink ] \| inherit |
| *Initial:* | none |
| *Applies to:* | text content elements |
| *Inherited:* | no (see prose) |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

This property describes decorations that are added to the text of an element. Conforming SVG Viewers are not required to support the **blink** value.

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2-text-decoration]. The rules for expressing the syntax of CSS property values can be found at [CSS2-propdef].

The CSS2 specification [CSS2] defines the behavior of the **'text-decoration'** property using the terminology "block-level elements" and "inline elements". For the purposes of the **'text-decoration'** property and SVG, a **'text'** element represents a block-level element and any of the potential children of a **'text'** element (e.g., a **'tspan'**) represent inline elements.

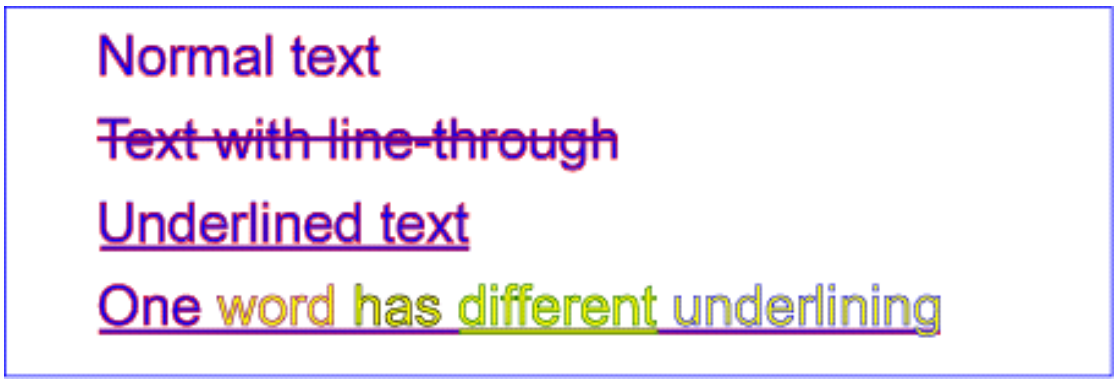
Also, the CSS2 definition of **'text-decoration'** specifies that the "color of the decorations" remain the same on descendant elements. Since SVG offers a painting model consisting of the ability to apply various types of paint (see Painting: Filling, Stroking and Marker Symbols) to both the interior (i.e., the "fill") and the outline (i.e., the "stroke") of text, for SVG the **'text-decoration'** property is defined such that, for an element which has a specified value for the **'text-decoration'** property, all decorations on its content and that of its descendants are rendered using the same fill and stroke properties as are present on the given element. If the **'text-decoration'** property is specified on a descendant, then that overrides the ancestor.

Because SVG allows text to be both filled and stroked, drawing order matters in some circumstances with text decorations. Text decoration drawing order should be as follows:

- All text decorations except line-through should be drawn before the text is filled and stroked; thus, the text is rendered on top of these decorations.
- Line-through should be drawn after the text is filled and stroked; thus, the line-through is rendered on top of the text.

Example textdecoration01 provides examples for **'text-decoration'**. The first line of text has no value for **'text-decoration'**, so the initial value of **'text-decoration:none'** is used. The second line shows **'text-decoration:line-through'**. The third line shows **'text-decoration:underline'**. The fourth line illustrates the rule whereby decorations are rendered using the same fill and stroke properties as are present on the element for which the **'text-decoration'** is specified. Since **'text-decoration'** is specified on the **'text'** element, all text within the **'text'** element has its underline rendered with the same fill and stroke properties as exist on the **'text'** element (i.e., blue fill, red stroke), even though the various words have different fill and stroke property values. However, the word "different" explicitly specifies a value for **'text-decoration'**; thus, its underline is rendered using the fill and stroke properties as the **'tspan'** element that surrounds the word "different" (i.e., yellow fill, darkgreen stroke):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example textdecoration01 - behavior of 'text-decoration' property</desc>
  <rect x="1" y="1" width="1198" height="398" fill="none" stroke="blue" stroke-width="2" />
  <g font-size="60" fill="blue" stroke="red" stroke-width="1" >
    <text x="100" y="75">Normal text</text>
    <text x="100" y="165" text-decoration="line-through" >Text with line-through</text>
    <text x="100" y="255" text-decoration="underline" >Underlined text</text>
    <text x="100" y="345" text-decoration="underline" >
      <tspan>One </tspan>
      <tspan fill="yellow" stroke="purple" >word </tspan>
      <tspan fill="yellow" stroke="black" >has </tspan>
      <tspan fill="yellow" stroke="darkgreen" text-decoration="underline" >different </tspan>
      <tspan fill="yellow" stroke="blue" >underlining</tspan>
    </text>
  </g>
</svg>
```




***Example textdecoration01***

*View this example as SVG (SVG-enabled and CSS-enabled browsers only)*

## 10.13 Text on a path

### 10.13.1 Introduction to text on a path

In addition to text drawn in a straight line, SVG also includes the ability to place text along the shape of a **'path'** element. To specify that a block of text is to be rendered along the shape of a **'path'**, include the given text within a **'textPath'** element which includes an **xlink:href** attribute with a URI reference to a **'path'** element.

### 10.13.2 The 'textPath' element

```
<!ENTITY % textPathExt "" >
<!ELEMENT textPath (#PCDATA|desc|title|metadata|tspan|tref|altGlyph|a|animate|set|animateColor
              %textPathExt;)* >
<!ATTLIST textPath
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %langSpaceAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  startOffset %Length; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED
  method (align|stretch) #IMPLIED
  spacing (auto|exact) #IMPLIED >
```

*Attribute definitions:*

**`startOffset` = "<length>"**
: An offset from the start of the **'path'** for the initial current text position, calculated using the user agent's distance along the path algorithm.
If a <length> other than a percentage is given, then the **startOffset** represents a distance along the path measured in the current user coordinate system.
If a percentage is given, then the **startOffset** represents a percentage distance along the entire path. Thus, **startOffset="0%"** indicates the start point of the **'path'** and **startOffset="100%"** indicates the end point of the **'path'**.
A negative value is an error (see Error processing).
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`method` = "align | stretch"**
: Indicates the method by which text should be rendered along the path.

A value of **align** indicates that the glyphs should be rendered using simple 2x3 transformations such that there is no stretching/warping of the glyphs. Typically, supplemental rotation, scaling and translation transformations are done for each glyph to be rendered. As a result, with **align**, fonts where the glyphs are designed to be connected (e.g., cursive fonts), the connections may not align properly when text is rendered along a path.
A value of **stretch** indicates that the glyph outlines will be converted into paths, and then all end points and control points will be adjusted to be along the perpendicular vectors from the path, thereby stretching and possibly warping the glyphs. With this approach, connected glyphs, such as in cursive scripts, will maintain their connections.
If the attribute is not specified, the effect is as if a value of **align** were specified.
*Animatable: yes.*

**`spacing` = "auto | exact"**

Indicates how the user agent should determine the spacing between glyphs that are to be rendered along a path.
A value of **exact** indicates that the glyphs should be rendered exactly according to the spacing rules as specified in Text on a path layout rules.
A value of **auto** indicates that the user agent should use text-on-a-path layout algorithms to adjust the spacing between glyphs in order to achieve visually appealing results.
If the attribute is not specified, the effect is as if a value of **exact** were specified.
*Animatable: yes.*

**`xlink:href` = "<uri>"**

A URI reference to the **'path'** element onto which the glyphs will be rendered. If <uri> is an invalid reference (e.g., no such element exists, or the referenced element is not a **'path'**), then the **'textPath'** element is in error and its entire contents shall not be rendered by the user agent.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %xlinkRefAttrs;, %langSpaceAttrs;, %testAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements;, %graphicsElementEvents;, textLength, lengthAdjust.**

The path data coordinates within the referenced **'path'** element are assumed to be in the same coordinate system as the current **'text'** element, not in the coordinate system where the **'path'** element is defined. The **transform** attribute on the referenced **'path'** element represents a supplemental transformation relative to the current user coordinate system for the current **'text'** element, including any adjustments to the current user coordinate system due to a possible **transform** attribute on the current **'text'** element. For example, the following fragment of SVG content:

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <g transform="translate(25,25)">
    <defs>
      <path id="path1" transform="scale(2)" d="..." fill="none" stroke="red"/>
    </defs>
  </g>
  <text transform="rotate(45)">
    <textPath xlink:href="#path1">Text along path1</textPath>
  </text>
</svg>
```

should have the same effect as the following:

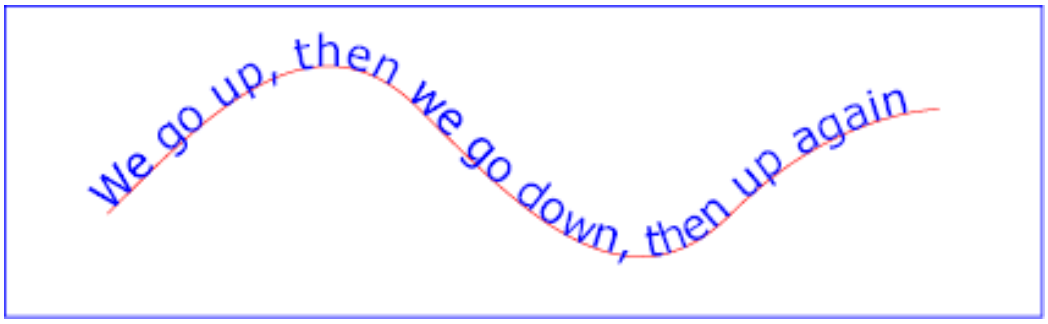
```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <g transform="rotate(45)">
```

```
    <defs>
      <path id="path1" transform="scale(2)" d="..." fill="none" stroke="red"/>
    </defs>
    <text>
      <textPath xlink:href="#path1">Text along path1</textPath>
    </text>
  </g>
</svg>
```

Note that the **`transform="translate(25,25)"`** has no effect on the **'textPath'** element, whereas the **`transform="rotate(45)"`** applies to both the **'text'** and the use of the **'path'** element as the referenced shape for text on a path.

Example toap01 provides a simple example of text on a path:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <path id="MyPath"
          d="M 100 200
             C 200 100 300   0 400 100
             C 500 200 600 300 700 200
             C 800 100 900 100 900 100" />
  </defs>
  <desc>Example toap01 - simple text on a path</desc>

  <use xlink:href="#MyPath" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="42.5" fill="blue" >
    <textPath xlink:href="#MyPath">
      We go up, then we go down, then up again
    </textPath>
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```



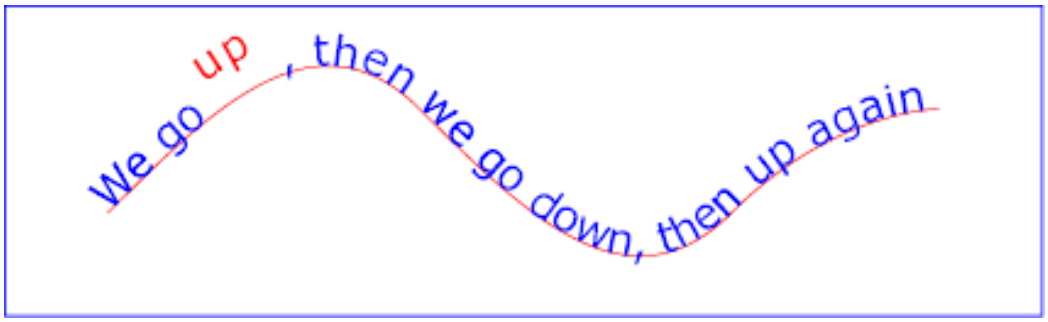

***Example toap01***

*View this example as SVG (SVG-enabled browsers only)*

Example toap02 shows how **'tspan'** elements can be included within **'textPath'** elements to adjust styling attributes and adjust the current text position before rendering a particular glyph. The first occurrence of the

word "up" is filled with the color red. Attribute **dy** is used to lift the word "up" from the baseline.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <path id="MyPath"
          d="M 100 200
             C 200 100 300   0 400 100
             C 500 200 600 300 700 200
             C 800 100 900 100 900 100" />
  </defs>
  <desc>Example toap02 - tspan within textPath</desc>

  <use xlink:href="#MyPath" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="42.5" fill="blue" >
    <textPath xlink:href="#MyPath">
      We go
      <tspan dy="-30" fill="red" >
        up
      </tspan>
      <tspan dy="30">
        ,
      </tspan>
      then we go down, then up again
    </textPath>
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```



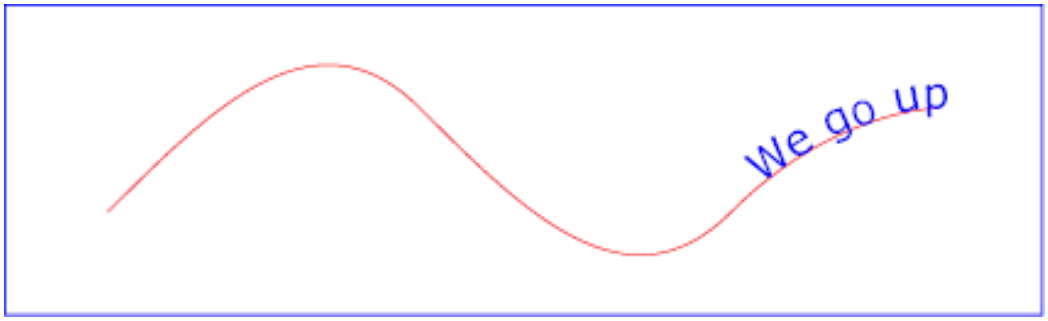

***Example toap02***

*View this example as SVG (SVG-enabled browsers only)*

Example toap03 demonstrates the use of the **startOffset** attribute on the **'textPath'** element to specify the start position of the text string as a particular position along the path. Notice that glyphs that fall off the end of the path are not rendered (see text on a path layout rules).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
```

```
    <path id="MyPath"
          d="M 100 200
             C 200 100 300   0 400 100
             C 500 200 600 300 700 200
             C 800 100 900 100 900 100" />
  </defs>
  <desc>Example toap03 - text on a path with startOffset attribute</desc>

  <use xlink:href="#MyPath" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="42.5" fill="blue" >
    <textPath xlink:href="#MyPath" startOffset="80%">
      We go up, then we go down, then up again
    </textPath>
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```




***Example toap03***

*View this example as SVG (SVG-enabled browsers only)*

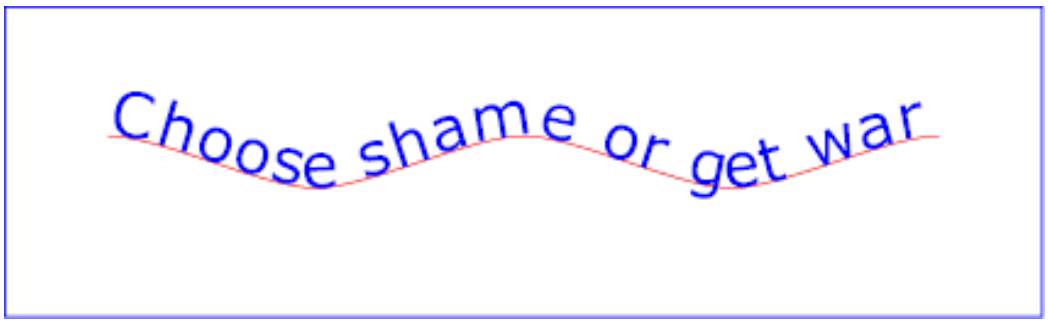
## 10.13.3 Text on a path layout rules

Conceptually, for text on a path the target path is stretched out into either a horizontal or vertical straight line segment. For horizontal text layout flows, the path is stretched out into a hypothetical horizontal line segment such that the start of the path is mapped to the left of the line segment. For vertical text layout flows, the path is stretched out into a hypothetical vertical line segment such that the start of the path is mapped to the top of the line segment. The standard text layout rules are applied to the hypothetical straight line segment and the result is mapped back onto the target path. Vertical and bidirectional text layout rules also apply to text on a path.

The reference orientation is determined individually for each glyph that is rendered along the path. For horizontal text layout flows, the reference orientation for a given glyph is the vector that starts at the intersection point on the path to which the glyph is attached and which points in the direction 90 degrees counter-clockwise from the angle of the curve at the intersection point. For vertical text layout flows, the reference orientation for a given glyph is the vector that starts at the intersection point on the path to which the glyph is attached and which points in the direction 180 degrees from the angle of the curve at the intersection point.

Example toap04 will be used to illustrate the particular layout rules for text on a path that supplement the basic text layout rules for straight line horizontal or vertical text.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <path id="MyPath"
          d="M 100 125
             C 150 125 250 175 300 175
             C 350 175 450 125 500 125
             C 550 125 650 175 700 175
             C 750 175 850 125 900 125" />
  </defs>
  <desc>Example toap04 - text on a path layout rules</desc>

  <use xlink:href="#MyPath" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="60" fill="blue" letter-spacing="2" >
    <textPath xlink:href="#MyPath">
      Choose shame or get war
    </textPath>
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
```



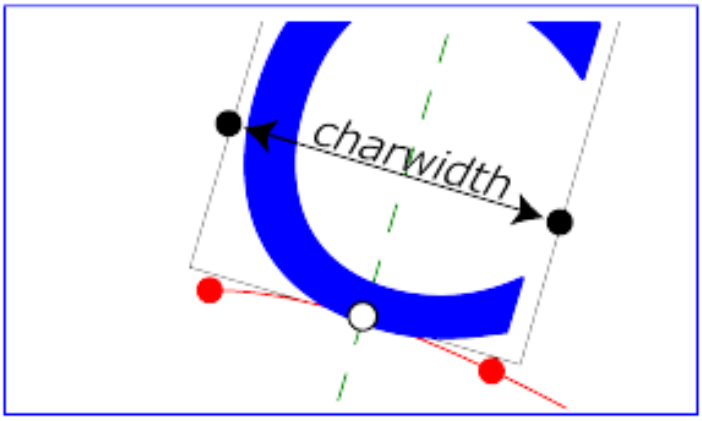

***Example toap04***

*View this example as SVG (SVG-enabled browsers only)*

The following picture does an initial zoom in on the first glyph in the **'text'** element.

The small dot above shows the point at which the glyph is attached to the path. The box around the glyph shows the glyph is rotated such that its horizontal axis is parallel to the tangent of the curve at the point at which the glyph is attached to the path. The box also shows the glyph's **charwidth** (i.e., the amount which the current text position advances horizontally when the glyph is drawn using horizontal text layout).

The next picture zooms in further to demonstrate the detailed layout rules.




For left-to-right horizontal text layout along a path (i.e., when the glyph orientation is perpendicular to the inline-progression-direction), the layout rules are as follows:

- Determine the **startpoint-on-the-path** for the first glyph using attribute **startOffset** and property **'text-anchor'**. For **'text-anchor:start'**, startpoint-on-the-path is the point on the path which represents the point on the path which is **startOffset** distance along the path from the start of the path, calculated using the user agent's distance along the path algorithm. For **'text-anchor:middle'**, startpoint-on-the-path is the point on the path which represents the point on the path which is [ **startOffset** minus half of the total advance values for all of the glyphs in the **'textPath'** element ] distance along the path from the start of the path, calculated using the user agent's distance along the path algorithm. For **'text-anchor:end'**, startpoint-on-the-path is the point on the path which represents the point on the path which is [ **startOffset** minus the total advance values for all of the glyphs in the **'textPath'** element ]. Before rendering the first glyph, the horizontal component of the startpoint-on-the-path is adjusted to take into account various horizontal alignment text properties and attributes, such as a **dx** attribute value on a **'tspan'** element. (In the picture above, the startpoint-on-the-path is the leftmost dot on the path.)
- Determine the glyph's charwidth (i.e., the amount which the current text position advances horizontally when the glyph is drawn using horizontal text layout). (In the picture above, the charwidth is the distance between the two dots at the side of the box.)
- Determine the point on the curve which is charwidth distance along the path from the startpoint-on-the-path for this glyph, calculated using the user agent's distance along the path algorithm. This point is the **endpoint-on-the-path** for the glyph. (In the picture above, the endpoint-on-the-path for the glyph is the rightmost dot on the path.)
- Determine the **midpoint-on-the-path**, which is the point on the path which is "halfway" (user agents can choose either a distance calculation or a parametric calculation) between the startpoint-on-the-path and the endpoint-on-the-path. (In the picture above, the midpoint-on-the-path is shown as a white dot.)
- Determine the **glyph-midline**, which is the vertical line in the glyph's coordinate system that goes through the glyph's x-axis midpoint. (In the picture above, the glyph-midline is shown as a dashed line.)
- Position the glyph such that the glyph-midline passes through the midpoint-on-the-path and is perpendicular to the line through the startpoint-on-the-path and the endpoint-on-the-path.
- Align the glyph vertically relative to the midpoint-on-the-path based on property **'alignment-baseline'** and any specified values for attribute **dy** on a **'tspan'** element. In the example above, the **'alignment-baseline'** property is unspecified, so the initial value of **'alignment-baseline:baseline'** will be used. There are no **'tspan'** elements; thus, the baseline of the glyph is aligned to the

midpoint-on-the-path.

- For each subsequent glyph, set a new startpoint-on-the-path as the previous endpoint-on-the-path, but with appropriate adjustments taking into account horizontal kerning tables in the font and current values of various attributes and properties, including spacing properties and **'tspan'** elements with values provided for attributes **dx** and **dy**. All adjustments are calculated as distance adjustments along the path, calculated using the user agent's distance along the path algorithm.
- Glyphs whose midpoint-on-the-path are off either end of the path are not rendered.
- Continue rendering glyphs until there are no more glyphs.

Comparable rules are used for top-to-bottom vertical text layout along a path (i.e., when the glyph orientation is parallel with the inline-progression-direction), the layout rules are as follows:

- Determine the startpoint-on-the-path using the same method as for horizontal text layout along a path, except that before rendering the first glyph, the horizontal component of the startpoint-on-the-path is adjusted to take into account various vertical alignment text properties and attributes, such as a **dy** attribute value on a **'tspan'** element.
- Determine the glyph's charheight (i.e., the amount which the current text position advances vertically when the glyph is drawn using vertical text layout).
- Determine the point on the curve which is charheight distance along the path from the startpoint-on-the-path for this glyph, calculated using the user agent's distance along the path algorithm. This point is the endpoint-on-the-path for the glyph.
- Determine the midpoint-on-the-path, which is the point on the path which is "halfway" (user agents can choose either a distance calculation or a parametric calculation) between the startpoint-on-the-path and the endpoint-on-the-path.
- Determine the glyph-midline, which is the horizontal line in the glyph's coordinate system that goes through the glyph's y-axis midpoint.
- Position the glyph such that the glyph-midline passes through the midpoint-on-the-path and is perpendicular to the line through the startpoint-on-the-path and the endpoint-on-the-path.
- Align the glyph horizontally (where horizontal is relative to the glyph's coordinate system) relative to the midpoint-on-the-path based on property **'alignment-baseline'** and any specified values for attribute **dx** on a **'tspan'** element.
- For each subsequent glyph, set a new startpoint-on-the-path as the previous endpoint-on-the-path, but with appropriate adjustments taking into account vertical kerning tables in the font and current values of various attributes and properties, including spacing properties and **'tspan'** elements with values provided for attributes **dx** and **dy**. All adjustments are calculated as distance adjustments along the path, calculated using the user agent's distance along the path algorithm.
- Glyphs whose midpoint-on-the-path are off either end of the path are not rendered.
- Continue rendering glyphs until there are no more glyphs.

In the calculations above, if either the startpoint-on-the-path or the endpoint-on-the-path is off the end of the path, then extend the path beyond its end points with a straight line that is parallel to the tangent at the path at its end point so that the midpoint-on-the-path can still be calculated.

When the inline-progression-direction is horizontal, then any **x** attributes on **'text'**, **'tspan'**, **'tref'** or **'altGlyph'** elements represent new absolute offsets along the path, thus providing explicit new values for startpoint-on-the-path. Any **y** attributes on **'text'**, **'tspan'**, **'tref'** or **'altGlyph'** elements are ignored. When the inline-progression-direction is horizontal, then any **y** attributes on **'text'**, **'tspan'**, **'tref'** or **'altGlyph'** elements represent new absolute offsets along the path, thus providing explicit new values for startpoint-on-the-path. Any **x** attributes on **'text'**, **'tspan'**, **'tref'** or **'altGlyph'** elements are ignored.

## 10.14 Alternate glyphs

There are situations such as ligatures, special-purpose fonts (e.g., a font for music symbols) or alternate glyphs for Asian text strings where it is required that a different set of glyphs is used than the glyph(s) which normally corresponds to the given character data.

The **'altGlyph'** element provides control over the glyphs used to render particular character data.

```
<!ENTITY % altGlyphExt "" >
<!ELEMENT altGlyph (#PCDATA %altGlyphExt;)* >
<!ATTLIST altGlyph
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  glyphRef CDATA #IMPLIED
  format CDATA #IMPLIED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**

A URI reference either to a **'glyph'** element in an SVG document fragment or to an **'altGlyphDef'** element.

If the reference is to a **'glyph'** element and that glyph is available, then that glyph is rendered instead of the character(s) that are inside of the **'altGlyph'** element.

If the reference is to an **'altGlyphDef'** element, then if an appropriate set of alternate glyphs is located from processing the **'altGlyphDef'** element, then those alternate glyphs are rendered instead of the character(s) that are inside of the **'altGlyph'** element.

*Animatable: no.*

**`glyphRef` = "<string>"**

The glyph identifier, the format of which is dependent on the **format** of the given font. (Same meaning as the **glyphRef** attribute on the **'glyphRef'** element.)

*Animatable: no.*

**`format` = "<string>"**

The format of the given font. If the font is in one of the formats listed in the [CSS2-src] specification (e.g., *TrueDoc™ Portable Font Resource* or *Embedded OpenType*), then the <string> must contain the corresponding font format string defined in [CSS2-src] (e.g., *truedoc-pfr* or *embedded-*

*opentype*). (Same meaning as the **format** attribute on the **'glyphRef'** element.)
*Animatable: no.*

**x = "<coordinate>+"**
The <coordinate> values are processed in the same manner as the **x** attribute on the **'tspan'** element, with the following exception: If the referenced alternate glyphs are rendered instead of the Unicode characters inside the **'altGlyph'** element, then any absolute X coordinates specified via an **x** attribute on this element or any ancestor **'text'** or **'tspan'** elements for Unicode characters 2 through <n> within the **'altGlyph'** element are ignored. Any absolute X coordinate specified via an **x** attribute on this element or any ancestor **'text'** or **'tspan'** elements for the first Unicode character within the **'altGlyph'** element sets a new absolute X coordinate for the current text position before rendering the first alternate glyph.
*Animatable: yes.*

**y = "<coordinate>+"**
The corresponding absolute Y coordinates for rendering the **'altGlyph'** element.
*Animatable: yes.*

**dx = "<length>+"**
The <length> values are processed in the same manner as the **dx** attribute on the **'tspan'** element, with the following exception: If the referenced alternate glyphs are rendered instead of the Unicode characters inside the **'altGlyph'** element, then any relative X coordinates specified via an **dx** attribute on this element or any ancestor **'text'** or **'tspan'** elements for Unicode characters 2 through <n> within the **'altGlyph'** element are ignored. Any relative X coordinate specified via an **dx** attribute on this element or any ancestor **'text'** or **'tspan'** elements for the first Unicode character within the **'altGlyph'** element sets a new relative X coordinate for the current text position before rendering the first alternate glyph.
*Animatable: yes.*

**dy = "<length>+"**
The corresponding relative Y coordinates for rendering the **'altGlyph'** element.
*Animatable: yes.*

**rotate = "<number>+"**
The <number> values are processed in the same manner as the **rotate** attribute on the **'tspan'** element, with the following exception: If the referenced alternate glyphs are rendered instead of the Unicode characters inside the **'altGlyph'** element, then any supplemental rotation values specified via an **rotate** attribute on this element or any ancestor **'text'** or **'tspan'** elements for Unicode characters 2 through <n> within the **'altGlyph'** element are ignored. Supplemental rotation values specified via an **rotate** attribute on this element or any ancestor **'text'** or **'tspan'** elements for the first Unicode character within the **'altGlyph'** element sets a new supplemental rotation angle before rendering the alternate glyphs.
*Animatable: yes (non-additive, 'set' and 'animate' elements only).*

***Attributes defined elsewhere:***
**%stdAttrs;, %xlinkRefAttrs;, %testAttrs;, %langSpaceAttrs;, externalResourcesRequired, class,, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements; %graphicsElementEvents;.**

If the references to alternate glyphs do not result in successful identification of alternate glyphs to use, then the character(s) that are inside of the **'altGlyph'** element are rendered as if the **'altGlyph'** element were a **'tspan'** element instead.

An **'altGlyph'** element either references a **'glyph'** element or an **'altGlyphDef'** element via its **xlink:href** attribute or identifies a glyph by means of font selection properties, a glyph identifier and a font format. If the

**xlink:href** attribute is specified, it takes precedence, and the other glyph identification attributes and properties are ignored.

The **'altGlyphDef'** element defines a set of possible glyph substitutions.

```
<!ENTITY % altGlyphDefExt "" >
<!ELEMENT altGlyphDef ((glyphRef+|altGlyphItem+) %altGlyphDefExt;) >
<!ATTLIST altGlyphDef
  %stdAttrs; >
```

***Attributes defined elsewhere:***

**%stdAttrs;.**

An **'altGlyphDef'** can contain either of the following:

- In the simplest case, an **'altGlyphDef'** contains one or more **'glyphRef'** elements. Each **'glyphRef'** element references a single glyph within a particular font. If all of the referenced glyphs are available, then these glyphs are rendered instead of the character(s) inside of the referencing **'altGlyph'** element. If any of the referenced glyphs are unavailable, then the character(s) that are inside of the **'altGlyph'** element are rendered as if there were not an **'altGlyph'** element surrounding those characters.
- In the more complex case, an **'altGlyphDef'** contains one or more **'altGlyphItem'** elements. Each **'altGlyphItem'** represents a candidate set of substitute glyphs. Each **'altGlyphItem'** contains one or more **'glyphRef'** elements. Each **'glyphRef'** element references a single glyph within a particular font. The first **'altGlyphItem'** in which all referenced glyphs are available is chosen. The glyphs referenced from this **'altGlyphItem'** are rendered instead of the character(s) that are inside of the referencing **'altGlyph'** element. If none of the **'altGlyphItem'** elements result in a successful match (i.e., none of the **'altGlyphItem'** elements has all of its referenced glyphs available), then the character(s) that are inside of the **'altGlyph'** element are rendered as if there were not an **'altGlyph'** element surrounding those characters.

The **'altGlyphItem'** element defines a candidate set of possible glyph substitutions. The first **'altGlyphItem'** element whose referenced glyphs are all available is chosen. Its glyphs are rendered instead of the character(s) that are inside of the referencing **'altGlyph'** element

```
<!ENTITY % altGlyphItemExt "" >
<!ELEMENT altGlyphItem (glyphRef+ %altGlyphItemExt;) >
<!ATTLIST altGlyphItem
  %stdAttrs; >
```

***Attributes defined elsewhere:***

**%stdAttrs;.**

The **'glyphRef'** element defines a possible glyph to use.

```
<!ELEMENT glyphRef EMPTY >
<!ATTLIST glyphRef
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-FontSpecification;
  glyphRef CDATA #IMPLIED
  format CDATA #IMPLIED
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  dx %Number; #IMPLIED
  dy %Number; #IMPLIED >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**
: A URI reference to a **'glyph'** element in an SVG document fragment. The referenced **'glyph'** is rendered as an alternate glyph.
*Animatable: no.*

**`glyphRef` = "<string>"**
: The glyph identifier, the format of which is dependent on the **format** of the given font.
*Animatable: no.*

**`format` = "<string>"**
: The format of the given font. If the font is in one of the formats listed in [CSS2-src] (e.g., *TrueDoc™ Portable Font Resource* or *Embedded OpenType*), then the <string> must contain the corresponding font format string defined in [CSS2-src] (e.g., *truedoc-pfr* or *embedded-opentype*).
*Animatable: no.*

**x = "<number>"**
: This value represents the new absolute X coordinate within the font's coordinate system for this glyph.
The font coordinate system is based on the *em square* model described in the "Fonts" chapter of the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
If the attribute is not specified, for the first **'glyphRef'** child element, the effect is as if the attribute were set to "0", whereas for subsequent **'glyphRef'** child elements, the effect is as if the attribute were set to the end X coordinate from the previous **'glyphRef'** element.
*Animatable: no.*

**y = "<number>"**
: The corresponding new absolute Y coordinate within the font's coordinate system for this glyph.
*Animatable: no.*

**dx = "<number>"**
: This value represents the relative X coordinate within the font's coordinate system for this glyph. The glyph is thus shifted by **<number>** units along the positive X axis within the font's coordinate system supplemental to the absolute X coordinate established by the **x** attribute (either due to an explicit **x** attribute or due to default value processing for the **x** attribute).
The font coordinate system is based on the *em square* model described in the "Fonts" chapter of the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
If the attribute is not specified, the effect is as if the attribute were set to "0".
*Animatable: no.*

**dy = "<number>"**
: The corresponding number of units within the font's coordinate system to shift the glyph along the

positive Y axis relative to the absolute Y coordinate established by the **y** attribute.
*Animatable: no.*

***Attributes defined elsewhere:***
**%stdAttrs;, %xlinkRefAttrs;, class, style, %PresentationAttributes-FontSpecification;.**

A **'glyphRef'** either references a **'glyph'** element in an SVG document fragment via its **xlink:href** attribute or identifies a glyph by means of font selection properties, a glyph identifier and a font format. If insufficient attributes and properties have been specified to identify a glyph, then the **'glyphRef'** is processed in the same manner as when a glyph reference is fully specified, but the given glyph is not available. If the **xlink:href** attribute is specified, it takes precedence, and the other glyph identification attributes and properties are ignored.

## 10.15 White space handling

SVG supports the standard XML attribute **xml:space** to specify the handling of white space characters within a given **'text'** element's character data. The SVG user agent has special processing rules associated with this attribute as described below. These are behaviors that occur subsequent to XML parsing [XML10] and any construction of a Document Object Model [DOM2].

**xml:space** is an inheritable attribute which can have one of two values:

- **default** (the initial/default value for xml:space) - When `xml:space="default"`, the SVG user agent will do the following using a copy of the original character data content. First, it will remove all newline characters. Then it will convert all tab characters into space characters. Then, it will strip off all leading and trailing space characters. Then, all contiguous space characters will be consolidated.
- **preserve** - When `xml:space="preserve"`, the SVG user agent will do the following using a copy of the original character data content. It will convert all newline and tab characters into space characters. Then, it will draw all space characters, including leading, trailing and multiple contiguous space characters. Thus, when drawn with `xml:space="preserve"`, the string `"a  b"` (three spaces between "a" and "b") will produce a larger separation between "a" and "b" than `"a b"` (one space between "a" and "b").

The following example illustrates that line indentation can be important when using `xml:space="default"`. The fragment below show two pairs of similar **'text'** elements, with both **'text'** elements using **xml:space='default'**. For these examples, there is no extra white space at the end of any of the lines (i.e., the line break occurs immediately after the last visible character).

```
[01]  <text xml:space='default'>
[02]    WS example
[03]    indented lines
[04]  </text>
[05]  <text xml:space='preserve'>WS example indented lines</text>
[06]
[07]  <text xml:space='default'>
[08]WS example
[09]non-indented lines
[10]  </text>
[11]  <text xml:space='preserve'>WS examplenon-indented lines</text>
```

The first pair of **'text'** elements above show the effect of indented character data. The attribute **xml:space='default'** in the first **'text'** element instructs the user agent to:

- convert all tabs (if any) to space characters,
- strip out all line breaks (i.e., strip out the line breaks at the end of lines [01], [02] and [03]),
- strip out all leading space characters (i.e., strip out space characters before "WS example" on line [02]),
- strip out all trailing space characters (i.e., strip out space characters before "</text>" on line [04]),
- consolidate all intermediate space characters (i.e., the space characters before "indented lines" on line [03]) into a single space character.

The second pair of **'text'** elements above show the effect of non-indented character data. The attribute **xml:space='default'** in the third **'text'** element instructs the user agent to:

- convert all tabs (if any) to space characters,
- strip out all line breaks (i.e., strip out the line breaks at the end of lines [07], [08] and [09]),
- strip out all leading space characters (there are no leading space characters in this example),
- strip out all trailing space characters (i.e., strip out space characters before "</text>" on line [10]),
- consolidate all intermediate space characters into a single space character (in this example, there are no intermediate space characters).

Note that XML parsers are required to convert the standard representations for a newline indicator (e.g., the literal two-character sequence "#xD#xA" or the stand-alone literals #xD or #xA) into the single character #xA before passing character data to the application. Thus, each newline in SVG will be represented by the single character #xA, no matter what representation for newlines might have been used in the original resource. (See XML end-of-line handling.)

Any features in the SVG language or the SVG DOM that are based on character position number, such as the **x**, **y**, **dx**, **dy** and **rotate** attributes on the **'text'**, **'tspan'**, **'tref'** **'altGlyph'** elements, are based on character position after applying the white space handling rules described here. In particular, if `xml:space="default"`, it is often the case that white space characters are removed as part of processing. Character position numbers index into the text string after the white space characters have been removed per the rules in this section.

The **xml:space** attribute is:

*Animatable: no.*

## 10.16 Text selection and clipboard operations

Conforming SVG viewers on systems which have the capacity for text selection (e.g., systems which are equipped with a pointer device such as a mouse) and which have system clipboards for copy/paste operations are required to support:

- user selection of text strings in SVG content
- the ability to copy selected text strings to the system clipboard

A text selection operation starts when all of the following occur:

- the user positions the pointing device over a glyph that has been rendered as part of a **'text'** element, initiates a *select* operation (e.g., pressing the standard system mouse button for select operations) and then moves the pointing device while continuing the *select* operation (e.g., continuing to press the standard system mouse button for select operations)

- no other visible graphics element has been painted above the glyph at the point at which the pointing device was clicked
- no links or events have been assigned to the **'text'** , **'tspan'** or **'textPath'** , element(s) (or their ancestors) associated with the given glyph.

As the text selection operation proceeds (e.g., the user continues to press the given mouse button), all associated events with other graphics elements are ignored (i.e., the text selection operation is modal) and the SVG user agent shall dynamically indicate which characters are selected by an appropriate highlighting technique, such as redrawing the selected glyphs with inverse colors. As the pointer is moved during the text selection process, the end glyph for the text selection operation is the glyph within the same **'text'** element whose glyph cell is closest to the pointer. All characters within the **'text'** element whose position within the **'text'** element is between the start of selection and end of selection shall be highlighted, regardless of position on the canvas and regardless of any graphics elements that might be above the end of selection point.

Once the text selection operation ends (e.g., the user releases the given mouse button), the selected text will stay highlighted until an event occurs which cancels text selection, such as a pointer device activation event (e.g., pressing a mouse button).

Detailed rules for determining which characters to highlight during a text selection operation are provided in Text selection implementation notes.

For systems which have system clipboards, the SVG user agent is required to provide a user interface for initiating a copy of the currently selected text to the system clipboard. It is sufficient for the SVG user agent to post the selected text string in the system's appropriate clipboard format for plain text, but it is preferable if the SVG user agent also posts a rich text alternative which captures the various font properties associated with the given text string.

For bidirectional text, the user agent must support text selection in logical order, which will result in discontinuous highlighting of glyphs due to the bidirectional reordering of characters. User agents can provide an alternative ability to select bidirectional text in visual rendering order (i.e., after bidirectional text layout algorithms have been applied), with the result that selected character data might be discontinuous logically. In this case, if the user requests that bidirectional text be copied to the clipboard, then the user agent is required to make appropriate adjustments to copy only the visually selected characters to the clipboard.

When feasible, it is recommended that generators of SVG attempt to order their text strings to facilitate properly ordered text selection within SVG viewing applications such as Web browsers.

## 10.17 DOM interfaces

The following interfaces are defined below: SVGTextContentElement, SVGTextPositioningElement, SVGTextElement, SVGTSpanElement, SVGTRefElement, SVGTextPathElement, SVGAltGlyphElement, SVGAltGlyphDefElement, SVGAltGlyphItemElement, SVGGlyphRefElement.

### Interface SVGTextContentElement

The **SVGTextContentElement** interface is inherited by various text-related interfaces, such as

SVGTextElement, SVGTSpanElement, SVGTRefElement, SVGAltGlyphElement and SVGTextPathElement.

### IDL Definition

```
interface SVGTextContentElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                events::EventTarget {

  // lengthAdjust Types
  const unsigned short LENGTHADJUST_UNKNOWN   = 0;
  const unsigned short LENGTHADJUST_SPACING     = 1;
  const unsigned short LENGTHADJUST_SPACINGANDGLYPHS     = 2;

  readonly attribute SVGAnimatedLength      textLength;
  readonly attribute SVGAnimatedEnumeration lengthAdjust;

  long     getNumberOfChars (  );
  float    getComputedTextLength (  );
  float    getSubStringLength ( in unsigned long charnum, in unsigned long nchars )
                raises( DOMException );
  SVGPoint getStartPositionOfChar ( in unsigned long charnum )
                raises( DOMException );
  SVGPoint getEndPositionOfChar ( in unsigned long charnum )
                raises( DOMException );
  SVGRect  getExtentOfChar ( in unsigned long charnum )
                raises( DOMException );
  float    getRotationOfChar ( in unsigned long charnum )
                raises( DOMException );
  long     getCharNumAtPosition ( in SVGPoint point );
  void     selectSubString ( in unsigned long charnum, in unsigned long nchars )
                raises( DOMException );
};
```

### Definition group lengthAdjust Types

#### Defined constants

| | |
|---|---|
| LENGTHADJUST_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| LENGTHADJUST_SPACING | Corresponds to value **spacing**. |
| LENGTHADJUST_SPACINGANDGLYPHS | Corresponds to value **spacingAndGlyphs**. |

### Attributes

**readonly SVGAnimatedLength textLength**

Corresponds to attribute **textLength** on the given element.

**readonly SVGAnimatedEnumeration lengthAdjust**

Corresponds to attribute **lengthAdjust** on the given element. The value must be one of the length adjust constants specified above.

### Methods

**getNumberOfChars**

Returns the total number of characters to be rendered within the current element. Includes characters which are included via a **'tref'** reference.

**No Parameters**

**Return value**

long Total number of characters.

**No Exceptions**

**getComputedTextLength**

The total sum of all of the advance values from rendering all of the characters within this element, including the advance value on the glyphs (horizontal or vertical), the effect of properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **dx** and **dy** on **'tspan'** elements. For non-rendering environments, the user agent shall make reasonable assumptions about glyph metrics.

**No Parameters**

**Return value**

float The text advance distance.

**No Exceptions**

**getSubStringLength**

The total sum of all of the advance values from rendering the specified substring of the characters, including the advance value on the glyphs (horizontal or vertical), the effect of properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **dx** and **dy** on **'tspan'** elements. For non-rendering environments, the user agent shall make reasonable assumptions about glyph metrics.

**Parameters**

| | |
|---|---|
| in unsigned long charnum | The index of the first character in the substring, where the first character has an index of 0. |
| in unsigned long nchars | The number of characters in the substring. |

**Return value**

float The text advance distance.

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum+nchars is greater than or equal to the number of characters at this node. |

**getStartPositionOfChar**

Returns the current text position before rendering the character in the user coordinate system for rendering the glyph(s) that correspond to the specified character. The current text position has already taken into account the effects of any inter-character adjustments due to properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **x**, **y**, **dx** and **dy**. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs), then each of the inseparable characters will return the start position for the first glyph.

**Parameters**

| | |
|---|---|
| in unsigned long charnum | The index of the character, where the first character has an index of 0. |

**Return value**

SVGPoint The character's start position.

**Exceptions**

| | |
|---|---|
| DOMException | INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum is greater than or equal to the number of characters at this node. |

**getEndPositionOfChar**

Returns the current text position after rendering the character in the user coordinate system for rendering the glyph(s) that correspond to the specified character. This current text position does *not* take into account the effects of any inter-character adjustments to prepare for the next character, such as properties **'kerning'**, **'letter-spacing'** and **'word-spacing'** and adjustments due to attributes **x**, **y**, **dx** and **dy**. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs), then each of the inseparable characters will return the end position for the last glyph.

**Parameters**

in unsigned long charnum — The index of the character, where the first character has an index of 0.

**Return value**

SVGPoint — The character's end position.

**Exceptions**

DOMException — INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum is greater than or equal to the number of characters at this node.

**getExtentOfChar**

Returns a tightest rectangle which defines the minimum and maximum X and Y values in the user coordinate system for rendering the glyph(s) that correspond to the specified character. The calculations assume that all glyphs occupy the full standard glyph cell for the font. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs), then each of the inseparable characters will return the same extent.

**Parameters**

in unsigned long charnum — The index of the character, where the first character has an index of 0.

**Return value**

SVGRect — The rectangle which encloses all of the rendered glyph(s).

**Exceptions**

DOMException — INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum is greater than or equal to the number of characters at this node.

**getRotationOfChar**

Returns the rotation value relative to the current user coordinate system used to render the glyph(s) corresponding to the specified character. If multiple glyph(s) are used to render the given character and the glyphs each have different rotations (e.g., due to text-on-a-path), the user agent shall return an average value (e.g., the rotation angle at the midpoint along the path for all glyphs used to render this character). The rotation value represents the rotation that is supplemental to any rotation due to properties **'glyph-orientation-horizontal'** and **'glyph-orientation-vertical'**; thus, any glyph rotations due to these properties are *not* included into the returned rotation value. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs), then each of the inseparable characters will return the same rotation value.

**Parameters**

in unsigned long charnum — The index of the character, where the first character has an index of 0.

**Return value**

float — The rotation angle.

**Exceptions**

DOMException — INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum is greater than or equal to the number of characters at this node.

**getCharNumAtPosition**

Returns the index of the character whose corresponding glyph cell bounding box contains the specified point. The calculations assume that all glyphs occupy the full standard glyph cell for the font. If no such character exists, a value of -1 is returned. If multiple such characters exist, the character within the element whose glyphs were rendered last (i.e., take into account any reordering such as for bidirectional text) is used. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs), then the user agent shall allocate an equal percentage of the text advance amount to each of the contributing characters in determining which of the characters is chosen.

**Parameters**

in SVGPoint point  A point in user space.

**Return value**

long  The index of the character which is at the given point, where the first character has an index of 0.

**No Exceptions**

**selectSubString**

Causes the specified substring to be selected just as if the user selected the substring interactively.

**Parameters**

in unsigned long charnum  The index of the start character which is at the given point, where the first character has an index of 0.

in unsigned long nchars  The number of characters in the substring. If nchars specifies more characters than are available, then the substring will consist of all characters starting with charnum until the end of the list of characters.

**No Return Value**

**Exceptions**

DOMException  INDEX_SIZE_ERR: Raised if the charnum is negative or if charnum is greater than or equal to the number of characters at this node.

## Interface SVGTextPositioningElement

The **SVGTextPositioningElement** interface is inherited by text-related interfaces: SVGTextElement, SVGTSpanElement, SVGTRefElement and SVGAltGlyphElement.

**IDL Definition**

```
interface SVGTextPositioningElement : SVGTextContentElement {
  readonly attribute SVGAnimatedLengthList x;
  readonly attribute SVGAnimatedLengthList y;
  readonly attribute SVGAnimatedLengthList dx;
  readonly attribute SVGAnimatedLengthList dy;
  readonly attribute SVGAnimatedNumberList rotate;
};
```

**Attributes**

**readonly SVGAnimatedLengthList x**
Corresponds to attribute **x** on the given element.

**readonly SVGAnimatedLengthList y**
Corresponds to attribute **y** on the given element.

**readonly SVGAnimatedLengthList dx**
Corresponds to attribute **dx** on the given element.

**readonly SVGAnimatedLengthList dy**
Corresponds to attribute **dy** on the given element.

**readonly SVGAnimatedNumberList rotate**
Corresponds to attribute **rotate** on the given element.

## Interface SVGTextElement

The **SVGTextElement** interface corresponds to the **'text'** element.

**IDL Definition**

```
interface SVGTextElement :
               SVGTextPositioningElement,
               SVGTransformable {};
```

## Interface SVGTSpanElement

The **SVGTSpanElement** interface corresponds to the **'tspan'** element.

**IDL Definition**

```
interface SVGTSpanElement : SVGTextPositioningElement {};
```

## Interface SVGTRefElement

The **SVGTRefElement** interface corresponds to the **'tref'** element.

**IDL Definition**

```
interface SVGTRefElement :
               SVGTextPositioningElement,
               SVGURIReference {};
```

## Interface SVGTextPathElement

The **SVGTextPathElement** interface corresponds to the **'textPath'** element.

### IDL Definition

```
interface SVGTextPathElement :
               SVGTextContentElement,
               SVGURIReference {

  // textPath Method Types
  const unsigned short TEXTPATH_METHODTYPE_UNKNOWN   = 0;
  const unsigned short TEXTPATH_METHODTYPE_ALIGN     = 1;
  const unsigned short TEXTPATH_METHODTYPE_STRETCH     = 2;
  // textPath Spacing Types
  const unsigned short TEXTPATH_SPACINGTYPE_UNKNOWN   = 0;
  const unsigned short TEXTPATH_SPACINGTYPE_AUTO     = 1;
  const unsigned short TEXTPATH_SPACINGTYPE_EXACT     = 2;

  readonly attribute SVGAnimatedLength             startOffset;
  readonly attribute SVGAnimatedEnumeration method;
  readonly attribute SVGAnimatedEnumeration spacing;
};
```

**Definition group textPath Method Types**

**Defined constants**

| | |
|---|---|
| TEXTPATH_METHODTYPE_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| TEXTPATH_METHODTYPE_ALIGN | Corresponds to value **align**. |
| TEXTPATH_METHODTYPE_STRETCH | Corresponds to value **stretch**. |

**Definition group textPath Spacing Types**

**Defined constants**

| | |
|---|---|
| TEXTPATH_SPACINGTYPE_UNKNOWN | The enumeration was set to a value that is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| TEXTPATH_SPACINGTYPE_AUTO | Corresponds to value **auto**. |
| TEXTPATH_SPACINGTYPE_EXACT | Corresponds to value **exact**. |

**Attributes**

**readonly SVGAnimatedLength startOffset**

Corresponds to attribute **startOffset** on the given **'textPath'** element.

**readonly SVGAnimatedEnumeration method**

Corresponds to attribute **method** on the given **'textPath'** element. The value must be one of the method type constants specified above.

**readonly SVGAnimatedEnumeration spacing**

Corresponds to attribute **spacing** on the given **'textPath'** element. The value must be one of the spacing type constants specified above.

## Interface SVGAltGlyphElement

The **SVGAltGlyphElement** interface corresponds to the **'altGlyph'** element.

**IDL Definition**

```
interface SVGAltGlyphElement :
               SVGTextPositioningElement,
               SVGURIReference {

          attribute DOMString glyphRef;
                      // raises DOMException on setting
          attribute DOMString format;
                      // raises DOMException on setting
};
```

**Attributes**

**DOMString glyphRef**

Corresponds to attribute **glyphRef** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**DOMString format**

Corresponds to attribute **format** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

## Interface SVGAltGlyphDefElement

The **SVGAltGlyphDefElement** interface corresponds to the **'altGlyphDef'** element.

**IDL Definition**

```
interface SVGAltGlyphDefElement : SVGElement {};
```

## Interface SVGAltGlyphItemElement

The **SVGAltGlyphItemElement** interface corresponds to the **'altGlyphItem'** element.

**IDL Definition**

```
interface SVGAltGlyphItemElement : SVGElement {};
```

## Interface SVGGlyphRefElement

The **SVGGlyphRefElement** interface corresponds to the **'glyphRef'** element.

### IDL Definition

```
interface SVGGlyphRefElement :
               SVGElement,
               SVGURIReference,
               SVGStylable {

          attribute DOMString glyphRef;
                      // raises DOMException on setting
          attribute DOMString format;
                      // raises DOMException on setting
          attribute float    x;
                      // raises DOMException on setting
          attribute float    y;
                      // raises DOMException on setting
          attribute float    dx;
                      // raises DOMException on setting
          attribute float    dy;
                      // raises DOMException on setting
};
```

### Attributes

**DOMString glyphRef**

Corresponds to attribute **glyphRef** on the given element.

**Exceptions on setting**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString format**

Corresponds to attribute **format** on the given element.

**Exceptions on setting**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float x**

Corresponds to attribute **x** on the given element.

**Exceptions on setting**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float y**

Corresponds to attribute **y** on the given element.

**Exceptions on setting**

DOMException — NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**float dx**

Corresponds to attribute **dx** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

**float dy**

Corresponds to attribute **dy** on the given element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

---

04 September 2001

# 11 Painting: Filling, Stroking and Marker Symbols

## Contents



## 11.1 Introduction

**'path'** elements, **'text'** elements and basic shapes can be **filled** (which means painting the interior of the object) and **stroked** (which means painting along the outline of the object). Filling and stroking both can be thought of in more general terms as **painting** operations.

Certain elements (i.e., **'path'**, **'polyline'**, **'polygon'** and **'line'** elements) can also have marker symbols drawn at their vertices.

With SVG, you can paint (i.e., fill or stroke) with:

- a single color

- a gradient (linear or radial)
- a pattern (vector or image, possibly tiled)
- custom paints available via extensibility

SVG uses the general notion of a **paint server**. Paint servers are specified using a URI reference on a **'fill'** or **'stroke'** property. Gradients and patterns are just specific types of paint servers.

## 11.2 Specifying paint

Properties **'fill'** and **'stroke'** take on a value of type **<paint>**, which is specified as follows:

**<paint>**: **none |**
**currentColor |**
**<color> [icc-color(<name>[,<icccolorvalue>]*)] |**
**<uri> [ none | currentColor | <color> [icc-color(<name>[,<icccolorvalue>]*)]] |**
**inherit**

**none**
Indicates that no paint is applied.

**currentColor**
Indicates that painting is done using the color specified by the **'color'** property. This mechanism is provided to facilitate sharing of color attributes between parent grammars such as other (non-SVG) XML. This mechanism allows you to define a style in your HTML which sets the 'color' property and then pass that style to the SVG user agent so that your SVG text will draw in the same color.

**<color> [icc-color(<name>[,<icccolorvalue>]*)]**
<color> is the explicit color (in the sRGB [SRGB] color space) to be used to paint the current object. SVG supports all of the syntax alternatives for <color> defined in [CSS2-color-types], with the exception that SVG contains an expanded list of recognized color keywords names. If an optional ICC color specification is provided, then the user agent searches the color profile description database for a color profile description entry whose name descriptor matches <name> and uses the last matching entry that is found. (If no match is found, then the ICC color specification is ignored.) The comma-separated list (with optional white space) of **<icccolorvalue>**'s is a set of ICC-profile-specific color values, expressed as <number>s. (In most cases, the **<icccolorvalue>**'s will be in the range 0-to-1.) On platforms which support ICC-based color management, the **icc-color** gets precedence over the <color> (which is in the sRGB color space). Note that color interpolation occurs in an RGB color space even if an ICC-based color specification is provided (see **'color-interpolation'** and **'color-interpolation-filters'**). Percentages are not allowed on **<icccolorvalue>**'s. For more on ICC-based colors, refer to Color profile descriptions.

**<uri>**
**[ none |**
**currentColor |**
**<color> [icc-color(<name>[,<icccolorvalue>]*)]**
The <uri> is how you identify a paint server such as a gradient, a pattern or a custom paint defined by an extension (see Extensibility). The <uri> provides the ID of the paint server (e.g., a gradient or a pattern) to be used to paint the current object. If the URI reference is not valid (e.g., it points to an object that doesn't exist or the object is not a valid paint server), then the paint method following the <uri> (i.e., **none | currentColor | <color> [icc-color(<name>[,<icccolorvalue>]*)]|**

**inherit**) is used if provided; otherwise, the document is in error (see Error processing).

## 11.3 Fill Properties

**'fill'**

| | |
|---|---|
| *Value:* | <paint> (See Specifying paint) |
| *Initial:* | black |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'fill'** property paints the interior of the given graphical element. The area to be painted consists of any areas inside the outline of the shape. To determine the inside of the shape, all subpaths are considered, and the interior is determined according to the rules associated with the current value of the **'fill-rule'** property. The zero-width geometric outline of a shape is included in the area to be painted.

The fill operation fills open subpaths by performing the fill operation as if an additional "closepath" command were added to the path to connect the last point of the subpath with the first point of the subpath. Thus, fill operations apply to both open subpaths within **'path'** elements (i.e., subpaths without a closepath command) and **'polyline'** elements.

.

**'fill-rule'**

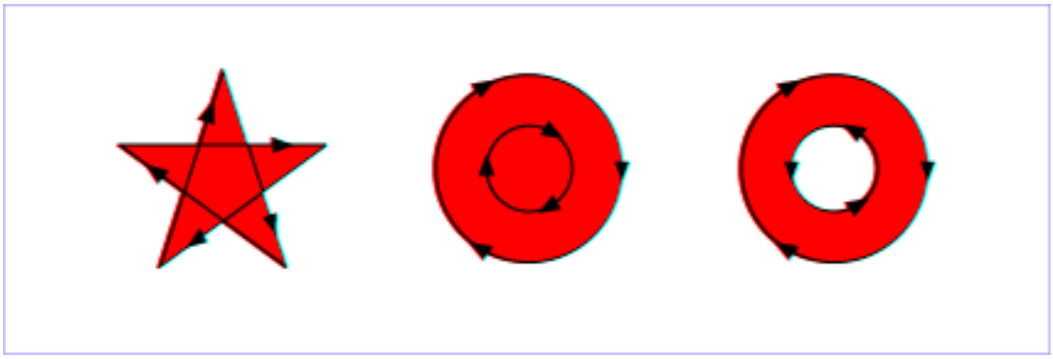
| | |
|---|---|
| *Value:* | nonzero \| evenodd \| inherit |
| *Initial:* | nonzero |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'fill-rule'** property indicates the algorithm which is to be used to determine what parts of the canvas are included inside the shape. For a simple, non-intersecting path, it is intuitively clear what region lies "inside"; however, for a more complex path, such as a path that intersects itself or where one subpath encloses another, the interpretation of "inside" is not so obvious.

The **'fill-rule'** property provides two options for how the inside of a shape is determined:

**nonzero**

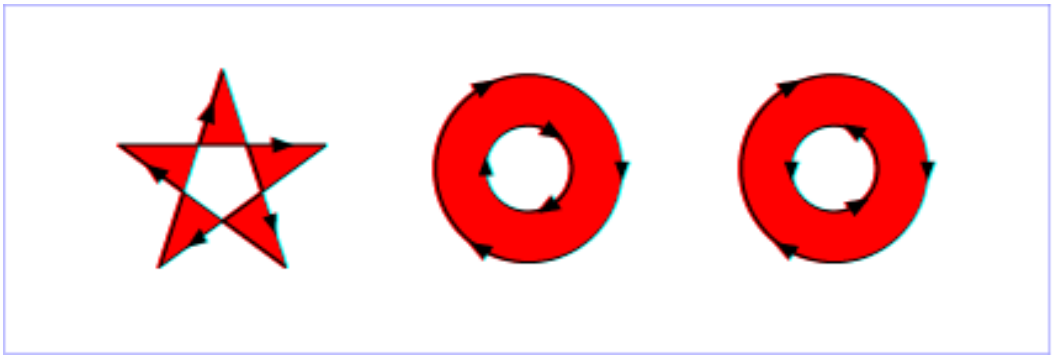
This rule determines the "insideness" of a point on the canvas by drawing a ray from that point to infinity in any direction and then examining the places where a segment of the shape crosses the ray. Starting with a count of zero, add one each time a path segment crosses the ray from left to right and subtract one each time a path segment crosses the ray from right to left. After counting the crossings, if the result is zero then the point is *outside* the path. Otherwise, it is *inside*. The following drawing illustrates the **nonzero** rule:



*View this example as SVG (SVG-enabled browsers only)*

**evenodd**

This rule determines the "insideness" of a point on the canvas by drawing a ray from that point to infinity in any direction and counting the number of path segments from the given shape that the ray crosses. If this number is odd, the point is inside; if even, the point is outside. The following drawing illustrates the **evenodd** rule:



*View this example as SVG (SVG-enabled browsers only)*

(Note: the above explanations do not specify what to do if a path segment coincides with or is tangent to the ray. Since any ray will do, one may simply choose a different ray that does not have such problem intersections.)

**'fill-opacity'**

| | |
|---|---|
| *Value:* | <opacity-value> \| inherit |
| *Initial:* | 1 |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**'fill-opacity'** specifies the opacity of the painting operation used to paint the interior the current object. (See Painting shapes and text.)

**<opacity-value>**

The opacity of the painting operation used to fill the current object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. (See Clamping values which are restricted to a particular range.)

Related properties: **'stroke-opacity'** and **'opacity'**.

## 11.4 Stroke Properties

The following are the properties which affect how an element is stroked.

In all cases, all stroking properties which are affected by directionality, such as those having to do with dash patterns, must be rendered such that the stroke operation starts at the same point at which the graphics element starts. In particular, for **'path'** elements, the start of the path is the first point of the initial "moveto" command.

For stroking properties such as dash patterns whose computations are dependent on progress along the outline of the graphics element, distance calculations are required to utilize the SVG user agent's standard Distance along a path algorithms.

When stroking is performed using a complex paint server, such as a gradient or a pattern, the stroke operation must be identical to the result that would have occurred if the geometric shape defined by the geometry of the current graphics element and its associated stroking properties were converted to an equivalent **'path'** element and then filled using the given paint server.

**'stroke'**

| | |
|---|---|
| *Value:* | <paint> (See Specifying paint) |
| *Initial:* | none |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'stroke'** property paints along the outline of the given graphical element.

A subpath (see Paths) consisting of a single moveto is not stroked. A subpath consisting of a moveto and lineto to the same exact location or a subpath consisting of a moveto and a closepath will be stroked only if the **'stroke-linecap'** property is set to "round", producing a circle centered at the given point.

**'stroke-width'**

| | |
|---|---|
| *Value:* | <length> \| inherit |
| *Initial:* | 1 |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | Yes |
| *Media:* | visual |

*Animatable:* yes

**<length>**

The width of the stroke on the current object. If a percentage is used, the value represents a percentage of the current viewport. (See Units.)

A zero value causes no stroke to be painted. A negative value is an error (see Error processing).

**'stroke-linecap'**

| | |
|---|---|
| *Value:* | butt \| round \| square \| inherit |
| *Initial:* | butt |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**'stroke-linecap'** specifies the shape to be used at the end of open subpaths when they are stroked.

**butt**

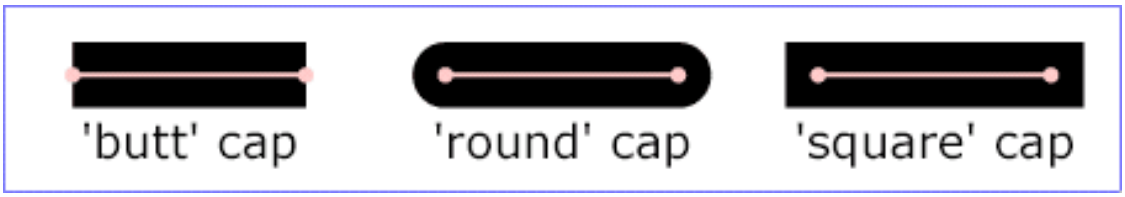
See drawing below.

**round**

See drawing below.

**square**

See drawing below.




*View this example as SVG (SVG- and CSS-enabled browsers only)*

**'stroke-linejoin'**

| | |
|---|---|
| *Value:* | miter \| round \| bevel \| inherit |
| *Initial:* | miter |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**'stroke-linejoin'** specifies the shape to be used at the corners of paths or basic shapes when they are stroked.

**miter**
: See drawing below.

**round**
: See drawing below.

**bevel**
: See drawing below.


*View this example as SVG (SVG- and CSS-enabled browsers only)*

**'stroke-miterlimit'**

| | |
|---|---|
| *Value:* | <miterlimit> \| inherit |
| *Initial:* | 4 |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

When two line segments meet at a sharp angle and **miter** joins have been specified for **'stroke-linejoin'**, it is possible for the miter to extend far beyond the thickness of the line stroking the path. The **'stroke-miterlimit'** imposes a limit on the ratio of the miter length to the **'stroke-linewidth'**.

**<miterlimit>**
: The limit on the ratio of the miter length to the **'stroke-linewidth'**. The value of **<miterlimit>** must be a number greater than or equal to 1. Any other value is an error (see Error processing).

**'stroke-dasharray'**

| | |
|---|---|
| *Value:* | none \| <dasharray> \| inherit |
| *Initial:* | none |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | yes (see below) |
| *Media:* | visual |
| *Animatable:* | yes (non-additive) |

**'stroke-dasharray'** controls the pattern of dashes and gaps used to stroke paths. **<dasharray>** contains a list of comma-separated (with optional white space) <length>s that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, **stroke-dasharray: 5,3,2** is equivalent to **stroke-dasharray: 5,3,2,5,3,2**.

**none**
: Indicates that no dashing is used. If stroked, the line is drawn solid.

**<dasharray>**
: A list of comma-separated <length>'s (with optional white space), each of which can have a unit identifier , including specification of a percentage. A percentage represents a distance as a percentage of the current viewport. (See Units.) A negative <length> value is an error (see Error processing). If the sum of the <length>'s is zero, then the stroke is rendered as if a value of **none** were specified.

**'stroke-dashoffset'**

| | |
|---|---|
| *Value:* | <length> \| inherit |
| *Initial:* | 0 |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | see prose |
| *Media:* | visual |
| *Animatable:* | yes |

**'stroke-dashoffset'** specifies the distance into the dash pattern to start the dash.

**<length>**
: If a percentage is used, the value represents a percentage of the current viewport (See Units.)
Values can be negative.

**'stroke-opacity'**

| | |
|---|---|
| *Value:* | <opacity-value> \| inherit |
| *Initial:* | 1 |
| *Applies to:* | shapes and text content elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**'stroke-opacity'** specifies the opacity of the painting operation used to stroke the current object. (See Painting shapes and text.)

**<opacity-value>**
: The opacity of the painting operation used to stroke the current object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. (See Clamping values which are restricted to a particular range.)

Related properties: **'fill-opacity'** and **'opacity'**.

## 11.5 Controlling visibility

SVG uses two properties, **'display'** and **'visibility'**, to control the visibility of graphical elements or (in the case of the **'display'** property) container elements.

The differences between the two properties are as follows:

- When applied to a container element, setting **'display'** to **none** causes the container and all of its children to be invisible; thus, it acts on groups of elements as a group. **'visibility'**, however, only applies to individual graphics elements. Setting **'visibility'** to **hidden** on a **'g'** will make its children invisible as long as the children do not specify their own **'visibility'** properties as **visible**. Note that **'visibility'** is *not* an inheritable property.
- When the **'display'** property is set to **none**, then the given element does not become part of the rendering tree. With **'visibility'** set to **hidden**, however, processing occurs as if the element were part of the rendering tree and still taking up space, but not actually rendered onto the canvas. This distinction has implications for the **'tspan'**, **'tref'** and **'altGlyph'** elements, event processing, for bounding box calculations and for calculation of clipping paths. If **'display'** is set to **none** on a **'tspan'**, **'tref'** or **'altGlyph'** element, then the text string is ignored for the purposes of text layout; however, if **'visibility'** is set to **hidden**, the text string is used for text layout (i.e., it takes up space) even though it is not rendered on the canvas. Regarding events, if **'display'** is set to **none**, the element receives no events; however, if **'visibility'** is set to **hidden**, the element might still receive events, depending on the value of property **'pointer-events'**. The geometry of a graphics element with **'display'** set to **none** is not included in bounding box and clipping paths calculations; however, even if **'visibility'** is to **hidden**, the geometry of the graphics element still contributes to bounding box and clipping path calculations.

**'display'**

| | |
|---|---|
| *Value:* | inline \| block \| list-item \| run-in \| compact \| marker \| table \| inline-table \| table-row-group \| table-header-group \| table-footer-group \| table-row \| table-column-group \| table-column \| table-cell \| table-caption \| none \| inherit |
| *Initial:* | inline |
| *Applies to:* | **'svg'**, **'g'**, **'switch'**, **'a'**, **'foreignObject'**, graphics elements (including the **'text'** element) and text sub-elements (i.e., **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'**) |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | all |
| *Animatable:* | yes |

A value of **display: none** indicates that the given element and its children shall not be rendered directly (i.e., those elements are not present in the rendering tree). Any value other than **none** or **inherit** indicates that the given element shall be rendered by the SVG user agent.

The **'display'** property only affects the direct rendering of a given element, whereas it does not prevent elements from being referenced by other elements. For example, setting **display: none** on a **'path'** element

will prevent that element from getting rendered directly onto the canvas, but the **'path'** element can still be referenced by a **'textPath'** element; furthermore, its geometry will be used in text-on-a-path processing even if the **'path'** has **display: none**.

The **'display'** property affects direct rendering into offscreen canvases also, such as occurs with the implementation model for masks. Thus, setting **display: none** on a child of a **'mask'** will prevent the given child element from being rendered as part of the mask. Similarly, setting **display: none** on a child of a **'clipPath'** element will prevent the given child element from contributing to the clipping path.

Elements with **display: none** do not take up space in text layout operations, do not receive events, and do not contribute to bounding box and clipping paths calculations.

Except for any additional information provided in this specification, the normative definition is the CSS2 definition of the 'display' property.

**'visibility'**

| | |
|---|---|
| *Value:* | visible \| hidden \| collapse \| inherit |
| *Initial:* | visible |
| *Applies to:* | graphics elements (including the **'text'** element) and text sub-elements (i.e., **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** and **'a'**) |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**visible**
: The current graphics element is visible.

**hidden or collapse**
: The current graphics element is invisible (i.e., nothing is painted on the canvas).

Note that if the **'visibility'** property is set to **hidden** on a **'tspan'**, **'tref'** or **'altGlyph'** element, then the text is invisible but still takes up space in text layout calculations.

Depending on the value of property **'pointer-events'**, graphics elements which have their **'visibility'** property set to **hidden** still might receive events.

Except for any additional information provided in this specification, the normative definition is the CSS2 definition of the 'visibility' property.

## 11.6 Markers

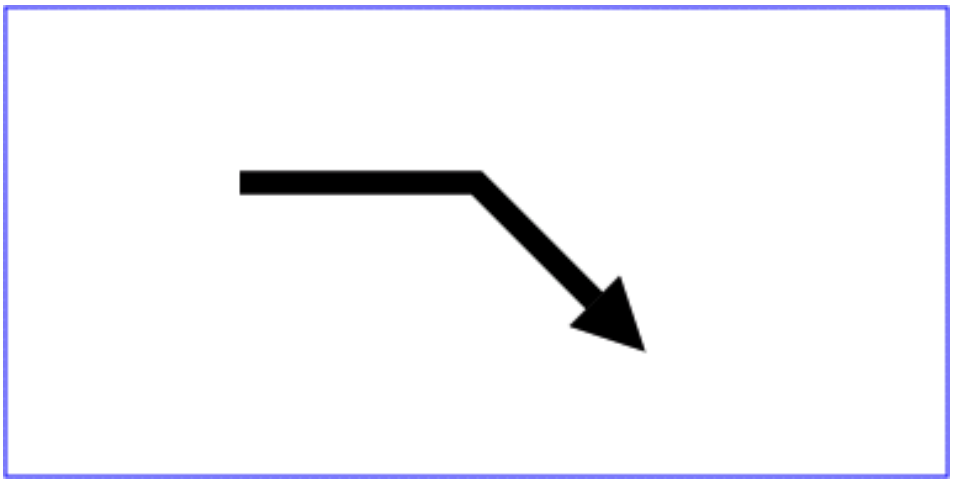
### 11.6.1 Introduction

A marker is a symbol which is attached to one or more vertices of **'path'**, **'line'**, **'polyline'** and **'polygon'** elements. Typically, markers are used to make arrowheads or polymarkers. Arrowheads can be defined by attaching a marker to the start or end vertices of **'path'**, **'line'** or **'polyline'** elements. Polymarkers can be defined by attaching a marker to all vertices of a **'path'**, **'line'**, **'polyline'** or **'polygon'** element.

The graphics for a marker are defined by a **'marker'** element. To indicate that a particular **'marker'** element should be rendered at the vertices of a particular **'path'**, **'line'**, **'polyline'** or **'polygon'** element, set one or more marker properties (**'marker'**, **'marker-start'**, **'marker-mid'** or **'marker-end'**) to reference the given **'marker'** element.

Example Marker draws a triangular marker symbol as an arrowhead at the end of a path.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="4" markerHeight="3"
      orient="auto">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>
  <rect x="10" y="10" width="3980" height="1980"
       fill="none" stroke="blue" stroke-width="10" />
  <desc>Placing an arrowhead at the end of a path.
  </desc>
  <path d="M 1000 750 L 2000 750 L 2500 1250"
        fill="none" stroke="black" stroke-width="100"
        marker-end="url(#Triangle)"  />
</svg>
```



***Example Marker***

*View this example as SVG (SVG-enabled browsers only)*

Markers can be animated. The animated effects will show on all current uses of the markers within the document.

## 11.6.2 The 'marker' element

The **'marker'** element defines the graphics that is to be used for drawing arrowheads or polymarkers on a given **'path'**, **'line'**, **'polyline'** or **'polygon'** element.

```
<!ENTITY % markerExt "" >
<!ELEMENT marker (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%markerExt;)* >
<!ATTLIST marker
  %stdAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  refX %Coordinate; #IMPLIED
  refY %Coordinate; #IMPLIED
  markerUnits (strokeWidth | userSpaceOnUse) #IMPLIED
  markerWidth  %Length; #IMPLIED
  markerHeight %Length; #IMPLIED
  orient CDATA #IMPLIED >
```

*Attribute definitions:*

**`markerUnits` = "strokeWidth | userSpaceOnUse"**

Defines the coordinate system for attributes **markerWidth**, **markerHeight** and the contents of the **'marker'**.

If **markerUnits="strokeWidth"**, **markerWidth**, **markerHeight** and the contents of the **'marker'** represent values in a coordinate system which has a single unit equal the size in user units of the current stroke width (see the **'stroke-width'** property) in place for the graphic object referencing the marker.

If **markerUnits="userSpaceOnUse"**, **markerWidth**, **markerHeight** and the contents of the **'marker'** represent values in the current user coordinate system in place for the graphic object referencing the marker (i.e., the user coordinate system for the element referencing the **'marker'** element via a **'marker'**, **'marker-start'**, **'marker-mid'** or **'marker-end'** property).

If attribute **markerUnits** is not specified, then the effect is as if a value of **strokeWidth** were specified.

*Animatable: yes.*

**`refX` = "<coordinate>"**

The x-axis coordinate of the reference point which is to be aligned exactly at the marker position. The coordinate is defined in the coordinate system after application of the **viewBox** and **preserveAspectRatio** attributes.

If the attribute is not specified, the effect is as if a value of "0" were specified.

*Animatable: yes.*

**`refY` = "<coordinate>"**

The y-axis coordinate of the reference point which is to be aligned exactly at the marker position. The coordinate is defined in the coordinate system after application of the **viewBox** and **preserveAspectRatio** attributes.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`markerWidth` = "<length>"**

Represents the width of the viewport into which the marker is to be fitted when it is rendered. A negative value is an error (see Error processing). A value of zero disables rendering of the element.
If the attribute is not specified, the effect is as if a value of "3" were specified.
*Animatable: yes.*

**`markerHeight` = "<length>"**

Represents the height of the viewport into which the marker is to be fitted when it is rendered. A negative value is an error (see Error processing). A value of zero disables rendering of the element.
If the attribute is not specified, the effect is as if a value of "3" were specified.
*Animatable: yes.*

**`orient` = "auto | <angle>"**

Indicates how the marker is rotated.

A value of *auto* indicates that the marker is oriented such that its positive x-axis is pointing as follows: (a) if there is a path segment coming into the vertex and another path segment going out of the vertex, the marker's positive x-axis should point toward the angle bisector for the angle at the given vertex, where that angle has one side consisting of tangent vector for the path segment going into the vertex and the other side the tangent vector for the path segment going out of the vertex (note: if the tangent vectors are the same, the angle bisector equals the two tangent vectors), (b) if there is only a path segment going into the vertex (e.g., the last vertex on an open path), the marker's positive x-axis should point in the same direction as the tangent vector for the path segment going into the vertex, (c) if there is only a path segment going out of the vertex (e.g., the first vertex on an open path), the marker's positive x-axis should point in the same direction as the tangent vector for the path segment going out of the vertex. (Refer to 'path' element implementation notes for a more thorough discussion of the directionality of path segments.)

In all cases for closed subpaths (e.g., subpaths which end with a 'closepath' command), the orientation of the marker corresponding to the initial point of the subpath is calculated assuming that:

- the path segment going into the vertex is the path segment corresponding to the closepath
- the path segment coming out of the vertex is the first path segment in the subpath

When a 'closepath' command is followed by a command other than a 'moveto' command, then the orientation of the marker corresponding to the 'closepath' command is calculated assuming that:

- the path segment going into the vertex is the path segment corresponding to the closepath
- the path segment coming out of the vertex is the first path segment of the subsequent subpath

A value of *<angle>* represents a particular orientation in the user space of the graphic object referencing the marker. For example, if a value of "0" is given, then the marker will be drawn such that its x-axis will align with the x-axis of the user space of the graphic object referencing the marker. If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes (non-additive, 'set' and 'animate' elements only).*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, externalResourcesRequired, viewBox, preserveAspectRatio, style, %PresentationAttributes-All;.**

Markers are drawn such that their reference point (i.e., attributes **refX** and **refY**) is positioned at the given vertex. In other words, a translation transformation is constructed by the user agent to achieve the effect of having point (**refX** and **refY**) within the marker content's coordinate system (after any transformations due to the **viewBox** and **preserveAspectRatio** attributes) align exactly with the given vertex.

SVG's user agent style sheet sets the **'overflow'** property for **'marker'** elements to **hidden**, which causes a rectangular clipping path to be created at the bounds of the marker tile. Unless the **'overflow'** property is overridden, any graphics within the marker which goes outside of the marker rectangle will be clipped.

The contents of the **'marker'** are relative to a new coordinate system. Attribute **markerUnits** determines an initial scale factor for transforming the graphics in the marker into the user coordinate system for the referencing element. An additional set of transformations might occur if there is a **viewBox** attribute, in which case the coordinate system for the contents of the **'marker'** will be transformed due to the processing of attributes **viewBox** and **preserveAspectRatio**. If there is no **viewBox** attribute, then the assumed default value for the the **viewBox** attribute has the origin of the viewBox coincident with the origin of the viewport and the width/height of the viewBox the same as the width/height of the viewport.

Properties inherit into the **'marker'** element from its ancestors; properties do *not* inherit from the element referencing the **'marker'** element.

**'marker'** elements are never rendered directly; their only usage is as something that can be referenced using the **'marker'**, **'marker-start'**, **'marker-end'** and **'marker-mid'** properties. The **'display'** property does not apply to the **'marker'** element; thus, **'marker'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'marker'** elements are available for referencing even when the **'display'** property on the **'marker'** element or any of its ancestors is set to **none**.

Event attributes and event listeners attached to the contents of a **'marker'** element are not processed; only the rendering aspects of **'marker'** elements are processed.

## 11.6.3 Marker properties

**'marker-start'** defines the arrowhead or polymarker that shall be drawn at the first vertex of the given **'path'** element or basic shape. **'marker-end'** defines the arrowhead or polymarker that shall be drawn at the final vertex. **'marker-mid'** defines the arrowhead or polymarker that shall be drawn at every other vertex (i.e., every vertex except the first and last). Note that for a **'path'** element which ends with a closed sub-path, the last vertex is the same as the initial vertex on the given sub-path. In this case, if **'marker-end'** does not equal **none**, then it is possible that two markers will be rendered on the given vertex. One way to prevent this is to set **'marker-end'** to **none**. (Note that the same comment applies to **'polygon'** elements.)

**'marker-start', 'marker-end', marker-mid'**

*Value:* none | inherit | <uri>

*Initial:* none
*Applies to:* **'path'**, **'line'**, **'polyline'** and **'polygon'** elements
*Inherited:* yes
*Percentages:* N/A
*Media:* visual
*Animatable:* yes

**none**
Indicates that no marker symbol shall be drawn at the given vertex (vertices).

**<uri>**
The <uri> is a URI reference to the **'marker'** element which shall be used as the arrowhead symbol or polymarker at the given vertex or vertices. If the URI reference is not valid (e.g., it points to an object that is undefined or the object is not a **'marker'** element), then the marker(s) shall not be drawn.

The **'marker'** property specifies the marker symbol that shall be used for all points on the sets the value for all vertices on the given **'path'** element or basic shape. It is a short-hand for the three individual marker properties:

**'marker'**
*Value:* see individual properties
*Initial:* see individual properties
*Applies to:* **'path'**, **'line'**, **'polyline'** and **'polygon'** elements
*Inherited:* yes
*Percentages:* N/A
*Media:* visual
*Animatable:* yes

## 11.6.4 Details on how markers are rendered

Markers are drawn after the given object is filled and stroked.

For each marker that is drawn, a temporary new user coordinate system is established so that the marker will be positioned and sized correctly, as follows:

- The axes of the temporary new user coordinate system are aligned according to the **orient** attribute on the **'marker'** element and the slope of the curve at the given vertex. (Note: if there is a discontinuity at a vertex, the slope is the average of the slopes of the two segments of the curve that join at the given vertex. If a slope cannot be determined, the slope is assumed to be zero.)
- A temporary new coordinate system is established by attribute **markerUnits**. If **markerUnits** equals **strokeWidth**, then the temporary new user coordinate system is the result of scaling the current user coordinate system by the current value of property **'stroke-width'**. If **markerUnits** equals **userSpaceOnUse**, then no extra scale transformation is applied.
- An additional set of transformations might occur if the **'marker'** element includes a **viewBox** attribute, in which case additional transformations are set up to produce the necessary result due to attributes **viewBox** and **preserveAspectRatio**.
- If the **'overflow'** property on the **'marker'** element indicates that the marker needs to be clipped to its viewport, then an implicit clipping path is established at the bounds of the viewport.

The rendering effect of a marker is as if the contents of the referenced **'marker'** element were deeply

cloned into a separate non-exposed DOM tree for each instance of the marker. Because the cloned DOM tree is non-exposed, the SVG DOM does not show the cloned instance of the marker.

For user agents that support Styling with CSS, the conceptual deep cloning of the referenced **'marker'** element into a non-exposed DOM tree also copies any property values resulting from the CSS cascade [CSS2-CASCADE] and property inheritance on the referenced element and its contents. CSS2 selectors can be applied to the original (i.e., referenced) elements because they are part of the formal document structure. CSS2 selectors cannot be applied to the (conceptually) cloned DOM tree because its contents are not part of the formal document structure.

For illustrative purposes, we'll repeat the marker example shown earlier:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="4" markerHeight="3"
      orient="auto">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>
  <rect x="10" y="10" width="3980" height="1980"
       fill="none" stroke="blue" stroke-width="10" />
  <desc>Placing an arrowhead at the end of a path.
  </desc>
  <path d="M 1000 750 L 2000 750 L 2500 1250"
        fill="none" stroke="black" stroke-width="100"
        marker-end="url(#Triangle)"  />
</svg>
```

The rendering effect of the above file will be visually identical to the following:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000"
     xmlns="http://www.w3.org/2000/svg">

  <desc>File which produces the same effect
      as the marker example file, but without
      using markers.
  </desc>
  <rect x="10" y="10" width="3980" height="1980"
       fill="none" stroke="blue" stroke-width="10" />
  <!-- The path draws as before, but without the marker properties -->
  <path d="M 1000 750 L 2000 750 L 2500 1250"
        fill="none" stroke="black" stroke-width="100"  />

  <!-- The following logic simulates drawing a marker
       at final vertex of the path. -->

  <!-- First off, move the origin of the user coordinate system
       so that the origin is now aligned with the end point of the path. -->
  <g transform="translate(2500,1250)" >

    <!-- Rotate the coordinate system 45 degrees because
         the marker specified orient="auto" and the final segment
```

```
      of the path is going in the direction of 45 degrees. -->
    <g transform="rotate(45)" >

      <!-- Scale the coordinate system to match the coordinate system
           indicated by the 'markerUnits' attributes, which in this case has
           a value of 'strokeWidth'. Therefore, scale the coordinate system
           by the current value of the 'stroke-width' property, which is 100. -->
      <g transform="scale(100)" >

        <!-- Translate the coordinate system by
             (-refX*viewBoxToMarkerUnitsScaleX, -refY*viewBoxToMarkerUnitsScaleY)
             in order that (refX,refY) within the marker will align with the vertex.
             In this case, we use the default value for preserveAspectRatio
             ('xMidYMid meet'), which means find a uniform scale factor
             (i.e., viewBoxToMarkerUnitsScaleX=viewBoxToMarkerUnitsScaleY)
             such that the viewBox fits entirely within the viewport ('meet') and
             is center-aligned ('xMidYMid'). In this case, the uniform scale factor
             is markerHeight/viewBoxHeight=3/10=.3. Therefore, translate by
             (-refX*.3,-refY*.3)=(0*.3,-5*.3)=(0,-1.5). -->
        <g transform="translate(0,-1.5)" >

          <!-- There is an implicit clipping path because the user agent style
               sheet says that the 'overflow' property for markers has the value
               'hidden'. To achieve this, create a clipping path at the bounds
               of the viewport. Note that in this case the viewport extends
               0.5 units to the left and right of the viewBox due to
               a uniform scale factor, different ratios for markerWidth/viewBoxWidth
               and markerHeight/viewBoxHeight, and 'xMidYMid' alignment -->
          <clipPath id="cp1" >
            <rect x="-0.5" y="0" width="4" height="3" />
          </clipPath>
          <g clip-path="url(#cp1)" >

            <!-- Scale the coordinate system by the uniform scale factor
                 markerHeight/viewBoxHeight=3/10=.3 to set the coordinate
                 system to viewBox units. -->
            <g transform="scale(.3)" >

              <!-- This 'g' element carries all property values that result from
                   cascading and inheritance of properties on the original 'marker' element.
                   In this example, neither fill nor stroke was specified on the 'marker'
                   element or any ancestors of the 'marker', so the initial values of
                   "black" and "none" are used, respectively. -->
              <g fill="black" stroke="none" >

                <!-- Expand out the contents of the 'marker' element. -->
                <path d="M 0 0 L 10 5 L 0 10 z" />
              </g>
            </g>
          </g>
        </g>
      </g>
    </g>
  </g>
</svg>
```

*View this example as SVG (SVG-enabled browsers only)*

## 11.7 Rendering properties

### 11.7.1 Color interpolation properties: 'color-interpolation' and 'color-interpolation-filters'

The SVG user agent performs color interpolations and compositing at various points as it processes SVG

content. Two properties, **'color-interpolation'** and **'color-interpolation-filters'**, control which color space is used for particular categories of graphics operations. The following table shows which property applies to which graphics operations:

| Graphics operation | Corresponding property |
|---|---|
| interpolating between gradient stops (see Gradient) | **'color-interpolation'** |
| interpolating color when performing color animations(see **'animateColor'**) | **'color-interpolation'** |
| alpha compositing of graphics elements into the current background | **'color-interpolation'** |
| filter effects | **'color-interpolation-filters'** |

Both properties choose between color operations occurring in the sRGB color space or in a (light energy linear) linearized RGB color space.

The conversion formulas between the sRGB color space (i.e., nonlinear with 2.2 gamma curve) and the linearized RGB color space (i.e., color values expressed as sRGB tristimulus values without a gamma curve) can be found in [SRGB]. For illustrative purposes, the following formula shows the conversion from sRGB to linearized RGB:

```
  R[sRGB] = R[sRGB-8bit] / 255
  G[sRGB] = G[sRGB-8bit] / 255
  B[sRGB] = B[sRGB-8bit] / 255

If R[sRGB], G[sRGB], B[sRGB] <= 0.04045

  R[linearRGB] = R[sRGB] / 12.92
  G[linearRGB] = G[sRGB] / 12.92
  B[linearRGB] = B[sRGB] / 12.92

else if R[sRGB], G[sRGB], B[sRGB] > 0.04045

  R[linearRGB] = ((R[sRGB] + 0.055) / 1.055) ^ 2.4
  G[linearRGB] = ((G[sRGB] + 0.055) / 1.055) ^ 2.4
  B[linearRGB] = ((B[sRGB] + 0.055) / 1.055) ^ 2.4

  R[linearRGB-8bit] = R[linearRGB] * 255
  G[linearRGB-8bit] = G[linearRGB] * 255
  B[linearRGB-8bit] = B[linearRGB] * 255
```

Out-of-range color values, if supported by the user agent, also are converted using the above formulas. (See Clamping values which are restricted to a particular range.)

**'color-interpolation'**

| | |
|---|---|
| *Value:* | auto \| sRGB \| linearRGB \| inherit |
| *Initial:* | sRGB |
| *Applies to:* | container elements, graphics elements and **'animateColor'** |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**

Indicates that the user agent can choose either the **sRGB** or **linearRGB** spaces for color interpolation. This option indicates that the author doesn't require that color interpolation occur in a

particular color space.

**sRGB**

Indicates that color interpolation should occur in the sRGB color space.

**linearRGB**

Indicates that color interpolation should occur in the linearized RGB color space as described above.

The **'color-interpolation'** property specifies the color space for gradient interpolations, color animations and alpha compositing.

When a child element is blended into a background, the value of the **'color-interpolation'** property on the child determines the type of blending, not the value of the **'color-interpolation'** on the parent. For gradients which make use of the **xlink:href** attribute to reference another gradient, the gradient uses the **'color-interpolation'** property value from the gradient element which is directly referenced by the **'fill'** or **'stroke'** property. When animating colors, color interpolation is performed according to the value of the **'color-interpolation'** property on the element being animated.

**'color-interpolation-filters'**

| | |
|---|---|
| *Value:* | auto \| sRGB \| linearRGB \| inherit |
| *Initial:* | linearRGB |
| *Applies to:* | filter primitives |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**

Indicates that the user agent can choose either the **sRGB** or **linearRGB** spaces for filter effects color operations. This option indicates that the author doesn't require that color operations occur in a particular color space.

**sRGB**

Indicates that filter effects color operations should occur in the sRGB color space.

**linearRGB**

Indicates that filter effects color operations should occur in the linearized RGB color space.

The **'color-interpolation-filters'** property specifies the color space for imaging operations performed via filter effects.

Note that **'color-interpolation-filters'** has a different initial value than **'color-interpolation'**. **'color-interpolation-filters'** has an initial value of **linearRGB**, whereas **'color-interpolation'** has an initial value of **sRGB**. Thus, in the default case, filter effects operations occur in the linearRGB color space, whereas all other color interpolations occur by default in the sRGB color space.

## 11.7.2 The 'color-rendering' property

The creator of SVG content might want to provide a hint to the implementation about how to make speed vs. quality tradeoffs as it performs color interpolation and compositing. The **'color-rendering'** property provides a hint to the SVG user agent about how to optimize its color interpolation and compositing operations.

**'color-rendering'** takes precedence over **'color-interpolation-filters'**. For example, assume **'color-**

**rendering:optimizeSpeed'** and **'color-interpolation-filters:linearRGB'**. In this case, the SVG user agent should perform color operations in a way that optimizes performance, which might mean sacrificing the color interpolation precision as specified by **'color-interpolation-filters:linearRGB'**.

**'color-rendering'**

| | |
|---|---|
| *Value:* | auto \| optimizeSpeed \| optimizeQuality \| inherit |
| *Initial:* | auto |
| *Applies to:* | container elements, graphics elements and **'animateColor'** |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**
Indicates that the user agent shall make appropriate tradeoffs to balance speed and quality, but quality shall be given more importance than speed.

**optimizeSpeed**
Indicates that the user agent shall emphasize rendering speed over quality. For RGB display devices, this option will sometimes cause the user agent to perform color interpolation and compositing in the device RGB color space.

**optimizeQuality**
Indicates that the user agent shall emphasize quality over rendering speed.

### 11.7.3 The 'shape-rendering' property

The creator of SVG content might want to provide a hint to the implementation about what tradeoffs to make as it renders vector graphics elements such as 'path' elements and basic shapes such as circles and rectangles. The **'shape-rendering'** property provides these hints.

**'shape-rendering'**

| | |
|---|---|
| *Value:* | auto \| optimizeSpeed \| crispEdges \| geometricPrecision \| inherit |
| *Initial:* | auto |
| *Applies to:* | shapes |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**
Indicates that the user agent shall make appropriate tradeoffs to balance speed, crisp edges and geometric precision, but with geometric precision given more importance than speed and crisp edges.

**optimizeSpeed**
Indicates that the user agent shall emphasize rendering speed over geometric precision and crisp edges. This option will sometimes cause the user agent to turn off shape anti-aliasing.

**crispEdges**
Indicates that the user agent shall attempt to emphasize the contrast between clean edges of artwork over rendering speed and geometric precision. To achieve crisp edges, the user agent might turn off anti-aliasing for all lines and curves or possibly just for straight lines which are close to vertical or horizontal. Also, the user agent might adjust line positions and line widths to align edges with device pixels.

**geometricPrecision**
Indicates that the user agent shall emphasize geometric precision over speed and crisp edges.

### 11.7.4 The 'text-rendering' property

The creator of SVG content might want to provide a hint to the implementation about what tradeoffs to make as it renders text. The **'text-rendering'** property provides these hints.

**'text-rendering'**

| | |
|---|---|
| *Value:* | auto \| optimizeSpeed \| optimizeLegibility \| geometricPrecision \| inherit |
| *Initial:* | auto |
| *Applies to:* | **'text'** elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**
Indicates that the user agent shall make appropriate tradeoffs to balance speed, legibility and geometric precision, but with legibility given more importance than speed and geometric precision.
**optimizeSpeed**
Indicates that the user agent shall emphasize rendering speed over legibility and geometric precision. This option will sometimes cause the user agent to turn off text anti-aliasing.
**optimizeLegibility**
Indicates that the user agent shall emphasize legibility over rendering speed and geometric precision. The user agent will often choose whether to apply anti-aliasing techniques, built-in font hinting or both to produce the most legible text.
**geometricPrecision**
Indicates that the user agent shall emphasize geometric precision over legibility and rendering speed. This option will usually cause the user agent to suspend the use of hinting so that glyph outlines are drawn with comparable geometric precision to the rendering of path data.

### 11.7.5 The 'image-rendering' property

The creator of SVG content might want to provide a hint to the implementation about how to make speed vs. quality tradeoffs as it performs image processing. The **'image-rendering'** property provides a hint to the SVG user agent about how to optimize its image rendering.:

**'image-rendering'**

| | |
|---|---|
| *Value:* | auto \| optimizeSpeed \| optimizeQuality \| inherit |
| *Initial:* | auto |
| *Applies to:* | images |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**
Indicates that the user agent shall make appropriate tradeoffs to balance speed and quality, but quality shall be given more importance than speed. The user agent shall employ a resampling algorithm at least as good as nearest neighbor resampling, but bilinear resampling is strongly

preferred. For Conforming High-Quality SVG Viewers, the user agent shall employ a resampling algorithm at least as good as bilinear resampling.

**optimizeQuality**
: Indicates that the user agent shall emphasize quality over rendering speed. The user agent shall employ a resampling algorithm at least as good as bilinear resampling.

**optimizeSpeed**
: Indicates that the user agent shall emphasize rendering speed over quality. The user agent should use a resampling algorithm which achieves the goal of fast rendering, with the requirement that the resampling algorithm shall be at least as good as nearest neighbor resampling. If performance goals can be achieved with higher quality algorithms, then the user agent should use the higher quality algorithms instead of nearest neighbor resampling.

In all cases, resampling must be done in a truecolor (e.g., 24-bit) color space even if the original data and/or the target device is indexed color.

## 11.8 Inheritance of painting properties

The values of any of the painting properties described in this chapter can be inherited from a given object's parent. Painting, however, is always done on each graphics element individually, never at the container element (e.g., a **'g'**) level. Thus, for the following SVG, even though the gradient fill is specified on the **'g'**, the gradient is simply inherited through the **'g'** element down into each rectangle, each of which is rendered such that its interior is painted with the gradient.

Example Inheritance

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="7cm" height="2cm" viewBox="0 0 700 200"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Gradients apply to leaf nodes
  </desc>
  <g>
    <defs>
      <linearGradient id="MyGradient" gradientUnits="objectBoundingBox">
        <stop offset="0%" stop-color="#F60" />
        <stop offset="100%" stop-color="#FF6" />
      </linearGradient>
    </defs>
    <rect x="1" y="1" width="698" height="198"
          fill="none" stroke="blue" stroke-width="2" />
    <g fill="url(#MyGradient)" >
      <rect x="100" y="50" width="200" height="100"/>
      <rect x="400" y="50" width="200" height="100"/>
   </g>
  </g>
</svg>
```

***Example Inheritance***

*View this example as SVG (SVG-enabled browsers only)*

Any painting properties defined in terms of the object's bounding box use the bounding box of the graphics element to which the operation applies. Note that text elements are defined such that any painting operations defined in terms of the object's bounding box use the bounding box of the entire **'text'** element. (See the discussion of object bounding box units and text elements.)

## 11.9 DOM interfaces

The following interfaces are defined below: SVGPaint, SVGMarkerElement.

### Interface SVGPaint

The **SVGPaint** interface corresponds to basic type <paint> and represents the values of properties **'fill'** and **'stroke'**.

**IDL Definition**

```
interface SVGPaint : SVGColor {
  // Paint Types
  const unsigned short SVG_PAINTTYPE_UNKNOWN               = 0;
  const unsigned short SVG_PAINTTYPE_RGBCOLOR              = 1;
  const unsigned short SVG_PAINTTYPE_RGBCOLOR_ICCCOLOR     = 2;
  const unsigned short SVG_PAINTTYPE_NONE                  = 101;
  const unsigned short SVG_PAINTTYPE_CURRENTCOLOR          = 102;
  const unsigned short SVG_PAINTTYPE_URI_NONE              = 103;
  const unsigned short SVG_PAINTTYPE_URI_CURRENTCOLOR      = 104;
  const unsigned short SVG_PAINTTYPE_URI_RGBCOLOR          = 105;
  const unsigned short SVG_PAINTTYPE_URI_RGBCOLOR_ICCCOLOR = 106;
  const unsigned short SVG_PAINTTYPE_URI                   = 107;

  readonly attribute unsigned short paintType;
  readonly attribute DOMString      uri;

  void setUri ( in DOMString uri );
  void setPaint ( in unsigned short paintType, in DOMString uri, in DOMString rgbColor, in DOMString iccColor )
                 raises( SVGException );
};
```

**Definition group Paint Types**

**Defined constants**

| | |
|---|---|
| SVG_PAINTTYPE_UNKNOWN | The paint type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |

| | |
|---|---|
| SVG_PAINTTYPE_RGBCOLOR | An sRGB color has been specified without an alternative ICC color specification. |
| SVG_PAINTTYPE_RGBCOLOR_ICCCOLOR | An sRGB color has been specified along with an alternative ICC color specification. |
| SVG_PAINTTYPE_NONE | Corresponds to a 'none' value on a <paint> specification. |
| SVG_PAINTTYPE_CURRENTCOLOR | Corresponds to a 'currentColor' value on a <paint> specification. |
| SVG_PAINTTYPE_URI_NONE | A URI has been specified, along with an explicit 'none' as the backup paint method in case the URI is unavailable or invalid. |
| SVG_PAINTTYPE_URI_CURRENTCOLOR | A URI has been specified, along with 'currentColor' as the backup paint method in case the URI is unavailable or invalid. |
| SVG_PAINTTYPE_URI_RGBCOLOR | A URI has been specified, along with an sRGB color as the backup paint method in case the URI is unavailable or invalid. |
| SVG_PAINTTYPE_URI_RGBCOLOR_ICCCOLOR | A URI has been specified, along with both an sRGB color and alternate ICC color as the backup paint method in case the URI is unavailable or invalid. |
| SVG_PAINTTYPE_URI | Only a URI has been specified. |

**Attributes**

**readonly unsigned short paintType**

The type of paint, identified by one of the constants above.

**readonly DOMString uri**

When the paintType specifies a URI, this attribute holds the URI string. When the paintType does not specify a URI, this attribute is null.

**Methods**

**setUri**

Sets the paintType to SVG_PAINTTYPE_URI_NONE and sets uri to the specified value.

**Parameters**

in DOMString uri The URI for the desired paint server.

**No Return Value**

**No Exceptions**

**setPaint**

Sets the paintType as specified by the parameters. If `paintType` requires a URI, then `uri` must be non-null and a valid string; otherwise, `uri` must be null. If `paintType` requires an RGBColor, then `rgbColor` must be a valid RGBColor object; otherwise, `rgbColor` must be null. If `paintType` requires an SVGICCColor, then `iccColor` must be a valid SVGICCColor object; otherwise, `iccColor` must be null.

**Parameters**

| | |
|---|---|
| in unsigned short paintType | One of the defined constants for paintType. |
| in DOMString uri | The URI for the desired paint server, or null. |
| in DOMString rgbColor | The specification of an sRGB color, or null. |
| in DOMString iccColor | The specification of an ICC color, or null. |

**No Return Value**

**Exceptions**

| | |
|---|---|
| SVGException | SVG_INVALID_VALUE_ERR: Raised if one of the parameters has an invalid value. |

## Interface SVGMarkerElement

The **SVGMarkerElement** interface corresponds to the **'marker'** element.

**IDL Definition**

```
interface SVGMarkerElement :
                SVGElement,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGFitToViewBox {

  // Marker Unit Types
  const unsigned short SVG_MARKERUNITS_UNKNOWN        = 0;
  const unsigned short SVG_MARKERUNITS_USERSPACEONUSE = 1;
  const unsigned short SVG_MARKERUNITS_STROKEWIDTH    = 2;
  // Marker Orientation Types
  const unsigned short SVG_MARKER_ORIENT_UNKNOWN      = 0;
  const unsigned short SVG_MARKER_ORIENT_AUTO         = 1;
  const unsigned short SVG_MARKER_ORIENT_ANGLE        = 2;

  readonly attribute SVGAnimatedLength      refX;
  readonly attribute SVGAnimatedLength      refY;
  readonly attribute SVGAnimatedEnumeration markerUnits;
  readonly attribute SVGAnimatedLength      markerWidth;
  readonly attribute SVGAnimatedLength      markerHeight;
  readonly attribute SVGAnimatedEnumeration orientType;
  readonly attribute SVGAnimatedAngle      orientAngle;

  void setOrientToAuto (  );
  void setOrientToAngle ( in SVGAngle angle );
};
```

**Definition group Marker Unit Types**

**Defined constants**

| | |
|---|---|
| SVG_MARKERUNITS_UNKNOWN | The marker unit type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_MARKERUNITS_USERSPACEONUSE | The value of attribute **markerUnits** is 'userSpaceOnUse'. |
| SVG_MARKERUNITS_STROKEWIDTH | The value of attribute **markerUnits** is 'strokeWidth'. |

**Definition group Marker Orientation Types**

**Defined constants**

| | |
|---|---|
| SVG_MARKER_ORIENT_UNKNOWN | The marker orientation is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |

| | |
|---|---|
| SVG_MARKER_ORIENT_AUTO | Attribute **orient** has value 'auto'. |
| SVG_MARKER_ORIENT_ANGLE | Attribute **orient** has an angle value. |

**Attributes**

**readonly SVGAnimatedLength refX**
Corresponds to attribute **refX** on the given **'marker'** element.

**readonly SVGAnimatedLength refY**
Corresponds to attribute **refY** on the given **'marker'** element.

**readonly SVGAnimatedEnumeration markerUnits**
Corresponds to attribute **markerUnits** on the given **'marker'** element. One of the Marker Units Types defined above.

**readonly SVGAnimatedLength markerWidth**
Corresponds to attribute **markerWidth** on the given **'marker'** element.

**readonly SVGAnimatedLength markerHeight**
Corresponds to attribute **markerHeight** on the given **'marker'** element.

**readonly SVGAnimatedEnumeration orientType**
Corresponds to attribute **orient** on the given **'marker'** element. One of the Marker Orientation Types defined above.

**readonly SVGAnimatedAngle orientAngle**
Corresponds to attribute **orient** on the given **'marker'** element. If markerUnits is SVG_MARKER_ORIENT_ANGLE, the angle value for attribute **orient**; otherwise, it will be set to zero.

**Methods**

**setOrientToAuto**
Sets the value of attribute **orient** to 'auto'.
**No Parameters**
**No Return Value**
**No Exceptions**

**setOrientToAngle**
Sets the value of attribute **orient** to the given angle.
**Parameters**

| | |
|---|---|
| in SVGAngle angle | The angle value to use for attribute **orient**. |

**No Return Value**
**No Exceptions**

---

04 September 2001

# 12 Color

## Contents



## 12.1 Introduction

All SVG colors are specified in the sRGB color space (see [SRGB]). At a minimum, SVG user agents shall conform to the color behavior requirements specified in the color units section and the minimal gamma correction rules defined in the CSS2 specification.

Additionally, SVG content can specify an alternate color specification using an ICC profile (see [ICC32]). If ICC-based colors are provided and the SVG user agent supports ICC color, then the ICC-based color takes precedence over the sRGB color specification. Note that color interpolation occurs in an RGB color space even if an ICC-based color specification is provided (see **'color-interpolation'**).

## 12.2 The 'color' property

The **'color'** property is used to provide a potential indirect value (**currentColor**) for the **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'**, **'lighting-color'** properties.

**'color'**

| | |
|---|---|
| *Value:* | <color> \| inherit |
| *Initial:* | depends on user agent |
| *Applies to:* | elements to which properties **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'**, **'lighting-color'** apply |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

Except for any additional information provided in this specification, the normative definition of the property is in [CSS2].

## 12.3 Color profile descriptions

### 12.3.1 Overview of color profile descriptions

The International Color Consortium has established a standard, the ICC Profile [ICC32], for documenting the color characteristics of input and output devices. Using these profiles, it is possible to build a transform and correct visual data for viewing on different devices.

A **color profile description** provides the bridge between an ICC profile and references to that ICC profile within SVG content. The color profile description is added to the user agent's list of known color profiles and then used to select the relevant profile. The color profile description contains descriptors for the location of the color profile on the Web, a name to reference the profile and information about rendering intent.

### 12.3.2 Alternative ways for defining a color profile description

Color profile descriptions can be specified in either of the following ways:

- a **'color-profile'** element
- an *@color-profile* rule within a CSS style sheet (only applicable for user agents which support using CSS [CSS2] to style the SVG content)

If a color profile with the same *name* value has been identified by both a **'color-profile'** element and *@color-profile* rules within a CSS style sheet, then the user agent shall first attempt to locate the profile by using the specifications in the *@color-profile* rules first.

### 12.3.3 The 'color-profile' element

```
<!ELEMENT color-profile (%descTitleMetadata;) >
<!ATTLIST color-profile
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  local CDATA #IMPLIED
  name CDATA #REQUIRED
  rendering-intent (auto | perceptual | relative-colorimetric | saturation | absolute-colorimetric) "auto" >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**
: The location of an ICC profile resource.
*Animatable: no.*

**`local` = "<string>"**
: The unique ID for a locally stored color profile. <string> is the profile's unique ID as specified by International Color Consortium. If both the **xlink:href** and the **local** attributes are specified, then the user agent shall search the local system for the locally stored color profile first, and, if not available locally, then attempt to use the resource identified by the **xlink:href** attribute. (Note: Profile description fields do *not* represent a profile's unique ID. With current ICC proposals, the profile's unique ID is an MD5-encoded value within the profile

header.).
*Animatable: no.*

**name = "<name>"**
The name which is used as the first parameter for **icc-color** specifications within **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'** and **'lighting-color'** property values to identify the color profile to use for the ICC color specification and the name which can be the value of the **'color-profile'** property. Note that if <name> is not provided, it will be impossible to reference the given color profile description. The name "sRGB" is predefined; any color profile descriptions with **<name>** set to "sRGB" will be ignored. For consistency with CSS lexical scanning and parsing rules, the keyword "sRGB" is case-insensitive; however, it is recommended that the mixed capitalization "sRGB" be used for consistency with common industry practice.
*Animatable: no.*

**rendering-intent = "auto | perceptual | relative-colorimetric | saturation | absolute-colorimetric"**

**'rendering-intent'** permits the specification of a color profile rendering intent other than the default. **'rendering-intent'** is applicable primarily to color profiles corresponding to CMYK color spaces. The different options cause different methods to be used for translating colors to the color gamut of the target rendering device:

**auto**
This is the default behavior. The user agent determines the best intent based on the content type. For image content containing an embedded profile, it shall be assumed that the intent specified within the profile is the desired intent. Otherwise, the user agent shall use the current profile and force the intent, overriding any intent that might be stored in the profile itself.

**perceptual**
This method, often the preferred choice for images, preserves the relationship between colors. It attempts to maintain relative color values among the pixels as they are mapped to the target device gamut. Sometimes pixel values that were originally within the target device gamut are changed in order to avoid hue shifts and discontinuities and to preserve as much as possible the overall appearance of the scene.

**saturation**
Preserves the relative saturation (chroma) values of the original pixels. Out of gamut colors are converted to colors that have the same saturation but fall just inside the gamut.

**relative colorimetric**
Leaves colors that fall inside the gamut unchanged. This method usually converts out of gamut colors to colors that have the same lightness but fall just inside the gamut.

**absolute colorimetric**
Disables white point matching when converting colors. This option is generally not recommended.

*Animatable: no.*

***Attributes defined elsewhere:***
**%stdAttrs;, %xlinkRefAttrs;.**

## 12.3.4 @color-profile when using CSS styling

When the document is styled using CSS, the **@color-profile** rule can be used to specify a color profile description. The general form is:

```
@color-profile { <color-profile-description> }
```

where the <color-profile-description> has the form:

```
descriptor: value;
[...]
descriptor: value;
```

Each @color-profile rule specifies a value for every color profile descriptor, either implicitly or explicitly. Those not given explicit values in the rule take the initial value listed with each descriptor in this specification. These descriptors apply solely within the context of the @color-profile rule in which they are defined, and do not apply to document language elements. Thus, there is no notion of which elements the descriptors apply to, or whether the values are inherited by child elements.

The following are the descriptors for a <color-profile-description>:

**'src' (Descriptor)**

*Values:* sRGB | <local-profile> | <uri> | (<local-profile> <uri>) | inherit
*Initial:* sRGB
*Media:* visual

**sRGB**

The source profile is the sRGB color space. For consistency with CSS lexical scanning and parsing rules, the keyword "sRGB" is case-insensitive; however, it is recommended that the mixed capitalization "sRGB" be used for consistency with common industry practice.

**<local-profile>**

The source profile is a locally-stored profile. The syntax for <local-profile> is:

```
"local(" + <string> + ")"
```

where <string> is the profile's unique ID as specified by International Color Consortium. (Note: Profile description fields do *not* represent a profile's unique ID. With current ICC proposals, the profile's unique ID is an MD5-encoded value within the profile header.)

**<uri>**

The source profile is a URI reference to a color profile.

**(<local-profile> <uri>)**

Two profiles are specified. If <local-profile> cannot be found on the local system, then the <uri> is used.

**'name' (Descriptor)**

*Values:* <name>
*Initial:* undefined
*Media:* visual

**<name>**

See the description for the **name** attribute on the **'color-profile'** element. Note that if <name> is not provided, it will be impossible to reference the given @color-profile definition.

**'rendering-intent' (Descriptor)**

| | |
|---|---|
| *Values:* | auto \| perceptual \| relative-colorimetric \| saturation \| absolute-colorimetric |
| *Initial:* | auto |
| *Media:* | visual |
| *Animatable:* | no |

See the description for the **rendering-intent** attribute on the **'color-profile'** element.

## 12.3.5 'color-profile' property

**'color-profile'**

| | |
|---|---|
| *Value:* | auto \| sRGB \| <name> \| <uri> \| inherit |
| *Initial:* | auto |
| *Applies to:* | **'image'** elements that refer to raster images |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**auto**

This is the default behavior. All colors are presumed to be defined in the sRGB color space unless a more precise embedded profile is specified within content data. For images that do have a profile built into their data, that profile is used. For images that do not have a profile, the sRGB profile is used.

**sRGB**

The source profile is assumed to be sRGB. This differs from auto in that it overrides an embedded profile inside an image.

For consistency with CSS lexical scanning and parsing rules, the keyword "sRGB" is case-insensitive; however, it is recommended that the mixed capitalization "sRGB" be used for consistency with common industry practice.

**<name>**

A name corresponding to a defined color profile that is in the user agent's color profile description database. The user agent searches the color profile description database for a color profile description entry whose name descriptor matches <name> and uses the last matching entry that is found. If a match is found, the corresponding profile overrides an embedded profile inside an image. If no match is found, then the embedded profile inside the image is used.

**<uri>**

A URI reference to the source color profile. The referenced color profile overrides an embedded profile inside the image.

## 12.4 DOM interfaces

The following interfaces are defined below: SVGColorProfileElement, SVGColorProfileRule.

### Interface SVGColorProfileElement

The **SVGColorProfileElement** interface corresponds to the **'color-profile'** element.

**IDL Definition**

```
interface SVGColorProfileElement :
               SVGElement,
               SVGURIReference,
               SVGRenderingIntent {

          attribute DOMString      local;
                      // raises DOMException on setting
          attribute DOMString      name;
                      // raises DOMException on setting
          attribute unsigned short renderingIntent;
                      // raises DOMException on setting
};
```

**Attributes**

**DOMString local**

Corresponds to attribute **local** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString name**

Corresponds to attribute **name** on the given element.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**unsigned short renderingIntent**

Corresponds to attribute **rendering-intent** on the given element. The type of rendering intent, identified by one of the SVGRenderingIntent constants.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

## Interface SVGColorProfileRule

The **SVGColorProfileRule** interface represents an @color-profile rule in a CSS style sheet. An @color-profile rule identifies a ICC profile which can be referenced within a given document.

Support for the **SVGColorProfileRule** interface is only required in user agents that support styling with CSS.

**IDL Definition**

```
interface SVGColorProfileRule :
                SVGCSSRule,
                SVGRenderingIntent {

          attribute DOMString      src;
                      // raises DOMException on setting
          attribute DOMString      name;
                      // raises DOMException on setting
          attribute unsigned short renderingIntent;
                      // raises DOMException on setting
};
```

**Attributes**

**DOMString src**

Corresponds to property **src** within an @color-profile rule.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**DOMString name**

Corresponds to property **name** within an @color-profile rule.

**Exceptions on setting**

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute.

**unsigned short renderingIntent**

The type of rendering intent, identified by one of the SVGRenderingIntent constants.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

---

04 September 2001

# 13 Gradients and Patterns

## Contents



## 13.1 Introduction

With SVG, you can fill (i.e., paint the interior) or stroke (i.e., paint the outline) of shapes and text using one of the following:

- color
- gradients (linear or radial)
- patterns (vector or image, possibly tiled)

SVG uses the general notion of a **paint server**. Gradients and patterns are just specific types of built-in paint servers.

Paint servers are referenced using a URI reference on a **'fill'** or **'stroke'** property.

## 13.2 Gradients

### 13.2.1 Introduction

Gradients consist of continuously smooth color transitions along a vector from one color to another, possibly followed by additional transitions along the same vector to other colors. SVG provides for two types of gradients, **linear gradients** and **radial gradients**.

Once defined, gradients are then referenced using **'fill'** or **'stroke'** properties on a given graphics element to indicate that the given element shall be filled or stroked with the referenced gradient.

## 13.2.2 Linear gradients

Linear gradients are defined by a **'linearGradient'** element.

```
<!ENTITY % linearGradientExt "" >
<!ELEMENT linearGradient (%descTitleMetadata;,(stop|animate|set|animateTransform
                  %linearGradientExt;)*) >
<!ATTLIST linearGradient
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  gradientUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  gradientTransform %TransformList; #IMPLIED
  x1 %Coordinate; #IMPLIED
  y1 %Coordinate; #IMPLIED
  x2 %Coordinate; #IMPLIED
  y2 %Coordinate; #IMPLIED
  spreadMethod (pad | reflect | repeat) #IMPLIED >
```

*Attribute definitions:*

**`gradientUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for attributes x1, y1, x2, y2.

If **gradientUnits="userSpaceOnUse"**, **x1, y1, x2, y2** represent values in the coordinate system that results from taking the current user coordinate system in place at the time when the gradient element is referenced (i.e., the user coordinate system for the element referencing the gradient element via a **'fill'** or **'stroke'** property) and then applying the transform specified by attribute **gradientTransform**.

If **gradientUnits="objectBoundingBox"**, the user coordinate system for attributes **x1, y1, x2, y2** is established using the bounding box of the element to which the gradient is applied (see Object bounding box units) and then applying the transform specified by attribute **gradientTransform**.

When **gradientUnits="objectBoundingBox"** and **gradientTransform** is the identity matrix, the stripes of the linear gradient are perpendicular to the gradient vector in object bounding box space (i.e., the abstract coordinate system where (0,0) is at the top/left of the object bounding box and (1,1) is at the bottom/right of the object bounding box). When the object's bounding box is not square, the stripes that are conceptually perpendicular to the gradient vector within object bounding box space will render non-perpendicular relative to the gradient vector in user space due to application of the non-uniform scaling transformation from bounding box space to user space.
If attribute **gradientUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.
*Animatable: yes.*

**`gradientTransform` = "<transform-list>"**

Contains the definition of an optional additional transformation from the gradient coordinate system onto the target coordinate system (i.e., userSpaceOnUse or objectBoundingBox). This allows for things such as skewing the gradient. This additional transformation matrix is post-multiplied to (i.e., inserted to the right of) any previously defined transformations, including the implicit transformation necessary to convert from object bounding box units to user space.
If attribute **gradientTransform** is not specified, then the effect is as if an identity transform were specified.
*Animatable: yes.*

**`x1` = "<coordinate>"**

**x1, y1, x2, y2** define a *gradient vector* for the linear gradient. This *gradient vector* provides starting and ending points onto which the gradient stops are mapped. The values of **x1, y1, x2, y2** can be either numbers or percentages.
If the attribute is not specified, the effect is as if a value of "0%" were specified.
*Animatable: yes.*

**`y1` = "<coordinate>"**

See x1.
If the attribute is not specified, the effect is as if a value of "0%" were specified.
*Animatable: yes.*

**`x2` = "<coordinate>"**

See x1.
If the attribute is not specified, the effect is as if a value of "100%" were specified.
*Animatable: yes.*

**`y2` = "<coordinate>"**

See x1.
If the attribute is not specified, the effect is as if a value of "0%" were specified.
*Animatable: yes.*

**`spreadMethod` = "pad | reflect | repeat"**

Indicates what happens if the gradient starts or ends inside the bounds of the *target rectangle*. Possible values are: *pad*, which says to use the terminal colors of the gradient to fill the remainder of the target region, *reflect*, which says to reflect the gradient pattern start-to-end, end-to-start, start-to-end, etc. continuously until the *target rectangle* is filled, and *repeat*, which says to repeat the gradient pattern start-to-end, start-to-end, start-to-end, etc. continuously until the target region is filled.
If the attribute is not specified, the effect is as if a value of "pad" were specified.
*Animatable: yes.*

**`xlink:href` = "<uri>"**

A URI reference to a different **'linearGradient'** or **'radialGradient'** element within the current SVG document fragment. Any **'linearGradient'** attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no defined gradient stops, and the referenced element does (possibly due to its own **href** attribute), then this element inherits the gradient stop from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attribute or gradient stops due to its own **href** attribute, then the current element can inherit those attributes or gradient stops.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %xlinkRefAttrs;, externalResourcesRequired, %PresentationAttributes-Color;, %PresentationAttributes-Gradients;.**

Percentages are allowed for **x1, y1, x2, y2**. For gradientUnits="userSpaceOnUse", percentages represent values relative to the current viewport. For gradientUnits="objectBoundingBox", percentages represent values relative to the bounding box for the object.

If **x1** = **x2** and **y1** = **y2**, then the area to be painted will be painted as a single color using the color and opacity of the last gradient stop.

Properties inherit into the **'linearGradient'** element from its ancestors; properties do *not* inherit from the element referencing the **'linearGradient'** element.

**'linearGradient'** elements are never rendered directly; their only usage is as something that can be referenced using the **'fill'** and **'stroke'** properties. The **'display'** property does not apply to the **'linearGradient'** element; thus, **'linearGradient'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'linearGradient'** elements are available for referencing even when the **'display'** property on the **'linearGradient'** element or any of its ancestors is set to **none**.

Example lingrad01 shows how to fill a rectangle by referencing a linear gradient paint server.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example lingrad01 - fill a rectangle using a
          linear gradient paint server</desc>
  <g>
    <defs>
      <linearGradient id="MyGradient">
        <stop offset="5%" stop-color="#F60" />
        <stop offset="95%" stop-color="#FF6" />
      </linearGradient>
    </defs>
```

```
        <!-- Outline the drawing area in blue -->
        <rect fill="none" stroke="blue"
              x="1" y="1" width="798" height="398"/>

        <!-- The rectangle is filled using a linear gradient paint server -->
        <rect fill="url(#MyGradient)" stroke="black" stroke-width="5"
              x="100" y="100" width="600" height="200"/>
      </g>
    </svg>
```

***Example lingrad01***

*View this example as SVG (SVG-enabled browsers only)*

## 13.2.3 Radial gradients

Radial gradients are defined by a **'radialGradient'** element.

```
<!ENTITY % radialGradientExt "" >
<!ELEMENT radialGradient (%descTitleMetadata;,(stop|animate|set|animateTransform
                  %radialGradientExt;)*) >
<!ATTLIST radialGradient
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  gradientUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  gradientTransform %TransformList; #IMPLIED
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  r %Length; #IMPLIED
  fx %Coordinate; #IMPLIED
```

```
  fy %Coordinate; #IMPLIED
  spreadMethod (pad | reflect | repeat) #IMPLIED >
```

*Attribute definitions:*

**`gradientUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for attributes cx, cy, r, fx, fy.

If **gradientUnits="userSpaceOnUse"**, **cx, cy, r, fx, fy** represent values in the coordinate system that results from taking the current user coordinate system in place at the time when the gradient element is referenced (i.e., the user coordinate system for the element referencing the gradient element via a **'fill'** or **'stroke'** property) and then applying the transform specified by attribute **gradientTransform**.

If **gradientUnits="objectBoundingBox"**, the user coordinate system for attributes **cx, cy, r, fx, fy** is established using the bounding box of the element to which the gradient is applied (see Object bounding box units) and then applying the transform specified by attribute **gradientTransform**.

When **gradientUnits="objectBoundingBox"** and **gradientTransform** is the identity matrix, then the rings of the radial gradient are circular with respect to the object bounding box space (i.e., the abstract coordinate system where (0,0) is at the top/left of the object bounding box and (1,1) is at the bottom/right of the object bounding box). When the object's bounding box is not square, the rings that are conceptually circular within object bounding box space will render as elliptical due to application of the non-uniform scaling transformation from bounding box space to user space.

If attribute **gradientUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.

*Animatable: yes.*

**`gradientTransform` = "<transform-list>"**

Contains the definitions of an optional additional transformation from the gradient coordinate system onto the target coordinate system (i.e., userSpaceOnUse or objectBoundingBox). This allows for things such as skewing the gradient. This additional transformation matrix is post-multiplied to (i.e., inserted to the right of) any previously defined transformations, including the implicit transformation necessary to convert from object bounding box units to user space.

If attribute **gradientTransform** is not specified, then the effect is as if an identity transform were specified.

*Animatable: yes.*

**`cx` = "<coordinate>"**

**cx, cy, r** define the largest (i.e., outermost) circle for the radial gradient. The gradient will be drawn such that the 100% gradient stop is mapped to the perimeter of this largest (i.e., outermost) circle.

If the attribute is not specified, the effect is as if a value of "50%" were specified.

*Animatable: yes.*

**`cy` = "<coordinate>"**

See cx.

If the attribute is not specified, the effect is as if a value of "50%" were specified.

*Animatable: yes.*

**`r` = "<length>"**

See **cx**.

A negative value is an error (see Error processing). A value of zero will cause the area to be painted as a single color using the color and opacity of the last gradient stop.

If the attribute is not specified, the effect is as if a value of "50%" were specified.

*Animatable: yes.*

**`fx = "<coordinate>"`**

**fx, fy** define the focal point for the radial gradient. The gradient will be drawn such that the 0% gradient stop is mapped to (fx, fy).

If attribute **fx** is not specified, **fx** will coincide with **cx**.

*Animatable: yes.*

**`fy = "<coordinate>"`**

See **fx**.

If attribute **fy** is not specified, **fy** will coincide with **cy**.

*Animatable: yes.*

**`spreadMethod = "pad | reflect | repeat"`**

Indicates what happens if the gradient starts or ends inside the bounds of the object(s) being painted by the gradient. Has the same values and meanings as the **spreadMethod** attribute on **'linearGradient'** element.

*Animatable: yes.*

**`xlink:href = "<uri>"`**

A URI reference to a different **'linearGradient'** or **'radialGradient'** element within the current SVG document fragment. Any **'radialGradient'** attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no defined gradient stops, and the referenced element does (possibly due to its own **href** attribute), then this element inherits the gradient stop from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attribute or gradient stops due to its own **href** attribute, then the current element can inherit those attributes or gradient stops.

*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %xlinkRefAttrs;, externalResourcesRequired, %PresentationAttributes-Color;, %PresentationAttributes-Gradients;.**

Percentages are allowed for attributes **cx**, **cy**, **r**, **fx** and **fy**. For **gradientUnits="userSpaceOnUse"**, percentages represent values relative to the current viewport. For **gradientUnits="objectBoundingBox"**, percentages represent values relative to the bounding box for the object.

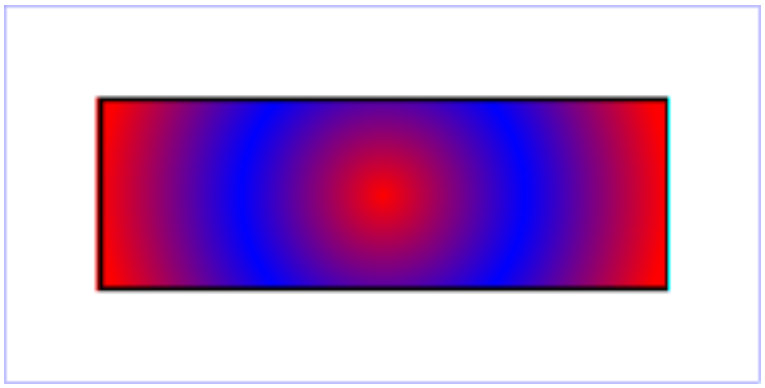
If the point defined by **fx** and **fy** lies outside the circle defined by **cx**, **cy** and **r**, then the user agent shall set the focal point to the intersection of the line from (**cx**, **cy**) to (**fx**, **fy**) with the circle defined by **cx**, **cy** and **r**.

Properties inherit into the **'radialGradient'** element from its ancestors; properties do *not* inherit from the element referencing the **'radialGradient'** element.

**'radialGradient'** elements are never rendered directly; their only usage is as something that can be referenced using the **'fill'** and **'stroke'** properties. The **'display'** property does not apply to the **'radialGradient'** element; thus, **'radialGradient'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'radialGradient'** elements are available for referencing even when the **'display'** property on the **'radialGradient'** element or any of its ancestors is set to **none**.

Example radgrad01 shows how to fill a rectangle by referencing a radial gradient paint server.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example radgrad01 - fill a rectangle by referencing a
          radial gradient paint server</desc>
  <g>
    <defs>
      <radialGradient id="MyGradient" gradientUnits="userSpaceOnUse"
                      cx="400" cy="200" r="300" fx="400" fy="200">
        <stop offset="0%" stop-color="red" />
        <stop offset="50%" stop-color="blue" />
        <stop offset="100%" stop-color="red" />
      </radialGradient>
    </defs>

    <!-- Outline the drawing area in blue -->
    <rect fill="none" stroke="blue"
          x="1" y="1" width="798" height="398"/>

    <!-- The rectangle is filled using a radial gradient paint server -->
    <rect fill="url(#MyGradient)" stroke="black" stroke-width="5"
          x="100" y="100" width="600" height="200"/>
  </g>
</svg>
```



***Example radgrad01***

*View this example as SVG (SVG-enabled browsers only)*

### 13.2.4 Gradient stops

The ramp of colors to use on a gradient is defined by the **'stop'** elements that are child elements to either the 'linearGradient' element or the 'radialGradient' element.

```
<!ENTITY % stopExt "" >
<!ELEMENT stop (animate|set|animateColor
                  %stopExt;)* >
<!ATTLIST stop
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  offset %NumberOrPercentage; #REQUIRED >
```

*Attribute definitions:*

**`offset` = "*<number>* | *<percentage>*"**

The **offset** attribute is either a <number> (usually ranging from 0 to 1) or a <percentage> (usually ranging from 0% to 100%) which indicates where the gradient stop is placed. For linear gradients, the **offset** attribute represents a location along the *gradient vector*. For radial gradients, it represents a percentage distance from (fx,fy) to the edge of the outermost/largest circle.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, class, style, %PresentationAttributes-Color;, %PresentationAttributes-Gradients;.**

The **'stop-color'** property indicates what color to use at that gradient stop. The keyword **currentColor** and ICC colors can be specified in the same manner as within a <paint> specification for the **'fill'** and **'stroke'** properties.

**'stop-color'**

*Value:* currentColor | <color> [icc-color(<name>[,<icccolorvalue>]*)] | inherit
*Initial:* black

*Applies to:* **'stop'** elements
*Inherited:* no
*Percentages:* N/A
*Media:* visual
*Animatable:* yes

The **'stop-opacity'** property defines the opacity of a given gradient stop.

**'stop-opacity'**

*Value:* <alphavalue> | inherit
*Initial:* 1
*Applies to:* **'stop'** elements
*Inherited:* no
*Percentages:* N/A
*Media:* visual
*Animatable:* yes

Some notes on gradients:

- Gradient offset values less than 0 (or less than 0%) are rounded up to 0%. Gradient offset values greater than 1 (or greater than 100%) are rounded down to 100%.
- It is necessary that at least two stops defined to have a gradient effect. If no stops are defined, then painting shall occur as if 'none' were specified as the paint style. If one stop is defined, then paint with the solid color fill using the color defined for that gradient stop.
- Each gradient offset value is required to be equal to or greater than the previous gradient stop's offset value. If a given gradient stop's offset value is not equal to or greater than all previous offset values, then the offset value is adjusted to be equal to the largest of all previous offset values.
- If two gradient stops have the same offset value, then the latter gradient stop controls the color value at the overlap point. In particular:

```
<stop offset="0" stop-color="white"/>
<stop offset=".2" stop-color="red"/>
<stop offset=".2" stop-color="blue"/>
<stop offset="1" stop-color="black"/>
```

will have approximately the same effect as:

```
<stop offset="0" stop-color="white"/>
<stop offset=".1999999999" stop-color="red"/>
<stop offset=".2" stop-color="blue"/>
<stop offset="1" stop-color="black"/>
```

which is a gradient that goes smoothly from white to red, then abruptly shifts from red to blue, and then goes smoothly from blue to black.

## 13.3 Patterns

A pattern is used to fill or stroke an object using a pre-defined graphic object which can be replicated ("tiled") at fixed intervals in *x* and *y* to cover the areas to be painted. Patterns are defined using a **'pattern'** element and then referenced by properties **'fill'** and **'stroke'** on a given graphics element to indicate that the given element shall be filled or stroked with the referenced pattern.

Attributes **x**, **y**, **width**, **height** and **patternUnits** define a reference rectangle somewhere on the infinite canvas. The reference rectangle has its top/left at (**x**,**y**) and its bottom/right at (**x**+**width**,**y**+**height**). The tiling theoretically extends a series of such rectangles to infinity in X and Y (positive and negative), with rectangles starting at (**x** + m*__width__, **y** + n*__height__) for each possible integer value for *m* and *n*.

```
<!ENTITY % patternExt "" >
<!ELEMENT pattern (desc|title|metadata|defs|
                   path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|view|switch|a|altGlyphDef|
                   script|style|symbol|marker|clipPath|mask|
                   linearGradient|radialGradient|pattern|filter|cursor|font|
                   animate|set|animateMotion|animateColor|animateTransform|
                   color-profile|font-face
                   %ceExt;%patternExt;)* >
<!ATTLIST pattern
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  patternUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  patternContentUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  patternTransform %TransformList; #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >
```

*Attribute definitions:*

**`patternUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for attributes **x, y, width, height**.
If **patternUnits="userSpaceOnUse"**, **x, y, width, height** represent values in the coordinate system that results from taking the current user coordinate system in place

at the time when the **'pattern'** element is referenced (i.e., the user coordinate system for the element referencing the **'pattern'** element via a **'fill'** or **'stroke'** property) and then applying the transform specified by attribute **patternTransform**.

If **patternUnits="objectBoundingBox"**, the user coordinate system for attributes **x, y, width, height** is established using the bounding box of the element to which the pattern is applied (see Object bounding box units) and then applying the transform specified by attribute **patternTransform**.

If attribute **patternUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.

*Animatable: yes.*

**`patternContentUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for the contents of the **'pattern'**. Note that this attribute has no effect if attribute **viewBox** is specified.

If **patternContentUnits="userSpaceOnUse"**, the user coordinate system for the contents of the **'pattern'** element is the coordinate system that results from taking the current user coordinate system in place at the time when the **'pattern'** element is referenced (i.e., the user coordinate system for the element referencing the **'pattern'** element via a **'fill'** or **'stroke'** property) and then applying the transform specified by attribute **patternTransform**.

If **patternContentUnits="objectBoundingBox"**, the user coordinate system for the contents of the **'pattern'** element is established using the bounding box of the element to which the pattern is applied (see Object bounding box units) and then applying the transform specified by attribute **patternTransform**.

If attribute **patternContentUnits** is not specified, then the effect is as if a value of **userSpaceOnUse** were specified.

*Animatable: yes.*

**`patternTransform` = "<transform-list>"**

Contains the definition of an optional additional transformation from the pattern coordinate system onto the target coordinate system (i.e., userSpaceOnUse or objectBoundingBox). This allows for things such as skewing the pattern tiles. This additional transformation matrix is post-multiplied to (i.e., inserted to the right of) any previously defined transformations, including the implicit transformation necessary to convert from object bounding box units to user space.

If attribute **patternTransform** is not specified, then the effect is as if an identity transform were specified.

*Animatable: yes.*

**x = "<coordinate>"**

**x, y, width, height** indicate how the pattern tiles are placed and spaced. These attributes represent coordinates and values in the coordinate space specified by the combination of attributes **patternUnits** and **patternTransform**.

If the attribute is not specified, the effect is as if a value of zero were specified.

*Animatable: yes.*

**y = "<coordinate>"**

See x.

If the attribute is not specified, the effect is as if a value of zero were specified.

*Animatable: yes.*

**`width` = "<length>"**

See x.

A negative value is an error (see Error processing). A value of zero disables rendering of the element (i.e., no paint is applied).
If the attribute is not specified, the effect is as if a value of zero were specified.
*Animatable: yes.*

**`height` = "<length>"**

See x.
A negative value is an error (see Error processing). A value of zero disables rendering of the element (i.e., no paint is applied).
If the attribute is not specified, the effect is as if a value of zero were specified.
*Animatable: yes.*

**`xlink:href` = "<uri>"**

A URI reference to a different **'pattern'** element within the current SVG document fragment. Any attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no children, and the referenced element does (possibly due to its own **href** attribute), then this element inherits the children from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attributes or children due to its own **href** attribute, then the current element can inherit those attributes or children.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, %testAttrs;, externalResourcesRequired, viewBox, preserveAspectRatio, %xlinkRefAttrs;, style, %PresentationAttributes-All;.**

SVG's user agent style sheet sets the **'overflow'** property for **'pattern'** elements to **hidden**, which causes a rectangular clipping path to be created at the bounds of the pattern tile. Unless the **'overflow'** property is overridden, any graphics within the pattern which goes outside of the pattern rectangle will be clipped. Example pattern01 below shows the effect of clipping to the pattern tile.

The contents of the **'pattern'** are relative to a new coordinate system. If there is a **viewBox** attribute, then the new coordinate system is fitted into the region defined by the **x**, **y**, **width**, **height** and **patternUnits** attributes on the **'pattern'** element using the standard rules for **viewBox** and **preserveAspectRatio**. If there is no **viewBox** attribute, then the new coordinate system has its origin at (x,y), where x is established by the **x** attribute on the **'pattern'** element, and y is established by the **y** attribute on the **'pattern'** element. Thus, in the following example:

```
<pattern x="10" y="10" width="20" height="20">
   <rect x="5" y="5" width="10" height="10"/>
</pattern>
```

the rectangle has its top/left located 5 units to the right and 5 units down from the origin of the pattern tile.

The **viewBox** attribute introduces a supplemental transformation which is applied on top of any transformations necessary to create a new pattern coordinate system due to attributes **x**, **y**, **width**, **height** and **patternUnits**.

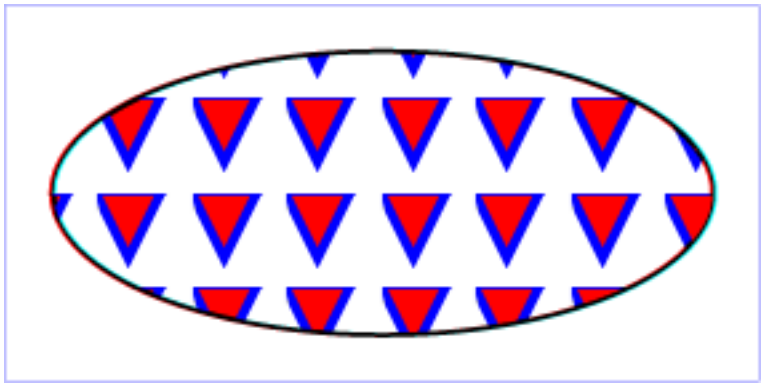
Properties inherit into the **'pattern'** element from its ancestors; properties do *not* inherit from the element referencing the **'pattern'** element.

**'pattern'** elements are never rendered directly; their only usage is as something that can be referenced using the **'fill'** and **'stroke'** properties. The **'display'** property does not apply to the **'pattern'** element; thus, **'pattern'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'pattern'** elements are available for referencing even when the **'display'** property on the **'pattern'** element or any of its ancestors is set to **none**.

Event attributes and event listeners attached to the contents of a **'pattern'** element are not processed; only the rendering aspects of **'pattern'** elements are processed.

Example pattern01 shows how to fill a rectangle by referencing a pattern paint server. Note how the blue stroke of each triangle has been clipped at the top and the left. This is due to SVG's user agent style sheet setting the **'overflow'** property for **'pattern'** elements to **hidden**, which causes the pattern to be clipped to the bounds of the pattern tile.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <pattern id="TrianglePattern" patternUnits="userSpaceOnUse"
             x="0" y="0" width="100" height="100"
             viewBox="0 0 10 10" >
      <path d="M 0 0 L 7 0 L 3.5 7 z" fill="red" stroke="blue" />
    </pattern>
  </defs>

  <!-- Outline the drawing area in blue -->
  <rect fill="none" stroke="blue"
        x="1" y="1" width="798" height="398"/>

  <!-- The ellipse is filled using a triangle pattern paint server
       and stroked with black -->
  <ellipse fill="url(#TrianglePattern)" stroke="black" stroke-width="5"
           cx="400" cy="200" rx="350" ry="150" />
</svg>
```



*Example pattern01*

*View this example as SVG (SVG-enabled browsers only)*

## 13.4 DOM interfaces

The following interfaces are defined below: SVGGradientElement, SVGLinearGradientElement, SVGRadialGradientElement, SVGStopElement, SVGPatternElement.

### Interface SVGGradientElement

The **SVGGradientElement** interface is a base interface used by **SVGLinearGradientElement** and **SVGRadialGradientElement**.

**IDL Definition**

```
interface SVGGradientElement :
               SVGElement,
               SVGURIReference,
               SVGExternalResourcesRequired,
               SVGStylable,
               SVGUnitTypes {

  // Spread Method Types
  const unsigned short SVG_SPREADMETHOD_UNKNOWN = 0;
  const unsigned short SVG_SPREADMETHOD_PAD     = 1;
  const unsigned short SVG_SPREADMETHOD_REFLECT = 2;
  const unsigned short SVG_SPREADMETHOD_REPEAT  = 3;

  readonly attribute SVGAnimatedEnumeration   gradientUnits;
  readonly attribute SVGAnimatedTransformList gradientTransform;
  readonly attribute SVGAnimatedEnumeration   spreadMethod;
};
```

**Definition group Spread Method Types**

**Defined constants**

| | |
|---|---|
| SVG_SPREADMETHOD_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_SPREADMETHOD_PAD | Corresponds to value **pad**. |
| SVG_SPREADMETHOD_REFLECT | Corresponds to value **reflect**. |
| SVG_SPREADMETHOD_REPEAT | Corresponds to value **repeat**. |

**Attributes**

**readonly SVGAnimatedEnumeration gradientUnits**
Corresponds to attribute **gradientUnits** on the given element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedTransformList gradientTransform**
Corresponds to attribute **gradientTransform** on the given element.

**readonly SVGAnimatedEnumeration spreadMethod**
Corresponds to attribute **spreadMethod** on the given element. One of the Spread Method Types.

## Interface SVGLinearGradientElement

The **SVGLinearGradientElement** interface corresponds to the **'linearGradient'** element.

**IDL Definition**

```
interface SVGLinearGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};
```

**Attributes**

**readonly SVGAnimatedLength x1**
Corresponds to attribute **x1** on the given **'linearGradient'** element.

**readonly SVGAnimatedLength y1**
Corresponds to attribute **y1** on the given **'linearGradient'** element.

**readonly SVGAnimatedLength x2**
Corresponds to attribute **x2** on the given **'linearGradient'** element.

**readonly SVGAnimatedLength y2**
Corresponds to attribute **y2** on the given **'linearGradient'** element.

## Interface SVGRadialGradientElement

The **SVGRadialGradientElement** interface corresponds to the **'radialGradient'** element.

### IDL Definition

```
interface SVGRadialGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
  readonly attribute SVGAnimatedLength fx;
  readonly attribute SVGAnimatedLength fy;
};
```

### Attributes

**readonly SVGAnimatedLength cx**
: Corresponds to attribute **cx** on the given **'radialGradient'** element.

**readonly SVGAnimatedLength cy**
: Corresponds to attribute **cy** on the given **'radialGradient'** element.

**readonly SVGAnimatedLength r**
: Corresponds to attribute **r** on the given **'radialGradient'** element.

**readonly SVGAnimatedLength fx**
: Corresponds to attribute **fx** on the given **'radialGradient'** element.

**readonly SVGAnimatedLength fy**
: Corresponds to attribute **fy** on the given **'radialGradient'** element.

## Interface SVGStopElement

The **SVGStopElement** interface corresponds to the **'stop'** element.

### IDL Definition

```
interface SVGStopElement :
               SVGElement,
               SVGStylable {

  readonly attribute SVGAnimatedNumber offset;
};
```

### Attributes

**readonly SVGAnimatedNumber offset**

Corresponds to attribute **offset** on the given **'stop'** element.

## Interface SVGPatternElement

The **SVGPatternElement** interface corresponds to the **'pattern'** element.

### IDL Definition

```
interface SVGPatternElement :
               SVGElement,
               SVGURIReference,
               SVGTests,
               SVGLangSpace,
               SVGExternalResourcesRequired,
               SVGStylable,
               SVGFitToViewBox,
               SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration   patternUnits;
  readonly attribute SVGAnimatedEnumeration   patternContentUnits;
  readonly attribute SVGAnimatedTransformList patternTransform;
  readonly attribute SVGAnimatedLength        x;
  readonly attribute SVGAnimatedLength        y;
  readonly attribute SVGAnimatedLength        width;
  readonly attribute SVGAnimatedLength        height;
};
```

### Attributes

**readonly SVGAnimatedEnumeration patternUnits**
: Corresponds to attribute **patternUnits** on the given **'pattern'** element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedEnumeration patternContentUnits**
: Corresponds to attribute **patternContentUnits** on the given **'pattern'** element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedTransformList patternTransform**
: Corresponds to attribute **patternTransform** on the given **'pattern'** element.

**readonly SVGAnimatedLength x**
: Corresponds to attribute **x** on the given **'pattern'** element.

**readonly SVGAnimatedLength y**
: Corresponds to attribute **y** on the given **'pattern'** element.

**readonly SVGAnimatedLength width**
: Corresponds to attribute **width** on the given **'pattern'** element.

**readonly SVGAnimatedLength height**
: Corresponds to attribute **height** on the given **'pattern'** element.

---

04 September 2001

# 14 Clipping, Masking and Compositing

## Contents



## 14.1 Introduction

SVG supports the following clipping/masking features:

- clipping paths, which uses any combination of **'path'**, **'text'** and basic shapes to serve as the outline of a (in the absence of anti-aliasing) 1-bit mask, where everything on the "inside" of the outline is allowed to show through but everything on the outside is masked out
- masks, which are container elements which can contain graphics elements or other container elements which define a set of graphics that is to be used as a semi-transparent mask for compositing foreground objects into the current background.

One key distinction between a clipping path and a mask is that clipping paths are hard masks (i.e., the silhouette consists of either fully opaque pixels or fully transparent pixels, with the possible exception of anti-aliasing along the edge of the silhouette) whereas masks consist of an image where each pixel value indicates the degree of transparency vs. opacity. In a mask, each pixel value can range from fully transparent to fully opaque.

SVG supports only simple alpha blending compositing (see Simple Alpha Compositing).

## 14.2 Simple alpha compositing

Graphics elements are blended into the elements already rendered on the canvas using simple alpha compositing, in which the resulting color and opacity at any given pixel on the canvas is the result of the following formulas (all color values use premultiplied alpha):

```
Er, Eg, Eb    - Element color value
Ea            - Element alpha value
Cr, Cg, Cb    - Canvas color value (before blending)
Ca            - Canvas alpha value (before blending)
Cr', Cg', Cb' - Canvas color value (after blending)
Ca'           - Canvas alpha value (after blending)

Ca' = 1 - (1 - Ea) * (1 - Ca)
Cr' = (1 - Ea) * Cr + Er
Cg' = (1 - Ea) * Cg + Eg
Cb' = (1 - Ea) * Cb + Eb
```

The following rendering properties, which provide information about the color space in which to perform the compositing operations, apply to compositing operations:

- **'color-interpolation'**
- **'color-rendering'**

## 14.3 Clipping paths

### 14.3.1 Introduction

The clipping path restricts the region to which paint can be applied. Conceptually, any parts of the drawing that lie outside of the region bounded by the currently active clipping path are not drawn. A clipping path can be thought of as a mask wherein those pixels outside the clipping path are black with an alpha value of zero and those pixels inside the clipping path are white with an alpha value of one (with the possible exception of anti-aliasing along the edge of the silhouette).

### 14.3.2 The initial clipping path

When an **'svg'** element is either the root element in the document or is embedded within a

document whose layout is determined according to the layout rules of CSS or XSL, then the user agent must establish an initial clipping path for the SVG document fragment. The **'overflow'** and **'clip'** properties along with additional SVG user agent processing rules determine the initial clipping path which the user agent establishes for the SVG document fragment:

### 14.3.3 The 'overflow' and 'clip' properties

**'overflow'**

| | |
|---|---|
| *Value:* | visible \| hidden \| scroll \| auto \| inherit |
| *Initial:* | see prose |
| *Applies to:* | elements which establish a new viewport, **'pattern'** elements and **'marker'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'overflow'** property has the same parameter values and has the same meaning as defined in [CSS2-overflow]; however, the following additional points apply:

- The **'overflow'** property applies to elements that establish new viewports (e.g., **'svg'** elements), **'pattern'** elements and **'marker'** elements. For all other elements, the property has no effect (i.e., a clipping rectangle is not created).
- For those elements to which the **'overflow'** property can apply, if the **'overflow'** property has the value **hidden** or **scroll**, the effect is that a new clipping path in the shape of a rectangle is created. The result is equivalent to defining a **'clipPath'** element whose content is a **'rect'** element which defines the equivalent rectangle, and then specifying the <uri> of this **'clipPath'** element on the **'clip-path'** property for the given element.
- If the **'overflow'** property has a value other than **hidden** or **scroll**, the property has no effect (i.e., a clipping rectangle is not created).
- Within SVG content, the value **auto** is equivalent to the value **visible**.
- When an outermost **'svg'** element is embedded inline within a parent XML grammar which uses CSS layout [CSS2-LAYOUT] or XSL formatting [XSL], if the **'overflow'** property has the value **hidden** or **scroll**, then the user agent will establish an initial clipping path equal to the bounds of the initial viewport; otherwise, the initial clipping path is set according to the clipping rules as defined in [CSS2-overflow].
- When an outermost SVG **'svg'** element is stand-alone or embedded inline within a parent XML grammar which does not use CSS layout [CSS2-LAYOUT] or XSL formatting [XSL], the **'overflow'** property on the outermost **'svg'** element is ignored for the purposes of visual rendering and the initial clipping path is set to the bounds of the initial viewport.
- The initial value for **'overflow'** as defined in [CSS2-overflow] is 'visible'; however, SVG's

user agent style sheet overrides this initial value and set the **'overflow'** property on elements that establish new viewports (e.g., **'svg'** elements), **'pattern'** elements and **'marker'** elements to the value 'hidden'.

As a result of the above, the default behavior of SVG user agents is to establish a clipping path to the bounds of the initial viewport and to establish a new clipping path for each element which establishes a new viewport and each **'pattern'** and **'marker'** element.

For related information, see Clip to viewport vs. clip to **viewBox**.

**'clip'**

| | |
|---|---|
| *Value:* | <shape> \| auto \| inherit |
| *Initial:* | auto |
| *Applies to:* | elements which establish a new viewport, **'pattern'** elements and **'marker'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'clip'** property has the same parameter values as defined in [CSS2-clip]. Unitless values, which indicate current user coordinates, are permitted on the coordinate values on the <shape>. The value of "auto" defines a clipping path along the bounds of the viewport created by the given element.

## 14.3.4 Clip to viewport vs. clip to viewBox

It is important to note that initial values for the **'overflow'** and **'clip'** properties and the user agent style sheet will result in an initial clipping path that is set to the bounds of the initial viewport. When attributes **viewBox** and **preserveAspectRatio** attributes are specified, it is sometime desirable that the clipping path be set to the bounds of the **viewBox** instead of the viewport (or reference rectangle, in the case of **'marker'** and **'pattern'** elements), particularly when **preserveAspectRatio** specifies uniform scaling and the aspect ratio of the **viewBox** does not match the aspect ratio of the viewport.

To set the initial clipping path to the bounds of the **viewBox**, set the bounds of **'clip'** property to the same rectangle as specified on the **viewBox** attribute. (Note that the parameters do not match. **'clip'** takes values <top>, <right>,<bottom> and <left>, whereas **viewBox** takes values <min-x>, <min-y>, <width> and <height>.)

## 14.3.5 Establishing a new clipping path

A clipping path is defined with a **'clipPath'** element. A clipping path is used/referenced using the **'clip-path'** property.

A **'clipPath'** element can contain **'path'** elements, **'text'** elements, basic shapes (such as **'circle'**) or a **'use'** element. If a **'use'** element is a child of a **'clipPath'** element, it must directly reference **'path'**, **'text'** or basic shape elements. Indirect references are an error (see Error processing).

The raw geometry of each child element exclusive of rendering properties such as **'fill'**, **'stroke'**, **'stroke-width'** within a **'clipPath'** conceptually defines a 1-bit mask (with the possible exception of anti-aliasing along the edge of the geometry) which represents the silhouette of the graphics associated with that element. Anything outside the outline of the object is masked out. When the **'clipPath'** element contains multiple child elements, the silhouettes of the child elements are logically OR'd together to create a single silhouette which is then used to restrict the region onto which paint can be applied. Thus, a point is inside the clipping path if it is inside any of the children of the **'clipPath'**.

It is an error if the **'clip-path'** property references a non-existent object or if the referenced object is not a **'clipPath'** element (see Error processing).

For a given graphics element, the actual clipping path used will be the intersection of the clipping path specified by its **'clip-path'** property (if any) with any clipping paths on its ancestors, as specified by the **'clip-path'** property on the ancestor elements, or by the **'overflow'** property on ancestor elements which establish a new viewport. Also, see the discussion of the initial clipping path.)

A couple of notes:

- The **'clipPath'** element itself and its child elements do *not* inherit clipping paths from the ancestors of the **'clipPath'** element.
- The **'clipPath'** element or any of its children can specify property **'clip-path'**.
  If a valid **'clip-path'** reference is placed on a **'clipPath'** element, the resulting clipping path is the intersection of the contents of the **'clipPath'** element with the referenced clipping path.
  If a valid **'clip-path'** reference is placed on one of the children of a **'clipPath'** element, then the given child element is clipped by the referenced clipping path before OR'ing the silhouette of the child element with the silhouettes of the other child elements.

```
<!ENTITY % clipPathExt "" >
<!ELEMENT clipPath (%descTitleMetadata;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%clipPathExt;)*) >
<!ATTLIST clipPath
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %PresentationAttributes-TextElements;
  transform %TransformList; #IMPLIED
  clipPathUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED >
```

*Attribute definitions:*

**`clipPathUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for the contents of the **'clipPath'**.
If **clipPathUnits="userSpaceOnUse"**, the contents of the **'clipPath'** represent values in the current user coordinate system in place at the time when the **'clipPath'** element is referenced (i.e., the user coordinate system for the element referencing the **'clipPath'** element via the **'clip-path'** property).
If **clipPathUnits="objectBoundingBox"**, then the user coordinate system for the contents of the 'clipPath' element is established using the bounding box of the element to which the clipping path is applied (see Object bounding box units).
If attribute **clipPathUnits** is not specified, then the effect is as if a value of **userSpaceOnUse** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %testAttrs;, %langSpaceAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-Color;, %PresentationAttributes-FillStroke;, %PresentationAttributes-FontSpecification;, %PresentationAttributes-Graphics;, %PresentationAttributes-TextContentElements;, %PresentationAttributes-TextElements;, transform.**

Properties inherit into the **'clipPath'** element from its ancestors; properties do *not* inherit from the element referencing the **'clipPath'** element.

**'clipPath'** elements are never rendered directly; their only usage is as something that can be referenced using the **'clip-path'** property. The **'display'** property does not apply to the **'clipPath'** element; thus, **'clipPath'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'clipPath'** elements are available for referencing even when the **'display'** property on the **'clipPath'** element or any of its ancestors is set to **none**.

**'clip-path'**

| | |
|---|---|
| *Value:* | <uri> \| none \| inherit |
| *Initial:* | none |
| *Applies to:* | container elements and graphics elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**<uri>**

A URI reference to another graphical object within the same SVG document fragment which will be used as the clipping path.

**'clip-rule'**

| | |
|---|---|
| *Value:* | nonzero \| evenodd \| inherit |
| *Initial:* | nonzero |
| *Applies to:* | graphics elements within a **'clipPath'** element |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**nonzero**

See description of **'fill-rule'** property.

**evenodd**

See description of **'fill-rule'** property.

The **'clip-rule'** property only applies to graphics elements that are contained within a **'clipPath'** element. The following fragment of code will cause an evenodd clipping rule to be applied to the clipping path because **'clip-rule'** is specified on the **'path'** element that defines the clipping shape:

```
<g clip-rule="nonzero">
  <clipPath id="MyClip">
    <path d="..." clip-rule="evenodd" />
  </clipPath>
  <rect clip-path="url(#MyClip)" ... />
</g>
```

whereas the following fragment of code will *not* cause an evenodd clipping rule to be applied because the **'clip-rule'** is specified on the referencing element, not on the object defining the clipping shape:

```
<g clip-rule="nonzero">
  <clipPath id="MyClip">
    <path d="..." />
  </clipPath>
  <rect clip-path="url(#MyClip)" clip-rule="evenodd" ... />
</g>
```

## 14.4 Masking

In SVG, you can specify that any other graphics object or **'g'** element can be used as an alpha mask for compositing the current object into the background.

A mask is defined with a **'mask'** element. A mask is used/referenced using the **'mask'** property.

A **'mask'** can contain any graphical elements or container elements such as a 'g'.

It is an error if the **'mask'** property references a non-existent object or if the referenced object is not a **'mask'** element (see Error Processing).

The effect is as if the child elements of the **'mask'** are rendered into an offscreen image which has been initialized to transparent black. Any graphical object which uses/references the given **'mask'** element will be painted onto the background through the mask, thus completely or partially masking out parts of the graphical object.

For any graphics object that is used as a mask, the mask value at any point is computed from the color channel values and alpha channel value as follows. A linear luminance value is computed from the color channel values. This can be done, for example, by first converting the original image color values (potentially in the sRGB color space) to the linear RGB color space (see Rendering properties). Then, using non-premultiplied linear RGB color values, apply the luminance-to-alpha coefficients (as defined in the **'feColorMatrix'** filter primitive) to convert the linear RGB color values to linear luminance values. If the graphics object also includes an alpha channel, then the computed linear luminance value is multiplied by the corresponding alpha value to produce the mask value.

For a four-channel RGBA graphics object that is used as a mask, both the color channels and the alpha channel of the mask contribute to the masking operation. The alpha mask that is used to composite the current object into the background represents the product of the luminance of the color channels (see previous paragraph) and the alpha channel from the mask.

For a three-channel RGB graphics object that is used as a mask (e.g., when referencing a 3-channel image file), the effect is as if the object were converted into a 4-channel RGBA image with the alpha channel uniformly set to 1.

For a single-channel image that is used as a mask (e.g., when referencing a 1-channel grayscale image file), the effect is as if the object were converted into a 4-channel RGBA image, where the single channel from the referenced object is used to compute the three color channels and the alpha channel is uniformly set to 1. Note that when referencing a grayscale image file, the transfer curve relating the encoded grayscale values to linear light values must be taken into account when computing the color channels.

The effect of a mask is identical to what would have happened if there were no mask but instead the alpha channel of the given object were multiplied with the mask's resulting alpha values (i.e., the product of the mask's luminance from its color channels multiplied by the mask's alpha channel).

Note that SVG 'path''s, shapes (e.g., 'circle') and 'text' are all treated as four-channel RGBA images for the purposes of masking operations.

Example mask01 uses an image to mask a rectangle.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="3cm" viewBox="0 0 800 300"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example mask01 - blue text masked with gradient against red background
  </desc>
  <defs>
    <linearGradient id="Gradient" gradientUnits="userSpaceOnUse"
                    x1="0" y1="0" x2="800" y2="0">
      <stop offset="0" stop-color="white" stop-opacity="0" />
      <stop offset="1" stop-color="white" stop-opacity="1" />
    </linearGradient>
    <mask id="Mask" maskUnits="userSpaceOnUse"
          x="0" y="0" width="800" height="300">
      <rect x="0" y="0" width="800" height="300" fill="url(#Gradient)"  />
    </mask>
    <text id="Text" x="400" y="200"
          font-family="Verdana" font-size="100" text-anchor="middle" >
      Masked text
    </text>
  </defs>

  <!-- Draw a pale red rectangle in the background -->
  <rect x="0" y="0" width="800" height="300" fill="#FF8080" />

  <!-- Draw the text string twice. First, filled blue, with the mask applied.
       Second, outlined in black without the mask. -->
  <use xlink:href="#Text" fill="blue" mask="url(#Mask)" />
  <use xlink:href="#Text" fill="none" stroke="black" stroke-width="2" />
</svg>
```


***Example mask01***

*View this example as SVG (SVG-enabled browsers only)*

```
<!ENTITY % maskExt "" >
<!ELEMENT mask (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%maskExt;)*  >
<!ATTLIST mask
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  maskUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  maskContentUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >
```

*Attribute definitions:*

**`maskUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for attributes **x, y, width, height**.
If **maskUnits="userSpaceOnUse"**, **x, y, width, height** represent values in the current user coordinate system in place at the time when the **'mask'** element is referenced (i.e., the user coordinate system for the element referencing the **'mask'** element via the **'mask'** property).
If **maskUnits="objectBoundingBox"**, **x, y, width, height** represent fractions or percentages of the bounding box of the element to which the mask is applied. (See

Object bounding box units.)

If attribute **maskUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.

*Animatable: yes.*

**`maskContentUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**

Defines the coordinate system for the contents of the **'mask'**.

If **maskContentUnits="userSpaceOnUse"**, the user coordinate system for the contents of the **'mask'** element is the current user coordinate system in place at the time when the **'mask'** element is referenced (i.e., the user coordinate system for the element referencing the **'mask'** element via the **'mask'** property).

If **maskContentUnits="objectBoundingBox"**, the user coordinate system for the contents of the **'mask'** is established using the bounding box of the element to which the mask is applied. (See Object bounding box units.)

If attribute **maskContentUnits** is not specified, then the effect is as if a value of **userSpaceOnUse** were specified.

*Animatable: yes.*

**`x` = "<coordinate>"**

The x-axis coordinate of one corner of the rectangle for the largest possible offscreen buffer. Note that the clipping path used to render any graphics within the mask will consist of the intersection of the current clipping path associated with the given object and the rectangle defined by **x, y, width, height**.

If the attribute is not specified, the effect is as if a value of "-10%" were specified.

*Animatable: yes.*

**`y` = "<coordinate>"**

The y-axis coordinate of one corner of the rectangle for the largest possible offscreen buffer.

If the attribute is not specified, the effect is as if a value of "-10%" were specified.

*Animatable: yes.*

**`width` = "<length>"**

The width of the largest possible offscreen buffer. Note that the clipping path used to render any graphics within the mask will consist of the intersection of the current clipping path associated with the given object and the rectangle defined by **x, y, width, height**.

A negative value is an error (see Error processing). A value of zero disables rendering of the element.

If the attribute is not specified, the effect is as if a value of "120%" were specified.

*Animatable: yes.*

**`height` = "<length>"**

The height of the largest possible offscreen buffer.

A negative value is an error (see Error processing). A value of zero disables rendering of the element.

If the attribute is not specified, the effect is as if a value of "120%" were specified.

*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %testAttrs;, %langSpaceAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-All;.**

Properties inherit into the **'mask'** element from its ancestors; properties do *not* inherit from the element referencing the **'mask'** element.

**'mask'** elements are never rendered directly; their only usage is as something that can be referenced using the **'mask'** property. The **'display'** property does not apply to the **'mask'** element; thus, **'mask'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'mask'** elements are available for referencing even when the **'display'** property on the **'mask'** element or any of its ancestors is set to **none**.

The following is a description of the **'mask'** property.

**'mask'**

| | |
|---|---|
| *Value:* | <uri> \| none \| inherit |
| *Initial:* | none |
| *Applies to:* | container elements and graphics elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**<uri>**

A URI reference to another graphical object which will be used as the mask.

## 14.5 Object and group opacity: the 'opacity' property

There are several opacity properties within SVG:

- Fill opacity
- Stroke opacity
- Gradient stop opacity
- Object/group opacity (described here)

Except for object/group opacity (described just below), all other opacity properties are involved in intermediate rendering operations. Object/group opacity can be thought of conceptually as a postprocessing operation. Conceptually, after the object/group is rendered into an RGBA offscreen image, the object/group opacity setting specifies how to blend the offscreen image into the current background.

**'opacity'**

| | |
|---|---|
| *Value:* | <alphavalue> \| inherit |
| *Initial:* | 1 |
| *Applies to:* | container elements and graphics elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |

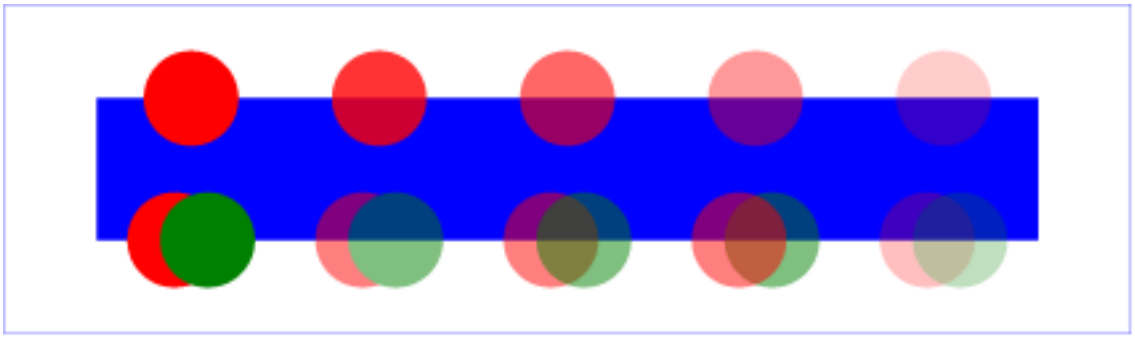
*Animatable*: yes

**<alphavalue>**

The uniform opacity setting to be applied across an entire object. Any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. (See Clamping values which are restricted to a particular range.) If the object is a container element such as a **'g'**, then the effect is as if the contents of the **'g'** were blended against the current background using a mask where the value of each pixel of the mask is <alphavalue>. (See Simple alpha compositing.)

Example opacity01, illustrates various usage of the **'opacity'** property on elements and groups.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="3.5cm" viewBox="0 0 1200 350"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example opacity01 - opacity property</desc>

  <rect x="1" y="1" width="1198" height="348"
        fill="none" stroke="blue" />

  <!-- Background blue rectangle -->
  <rect x="100" y="100" width="1000" height="150" fill="#0000ff"  />

  <!-- Red circles going from opaque to nearly transparent -->
  <circle cx="200" cy="100" r="50" fill="red" opacity="1"  />
  <circle cx="400" cy="100" r="50" fill="red" opacity=".8"  />
  <circle cx="600" cy="100" r="50" fill="red" opacity=".6"  />
  <circle cx="800" cy="100" r="50" fill="red" opacity=".4"  />
  <circle cx="1000" cy="100" r="50" fill="red" opacity=".2"  />

  <!-- Opaque group, opaque circles -->
  <g opacity="1" >
    <circle cx="182.5" cy="250" r="50" fill="red" opacity="1"  />
    <circle cx="217.5" cy="250" r="50" fill="green" opacity="1"  />
  </g>
  <!-- Group opacity: .5, opacity circles -->
  <g opacity=".5" >
    <circle cx="382.5" cy="250" r="50" fill="red" opacity="1"  />
    <circle cx="417.5" cy="250" r="50" fill="green" opacity="1"  />
  </g>
  <!-- Opaque group, semi-transparent green over red -->
  <g opacity="1" >
    <circle cx="582.5" cy="250" r="50" fill="red" opacity=".5"  />
    <circle cx="617.5" cy="250" r="50" fill="green" opacity=".5"  />
  </g>
  <!-- Opaque group, semi-transparent red over green -->
  <g opacity="1" >
    <circle cx="817.5" cy="250" r="50" fill="green" opacity=".5"  />
    <circle cx="782.5" cy="250" r="50" fill="red" opacity=".5"  />
  </g>
  <!-- Group opacity .5, semi-transparent green over red -->
  <g opacity=".5" >
    <circle cx="982.5" cy="250" r="50" fill="red" opacity=".5"  />
    <circle cx="1017.5" cy="250" r="50" fill="green" opacity=".5"  />
  </g>
</svg>
```



*Example opacity01*

*View this example as SVG (SVG-enabled browsers only)*

In the example above, the top row of circles have differing opacities, ranging from 1.0 to 0.2. The bottom row illustrates five **'g'** elements, each of which contains overlapping red and green circles, as follows:

- The first group shows the opaque case for reference. The group has opacity of 1, as do the circles.
- The second group shows group opacity when the elements in the group are opaque.
- The third and fourth group show that opacity is not commutative. In the third group (which has opacity of 1), a semi-transparent green circle is drawn on top of a semi-transparent red circle, whereas in the fourth group a semi-transparent red circle is drawn on top of a semi-transparent green circle. Note that area where the two circles intersect display different colors. The third group shows more green color in the intersection area, whereas the fourth group shows more red color.
- The fifth group shows the multiplicative effect of opacity settings. Both the circles and the group itself have opacity settings of .5. The result is that the portion of the red circle which does not overlap with the green circle (i.e., the top/right of the red circle) will blend into the blue rectangle with accumulative opacity of .25 (i.e., .5*.5), which, after blending into the blue rectangle, results in a blended color which is 25% red and 75% blue.

## 14.6 DOM interfaces

The following interfaces are defined below: SVGClipPathElement, SVGMaskElement.

### Interface SVGClipPathElement

The **SVGClipPathElement** interface corresponds to the **'clipPath'** element.

**IDL Definition**

```
interface SVGClipPathElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration clipPathUnits;
};
```

**Attributes**

**readonly SVGAnimatedEnumeration clipPathUnits**

Corresponds to attribute **clipPathUnits** on the given **'clipPath'** element. Takes one of the constants defined in **SVGUnitTypes**.

## Interface SVGMaskElement

The **SVGMaskElement** interface corresponds to the **'mask'** element.

**IDL Definition**

```
interface SVGMaskElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration maskUnits;
  readonly attribute SVGAnimatedEnumeration maskContentUnits;
  readonly attribute SVGAnimatedLength      x;
  readonly attribute SVGAnimatedLength      y;
  readonly attribute SVGAnimatedLength      width;
  readonly attribute SVGAnimatedLength      height;
};
```

**Attributes**

**readonly SVGAnimatedEnumeration maskUnits**

Corresponds to attribute **maskUnits** on the given **'mask'** element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedEnumeration maskContentUnits**

Corresponds to attribute **maskContentUnits** on the given **'mask'** element.

Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedLength x**
Corresponds to attribute **x** on the given **'mask'** element.

**readonly SVGAnimatedLength y**
Corresponds to attribute **y** on the given **'mask'** element.

**readonly SVGAnimatedLength width**
Corresponds to attribute **width** on the given **'mask'** element.

**readonly SVGAnimatedLength height**
Corresponds to attribute **height** on the given **'mask'** element.

---

04 September 2001

# 15 Filter Effects

## Contents

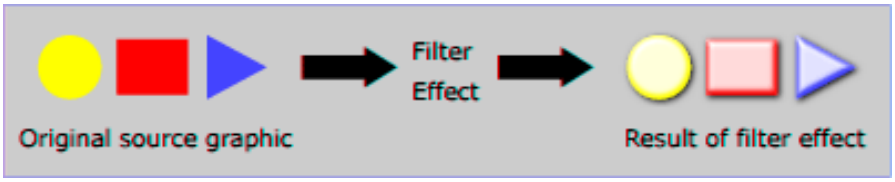


## 15.1 Introduction

This chapter describes SVG's declarative filter effects feature set, which when combined with the 2D power of SVG can describe much of the common artwork on the Web in such a way that client-side generation and alteration can be performed easily. In addition, the ability to apply filter effects to SVG graphics elements and container elements helps to

maintain the semantic structure of the document, instead of resorting to images which aside from generally being a fixed resolution tend to obscure the original semantics of the elements they replace. This is especially true for effects applied to text.

A filter effect consists of a series of graphics operations that are applied to a given **source graphic** to produce a modified graphical result. The result of the filter effect is rendered to the target device instead of the original source graphic. The following illustrates the process:




*View this example as SVG (SVG-enabled browsers only)*

Filter effects are defined by **'filter'** elements. To apply a filter effect to a graphics element or a container element, you set the value of the **'filter'** property on the given element such that it references the filter effect.

Each **'filter'** element contains a set of **filter primitives** as its children. Each filter primitive performs a single fundamental graphical operation (e.g., a blur or a lighting effect) on one or more inputs, producing a graphical result. Because most of the filter primitives represent some form of image processing, in most cases the output from a filter primitive is a single RGBA image.

The original source graphic or the result from a filter primitive can be used as input into one or more other filter primitives. A common application is to use the source graphic multiple times. For example, a simple filter could replace one graphic by two by adding a black copy of original source graphic offset to create a drop shadow. In effect, there are now two layers of graphics, both with the same original source graphics.

When applied to container elements such as **'g'**, the **'filter'** property applies to the contents of the group as a whole. The group's children do not render to the screen directly; instead, the graphics commands necessary to render the children are stored temporarily. Typically, the graphics commands are executed as part of the processing of the referenced **'filter'** element via use of the keywords **SourceGraphic** or **SourceAlpha**. Filter effects can be applied to container elements with no content (e.g., an empty **'g'** element), in which case the **SourceGraphic** or **SourceAlpha** consist of a transparent black rectangle that is the size of the **filter effects region**.

Sometimes filter primitives result in undefined pixels. For example, filter primitive **'feOffset'** can shift an image down and to the right, leaving undefined pixels at the top and left. In these cases, the undefined pixels are set to transparent black.

## 15.2 An example

The following shows an example of a filter effect.

Example filters01 - introducing filter effects.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
              "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="7.5cm" height="5cm" viewBox="0 0 200 120"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example filters01.svg - introducing filter effects</title>
  <desc>An example which combines multiple filter primitives
        to produce a 3D lighting effect on a graphic consisting
        of the string "SVG" sitting on top of oval filled in red
```

```
      and surrounded by an oval outlined in red.</desc>
  <defs>
    <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120">
      <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
      <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
      <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
                          specularExponent="20" lighting-color="#bbbbbb"
                          result="specOut">
        <fePointLight x="-5000" y="-10000" z="20000"/>
      </feSpecularLighting>
      <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
      <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
                   k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
      <feMerge>
        <feMergeNode in="offsetBlur"/>
        <feMergeNode in="litPaint"/>
      </feMerge>
    </filter>
  </defs>
  <rect x="1" y="1" width="198" height="118" fill="#888888" stroke="blue" />
  <g filter="url(#MyFilter)" >
          <g>
      <path fill="none" stroke="#D90000" stroke-width="10"
            d="M50,90 C0,90 0,30 50,30 L150,30 C200,30 200,90 150,90 z" />
      <path fill="#D90000"
            d="M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z" />
      <g fill="#FFFFFF" stroke="black" font-size="45" font-family="Verdana" >
        <text x="52" y="76">SVG</text>
      </g>
    </g>
  </g>
</svg>
```


***Example filters01***

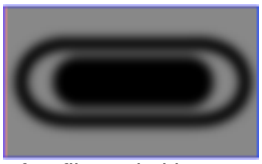
*View this example as SVG (SVG-enabled browsers only)*

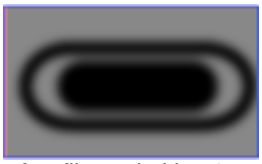
The filter effect used in the example above is repeated here with reference numbers in the left column before each of the six filter primitives:

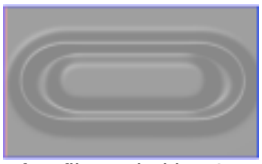
```
        <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120">
          <desc>Produces a 3D lighting effect.</desc>
1         <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
2         <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
3         <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
                              specularExponent="20" lighting-color="#bbbbbb"
                              result="specOut">
            <fePointLight x="-5000" y="-10000" z="20000"/>
          </feSpecularLighting>
4         <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
5         <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
                       k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
6         <feMerge>
            <feMergeNode in="offsetBlur"/>
            <feMergeNode in="litPaint"/>
          </feMerge>
        </filter>
```

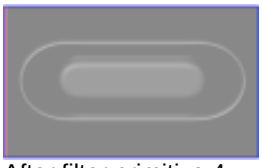
The following pictures show the intermediate image results from each of the six filter elements:

Source graphic


After filter primitive 1


After filter primitive 2


After filter primitive 3


After filter primitive 4

After filter primitive 5

After filter primitive 6

1. Filter primitive **'feGaussianBlur'** takes input **SourceAlpha**, which is the alpha channel of the source graphic. The result is stored in a temporary buffer named "blur". Note that "blur" is used as input to both filter primitives 2 and 3.
2. Filter primitive **'feOffset'** takes buffer "blur", shifts the result in a positive direction in both x and y, and creates a new buffer named "offsetBlur". The effect is that of a drop shadow.
3. Filter primitive **'feSpecularLighting'**, uses buffer "blur" as a model of a surface elevation and generates a lighting effect from a single point source. The result is stored in buffer "specOut".
4. Filter primitive **'feComposite'** masks out the result of filter primitive 3 by the original source graphics alpha channel so that the intermediate result is no bigger than the original source graphic.
5. Filter primitive **'feComposite'** composites the result of the specular lighting with the original source graphic.
6. Filter primitive **'feMerge'** composites two layers together. The lower layer consists of the drop shadow result from filter primitive 2. The upper layer consists of the specular lighting result from filter primitive 5.

## 15.3 The 'filter' element

The description of the **'filter'** element follows:

```
<!ENTITY % filterExt "" >
<!ELEMENT filter (%descTitleMetadata;,(feBlend|feFlood|
  feColorMatrix|feComponentTransfer|
  feComposite|feConvolveMatrix|feDiffuseLighting|feDisplacementMap|
  feGaussianBlur|feImage|feMerge|
  feMorphology|feOffset|feSpecularLighting|
  feTile|feTurbulence|
  animate|set
  %filterExt;)*) >
<!ATTLIST filter
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  filterUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  primitiveUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED
  filterRes %NumberOptionalNumber; #IMPLIED >
```

*Attribute definitions:*

**`filterUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**
: See Filter effects region.

**`primitiveUnits` = "*userSpaceOnUse* | *objectBoundingBox*"**
: Specifies the coordinate system for the various length values within the filter primitives and for the attributes that define the filter primitive subregion.

If **primitiveUnits="userSpaceOnUse"**, any length values within the filter definitions represent values in the current user coordinate system in place at the time when the **'filter'** element is referenced (i.e., the user coordinate system for the element referencing the **'filter'** element via a **'filter'** property).

If **primitiveUnits="objectBoundingBox"**, then any length values within the filter definitions represent fractions or percentages of the bounding box on the referencing element (see Object bounding box units).

If attribute **primitiveUnits** is not specified, then the effect is as if a value of **userSpaceOnUse** were specified.
*Animatable: yes.*

**`x` = "*<coordinate>*"**
: See Filter effects region.

**`y` = "*<coordinate>*"**
: See Filter effects region.

**`width` = "*<length>*"**
: See Filter effects region.

**`height` = "*<length>*"**
: See Filter effects region.

**`filterRes` = "*<number-optional-number>*"**
: See Filter effects region.

**`xlink:href` = "<uri>"**
: A URI reference to another **'filter'** element within the current SVG document fragment. Any attributes which are defined on the referenced **'filter'** element which are not defined on this element are inherited by this element. If this element has no defined filter nodes, and the referenced element has defined filter nodes (possibly due to its own **href** attribute), then this element inherits the filter nodes defined from the referenced **'filter'** element. Inheritance can be indirect to an arbitrary level; thus, if the referenced **'filter'** element inherits attributes or its filter node specification due to its own **href** attribute, then the current element can inherit those attributes or filter node specifications.
*Animatable: yes.*

***Attributes defined elsewhere:***
**%stdAttrs;, %langSpaceAttrs;, %xlinkRefAttrs; externalResourcesRequired, %PresentationAttributes-All;.**

Properties inherit into the **'filter'** element from its ancestors; properties do *not* inherit from the element referencing the **'filter'** element.

**'filter'** elements are never rendered directly; their only usage is as something that can be referenced using the **'filter'** property. The **'display'** property does not apply to the **'filter'** element; thus, **'filter'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'filter'** elements are available for referencing even when the **'display'** property on the **'filter'** element or any of its ancestors is set to **none**.

## 15.4 The 'filter' property

The description of the **'filter'** property is as follows:

**'filter'**

| | |
|---|---|
| *Value:* | <uri> \| none \| inherit |
| *Initial:* | none |
| *Applies to:* | container elements and graphics elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**<uri>**
A URI reference to a **'filter'** element which defines the filter effects that shall be applied to this element.
**none**
Do not apply any filter effects to this element.

## 15.5 Filter effects region

A **'filter'** element can define a region on the canvas to which a given filter effect applies and can provide a resolution for any intermediate continuous tone images used to process any raster-based filter primitives. The **'filter'** element has the following attributes which work together to define the filter effects region:

- **filterUnits={ userSpaceOnUse | objectBoundingBox }**.
  Defines the coordinate system for attributes **x, y, width, height**.
  If **filterUnits="userSpaceOnUse"**, **x, y, width, height** represent values in the current user coordinate system in place at the time when the **'filter'** element is referenced (i.e., the user coordinate system for the element referencing the **'filter'** element via a **'filter'** property).
  If **filterUnits="objectBoundingBox"**, then **x, y, width, height** represent fractions or percentages of the bounding box on the referencing element (see Object bounding box units).
  If attribute **filterUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.
  *Animatable: yes.*
- **x, y, width, height**, which define a rectangular region on the canvas to which this filter applies.
  The amount of memory and processing time required to apply the filter are related to the size of this rectangle and the **filterRes** attribute of the filter.
  The coordinate system for these attributes depends on the value for attribute **filterUnits**.
  Negative values for **width** or **height** are an error (see Error processing). Zero values disable rendering of the element which referenced the filter.

The bounds of this rectangle act as a hard clipping region for each filter primitive included with a given **'filter'** element; thus, if the effect of a given filter primitive would extend beyond the bounds of the rectangle (this sometimes happens when using a **'feGaussianBlur'** filter primitive with a very large **stdDeviation**), parts of the effect will get clipped.
If **x** or **y** is not specified, the effect is as if a value of "-10%" were specified.
If **width** or **height** is not specified, the effect is as if a value of "120%" were specified.
*Animatable: yes.*

- **filterRes** (which has the form `x-pixels [y-pixels]`) indicates the width and height of the intermediate images in pixels. If not provided, then a reasonable default resolution appropriate for the target device will be used. (For displays, an appropriate display resolution, preferably the current display's pixel resolution, is the default. For printing, an appropriate common printer resolution, such as 400dpi, is the default.)
  Care should be taken when assigning a non-default value to this attribute. Too small of a value may result in unwanted pixelation in the result. Too large of a value may result in slow processing and large memory usage. Negative values are an error (see Error processing). Zero values disable rendering of the element which referenced the filter.
  *Animatable: yes.*

Note that both of the two possible value for **filterUnits** (i.e., **objectBoundingBox** and **userSpaceOnUse**) result in a filter region whose coordinate system has its X-axis and Y-axis each parallel to the X-axis and Y-axis, respectively, of the user coordinate system for the element to which the filter will be applied.

Sometimes implementers can achieve faster performance when the filter region can be mapped directly to device pixels; thus, for best performance on display devices, it is suggested that authors define their region such that SVG user agent can align the filter region pixel-for-pixel with the background. In particular, for best filter effects performance, avoid rotating or skewing the user coordinate system. Explicit values for attribute **filterRes** can either help or harm performance. If **filterRes** is smaller than the automatic (i.e., default) filter resolution, then filter effect might have faster performance (usually at the expense of quality). If **filterRes** is larger than the automatic (i.e., default) filter resolution, then filter effects performance will usually be slower.

It is often necessary to provide padding space because the filter effect might impact bits slightly outside the tight-fitting bounding box on a given object. For these purposes, it is possible to provide negative percentage values for **x, y** and percentages values greater than 100% for **width, height**. This, for example, is why the defaults for the filter effects region are x="-10%" y="-10%" width="120%" height="120%".

## 15.6 Accessing the background image

Two possible pseudo input images for filter effects are BackgroundImage and BackgroundAlpha, which each represent an image snapshot of the canvas under the filter region at the time that the **'filter'** element is invoked. BackgroundImage represents both the color values and alpha channel of the canvas (i.e., RGBA pixel values), whereas BackgroundAlpha represents only the alpha channel.

Implementations of SVG user agents often will need to maintain supplemental background image buffers in order to support the BackgroundImage and BackgroundAlpha pseudo input images. Sometimes, the background image buffers will contain an in-memory copy of the accumulated painting operations on the current canvas.

Because in-memory image buffers can take up significant system resources, SVG content must explicitly indicate to the SVG user agent that the document needs access to the background image before BackgroundImage and BackgroundAlpha pseudo input images can be used. The property which enables access to the background image is **'enable-background'**:

**'enable-background'**

| | |
|---|---|
| *Value:* | accumulate \| new [ <x> <y> <width> <height> ] \| inherit |
| *Initial:* | accumulate |
| *Applies to:* | container elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |

*Animatable:* no

**'enable-background'** is only applicable to container elements and specifies how the SVG user agents manages the accumulation of the background image.

A value of **new** indicates two things:

- It enables the ability of children of the current container element to access the background image.
- It indicates that a new (i.e., initially transparent black) background image canvas is established and that (in effect) all children of the current container element shall be rendered into the new background image canvas in addition to being rendered onto the target device.

A meaning of **enable-background: accumulate** (the initial/default value) depends on context:

- If an ancestor container element has a property value of 'enable-background:new', then all graphics elements within the current container element are rendered both onto the parent container element's background image canvas and onto the target device.
- Otherwise, there is no current background image canvas, so it is only necessary to render graphics elements onto the target device. (No need to render to the background image canvas.)

If a filter effect specifies either the BackgroundImage or the BackgroundAlpha pseudo input images and no ancestor container element has a property value of 'enable-background:new', then the background image request is technically in error. Processing will proceed without interruption (i.e., no error message) and a transparent black image shall be provided in response to the request.

The optional **<x>,<y>,<width>,<height>** parameters on the **new** value indicate the subregion of the container element's user space where access to the background image is allowed to happen. These parameters enable the SVG user agent potentially to allocate smaller temporary image buffers than the default values, which might require the SVG user agent to allocate buffers as large as the current viewport. Thus, the values <x>,<y>,<width>,<height> act as a clipping rectangle on the background image canvas. Negative values for <width> or <height> are an error (see Error processing). If more than zero but less than four of the values <x>,<y>,<width> and <height> are specified or if zero values are specified for <width> or <height>, BackgroundImage and BackgroundAlpha are processed as if background image processing were not enabled.

Assume you have an element E in the document and that E has a series of ancestors $A_1$ (its immediate parent), $A_2$, etc. (Note: $A_0$ is E.) Each ancestor $A_i$ will have a corresponding temporary background image offscreen buffer $BUF_i$. The contents of the *background image* available to a **'filter'** referenced by E is defined as follows:

- Find the element $A_i$ with the smallest subscript i (including $A_0$=E) for which the **'enable-background'** property has the value **new**. (Note: if there is no such ancestor element, then there is no background image available to E, in which case a transparent black image will be used as E's background image.)
- For each $A_i$ (from i=n to 1), initialize $BUF_i$ to transparent black. Render all children of $A_i$ up to but not including $A_{i-1}$ into $BUF_i$. The children are painted, then filtered, clipped, masked and composited using the various painting, filtering, clipping, masking and object opacity settings on the given child. Any filter effects, masking and group opacity that might be set on $A_i$ do *not* apply when rendering the children of $A_i$ into $BUF_i$.
  (Note that for the case of $A_0$=E, the graphical contents of E are not rendered into $BUF_1$ and thus are not part of the background image available to E. Instead, the graphical contents of E are available via the SourceGraphic and SourceAlpha pseudo input images.)
- Then, for each $A_i$ (from i=1 to n-1), composite $BUF_i$ into $BUF_{i+1}$.
- The accumulated result (i.e., $BUF_n$) represents the background image available to E.

Example enable-background-01 illustrates the rules for background image processing.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

```
<svg width="13.5cm" height="2.7cm" viewBox="0 0 1350 270"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example enable-background01</title>
  <desc>This test case shows five pictures which illustrate the rules
        for background image processing.</desc>

  <defs>
    <filter id="ShiftBGAndBlur"
            filterUnits="userSpaceOnUse" x="0" y="0" width="1200" height="400">
      <desc>
        This filter discards the SourceGraphic, if any, and just produces
        a result consisting of the BackgroundImage shifted down 125 units
        and then blurred.
      </desc>
      <feOffset in="BackgroundImage" dx="0" dy="125" />
      <feGaussianBlur stdDeviation="8" />
    </filter>
    <filter id="ShiftBGAndBlur_WithSourceGraphic"
            filterUnits="userSpaceOnUse" x="0" y="0" width="1200" height="400">
      <desc>
        This filter takes the BackgroundImage, shifts it down 125 units, blurs it,
        and then renders the SourceGraphic on top of the shifted/blurred background.
      </desc>
      <feOffset in="BackgroundImage" dx="0" dy="125" />
      <feGaussianBlur stdDeviation="8" result="blur" />
      <feMerge>
        <feMergeNode in="blur"/>
        <feMergeNode in="SourceGraphic"/>
      </feMerge>
    </filter>
  </defs>

  <g transform="translate(0,0)">
    <desc>The first picture is our reference graphic without filters.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>

  <g enable-background="new" transform="translate(270,0)">
    <desc>The second adds an empty 'g' element which invokes ShiftBGAndBlur.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <g filter="url(#ShiftBGAndBlur)"/>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>

  <g enable-background="new" transform="translate(540,0)">
    <desc>The third invokes ShiftBGAndBlur on the inner group.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g filter="url(#ShiftBGAndBlur)" opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>

  <g enable-background="new" transform="translate(810,0)">
    <desc>The fourth invokes ShiftBGAndBlur on the triangle.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"
               filter="url(#ShiftBGAndBlur)"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>

  <g enable-background="new" transform="translate(1080,0)">
    <desc>The fifth invokes ShiftBGAndBlur_WithSourceGraphic on the triangle.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"
               filter="url(#ShiftBGAndBlur_WithSourceGraphic)"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
</svg>
```

*Example enable-background-01*

*View this example as SVG (SVG-enabled browsers only)*

The example above contains five parts, described as follows:

1. The first set is the reference graphic. The reference graphic consists of a red rectangle followed by a 50% transparent **'g'** element. Inside the **'g'** is a green circle that partially overlaps the rectangle and a a blue triangle that partially overlaps the circle. The three objects are then outlined by a rectangle stroked with a thin blue line. No filters are applied to the reference graphic.
2. The second set enables background image processing and adds an empty **'g'** element which invokes the ShiftBGAndBlur filter. This filter takes the current accumulated background image (i.e., the entire reference graphic) as input, shifts its offscreen down, blurs it, and then writes the result to the canvas. Note that the offscreen for the filter is initialized to transparent black, which allows the already rendered rectangle, circle and triangle to show through after the filter renders its own result to the canvas.
3. The third set enables background image processing and instead invokes the ShiftBGAndBlur filter on the inner **'g'** element. The accumulated background at the time the filter is applied contains only the red rectangle. Because the children of the inner **'g'** (i.e., the circle and triangle) are not part of the inner **'g'** element's background and because ShiftBGAndBlur ignores SourceGraphic, the children of the inner **'g'** do not appear in the result.
4. The fourth set enables background image processing and invokes the ShiftBGAndBlur on the **'polygon'** element that draws the triangle. The accumulated background at the time the filter is applied contains the red rectangle plus the green circle ignoring the effect of the **'opacity'** property on the inner **'g'** element. (Note that the blurred green circle at the bottom does not let the red rectangle show through on its left side. This is due to ignoring the effect of the **'opacity'** property.) Because the triangle itself is not part of the accumulated background and because ShiftBGAndBlur ignores SourceGraphic, the triangle does not appear in the result.
5. The fifth set is the same as the fourth except that filter ShiftBGAndBlur_WithSourceGraphic is invoked instead of ShiftBGAndBlur. ShiftBGAndBlur_WithSourceGraphic performs the same effect as ShiftBGAndBlur, but then renders the SourceGraphic on top of the shifted, blurred background image. In this case, SourceGraphic is the blue triangle; thus, the result is the same as in the fourth case except that the blue triangle now appears.

## 15.7 Filter primitives overview

### 15.7.1 Overview

This section describes the various filter primtives that can be assembled to achieve a particular filter effect.

Unless otherwise stated, all image filters operate on premultiplied RGBA samples. Filters which work more naturally on non-premultiplied data (feColorMatrix and feComponentTransfer) will temporarily undo and redo premultiplication as specified. All raster effect filtering operations take 1 to N input RGBA images, additional attributes as parameters, and produce a single output RGBA image.

The RGBA result from each filter primitive will be clamped into the allowable ranges for colors and opacity values. Thus, for example, the result from a given filter primitive will have any negative color values or opacity values adjusted up to color/opacity of zero.

The color space in which a particular filter primitive performs its operations is determined by the value of property **'color-**

**interpolation-filters'** on the given filter primitive. A different property, **'color-interpolation'** determines the color space for other color operations. Because these two properties have different initial values (**'color-interpolation-filters'** has an initial value of **linearRGB** whereas **'color-interpolation'** has an initial value of **sRGB**), in some cases to achieve certain results (e.g., when coordinating gradient interpolation with a filtering operation) it will be necessary to explicitly set **'color-interpolation'** to **linearRGB** or **'color-interpolation-filters'** to **sRGB** on particular elements. Note that the examples below do not explicitly set either **'color-interpolation'** or **'color-interpolation-filters'**, so the initial values for these properties apply to the examples.

### 15.7.2 Common attributes

The following attributes are available for most of the filter primitives:

```
<!ENTITY % filter_primitive_attributes
  "x %Coordinate; #IMPLIED
   y %Coordinate; #IMPLIED
   width %Length; #IMPLIED
   height %Length; #IMPLIED
   result CDATA #IMPLIED" >

<!ENTITY % filter_primitive_attributes_with_in
  "%filter_primitive_attributes;
   in CDATA #IMPLIED">
```

*Attribute definitions:*

**x = "*<coordinate>*"**
: The minimum x coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.
*Animatable: yes.*

**y = "*<coordinate>*"**
: The minimum y coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion. *Animatable: yes.*

**width = "*<length>*"**
: The width of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).
*Animatable: yes.*

**height = "*<length>*"**
: The height of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).
*Animatable: yes.*

**result = "*<filter-primitive-reference>*"**
: Assigned name for this filter primitive. If supplied, then graphics that result from processing this filter primitive can be referenced by an **in** attribute on a subsequent filter primitive within the same **'filter'** element. If no value is provided, the output will only be available for re-use as the implicit input into the next filter primitive if that filter primitive provides no value for its **in** attribute.
Note that a *<filter-primitive-reference>* is not an XML ID; instead, a *<filter-primitive-reference>* is only meaningful within a given **'filter'** element and thus have only local scope. It is legal for the same *<filter-primitive-reference>* to appear multiple times within the same **'filter'** element. When referenced, the *<filter-primitive-reference>* will use the closest preceding filter primitive with the given result.
*Animatable: yes.*

**in = "*SourceGraphic* | *SourceAlpha* | *BackgroundImage* | *BackgroundAlpha* | *FillPaint* | *StrokePaint* | *<filter-primitive-reference>*"**
: Identifies input for the given filter primitive. The value can be either one of six keywords or can be a string which matches a previous **result** attribute value within the same **'filter'** element. If no value is provided and this is the

first filter primitive, then this filter primitive will use **SourceGraphic** as its input. If no value is provided and this is a subsequent filter primitive, then this filter primitive will use the result from the previous filter primitive as its input.

If the value for **result** appears multiple times within a given **'filter'** element, then a reference to that result will use the closest preceding filter primitive with the given value for attribute **result**. Forward references to results are an error.

Definitions for the six keywords:

**SourceGraphic**
: This keyword represents the graphics elements that were the original input into the **'filter'** element. For raster effects filter primitives, the graphics elements will be rasterized into an initially clear RGBA raster in image space. Pixels left untouched by the original graphic will be left clear. The image is specified to be rendered in linear RGBA pixels. The alpha channel of this image captures any anti-aliasing specified by SVG. (Since the raster is linear, the alpha channel of this image will represent the exact percent coverage of each pixel.)

**SourceAlpha**
: This keyword represents the graphics elements that were the original input into the **'filter'** element. **SourceAlpha** has all of the same rules as **SourceGraphic** except that only the alpha channel is used. The input image is an RGBA image consisting of implicitly black color values for the RGB channels, but whose alpha channel is the same as **SourceGraphic**. If this option is used, then some implementations might need to rasterize the graphics elements in order to extract the alpha channel.

**BackgroundImage**
: This keyword represents an image snapshot of the canvas under the filter region at the time that the **'filter'** element was invoked. See Accessing the background image.

**BackgroundAlpha**
: Same as BackgroundImage except only the alpha channel is used. See **SourceAlpha** and Accessing the background image.

**FillPaint**
: This keyword represents the value of the **'fill'** property on the target element for the filter effect. The FillPaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

**StrokePaint**
: This keyword represents the value of the **'stroke'** property on the target element for the filter effect. The StrokePaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

*Animatable: yes.*

### 15.7.3 Filter primitive subregion

All filter primitives have attributes **x**, **y**, **width** and **height** which identify a subregion which restricts calculation and rendering of the given filter primitive. These attributes are defined according to the same rules as other filter primitives' coordinate and length attributes and thus represent values in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.

**x**, **y**, **width** and **height** default to the union (i.e., tightest fitting bounding box) of the subregions defined for all referenced nodes. If there are no referenced nodes (e.g., for **'feImage'** or **'feTurbulence'**), or one or more of the referenced nodes is a standard input (one of **SourceGraphic**, **SourceAlpha**, **BackgroundImage**, **BackgroundAlpha**, **FillPaint** or **StrokePaint**), or for **'feTile'** (which is special because its principal function is to replicate the referenced node in X and Y and thereby produce a usually larger result), the default subregion is 0%,0%,100%,100%, where percentages are relative to the dimensions of the filter region.

**x**, **y**, **width** and **height** act as a hard clip clipping rectangle.

All intermediate offscreens are defined to not exceed the intersection of **x**, **y**, **width** and **height** with the filter region. The filter region and any of the **x**, **y**, **width** and **height** subregions are to be set up such that all offscreens are made big enough to accommodate any pixels which even partly intersect with either the filter region or the x,y,width,height

subregions.

**'feTile'** references a previous filter primitive and then stitches the tiles together based on the **x**, **y**, **width** and **height** values of the referenced filter primitive in order to fill its own filter primitive subregion.

## 15.8 Light source elements and properties

### 15.8.1 Introduction

The following sections define the elements that define a light source, **'feDistantLight'**, **'fePointLight'** and **'feSpotLight'**, and property **'lighting-color'**, which defines the color of the light.

### 15.8.2 Light source 'feDistantLight'

```
<!ELEMENT feDistantLight (animate|set)* >
<!ATTLIST feDistantLight
  %stdAttrs;
  azimuth %Number; #IMPLIED
  elevation %Number; #IMPLIED >
```

*Attribute definitions:*

**azimuth = "*<number>*"**
: Direction angle for the light source on the XY plane, in degrees.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**elevation = "*<number>*"**
: Direction angle for the light source on the YZ plane, in degrees.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
: %stdAttrs;.

### 15.8.3 Light source 'fePointLight'

```
<!ELEMENT fePointLight (animate|set)* >
<!ATTLIST fePointLight
  %stdAttrs;
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  z %Number; #IMPLIED >
```

*Attribute definitions:*

**x = "*<number>*"**
: X location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**y = "*<number>*"**
: Y location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'**

element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**z = "*<number>*"**
Z location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
%stdAttrs;.

### 15.8.4 Light source 'feSpotLight'

```
<!ELEMENT feSpotLight (animate|set)* >
<!ATTLIST feSpotLight
  %stdAttrs;
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  z %Number; #IMPLIED
  pointsAtX %Number; #IMPLIED
  pointsAtY %Number; #IMPLIED
  pointsAtZ %Number; #IMPLIED
  specularExponent %Number; #IMPLIED
  limitingConeAngle %Number; #IMPLIED >
```

*Attribute definitions:*

**x = "*<number>*"**
X location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**y = "*<number>*"**
Y location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**z = "*<number>*"**
Z location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**pointsAtX = "*<number>*"**
X location in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element of the point at which the light source is pointing.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**pointsAtY = "*<number>*"**
Y location in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element of the point at which the light source is pointing.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**pointsAtZ = "*<number>*"**

Z location of the point at which the light source is pointing, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**`specularExponent` = "*<number>*"**

Exponent value controlling the focus for the light source.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`limitingConeAngle` = "*<number>*"**

A limiting cone which restricts the region where the light is projected. No light is projected outside the cone. **limitingConeAngle** represents the angle between the spot light axis (i.e. the axis between the light source and the point to which it is pointing at) and the spot light cone. User agents should apply a smoothing technique such as anti-aliasing at the boundary of the cone.
If no value is specified, then no limiting cone will be applied.
*Animatable: yes.*

***Attributes defined elsewhere:***

%stdAttrs;.

### 15.8.5 The 'lighting-color' property

The **'lighting-color'** property defines the color of the light source for filter primitives **'feDiffuseLighting'** and **'feSpecularLighting'**.

**'lighting-color'**

| | |
|---|---|
| *Value:* | currentColor \| <color> [icc-color(<name>[,<icccolorvalue>]*)] \| inherit |
| *Initial:* | white |
| *Applies to:* | **'feDiffuseLighting'** and **'feSpecularLighting'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

## 15.9 Filter primitive 'feBlend'

This filter composites two objects together using commonly used imaging software blending modes. It performs a pixel-wise combination of two input images.

```
<!ELEMENT feBlend (animate|set)* >
<!ATTLIST feBlend
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  in2 CDATA #REQUIRED
  mode (normal | multiply | screen | darken | lighten) "normal" >
```

*Attribute definitions:*

**`mode` = "*normal* | *multiply* | *screen* | *darken* | *lighten*"**

One of the image blending modes (see table below). Default is: normal.
*Animatable: yes.*

**`in2` = "*(see in attribute)*"**

The second input image to the blending operation. This attribute can take on the same values as the **in** attribute.
*Animatable: yes.*

***Attributes defined elsewhere:***
%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

For all feBlend modes, the result opacity is computed as follows:

```
qr = 1 - (1-qa)*(1-qb)
```

For the compositing formulas below, the following definitions apply:

```
cr = Result color (RGB) - premultiplied
qa = Opacity value at a given pixel for image A
qb = Opacity value at a given pixel for image B
ca = Color (RGB) at a given pixel for image A - premultiplied
cb = Color (RGB) at a given pixel for image B - premultiplied
```

The following table provides the list of available image blending modes:

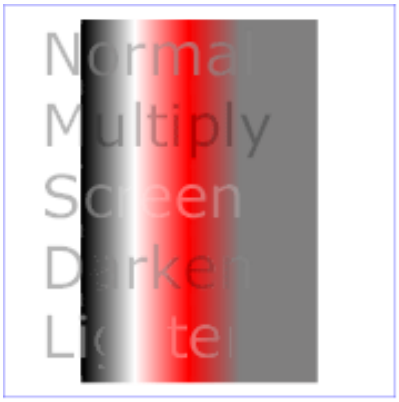
| Image Blending Mode | Formula for computing result color |
|---|---|
| normal | cr = (1 - qa) * cb + ca |
| multiply | cr = (1-qa)*cb + (1-qb)*ca + ca*cb |
| screen | cr = cb + ca - ca * cb |
| darken | cr = Min ((1 - qa) * cb + ca, (1 - qb) * ca + cb) |
| lighten | cr = Max ((1 - qa) * cb + ca, (1 - qb) * ca + cb) |

**'normal'** blend mode is equivalent to **operator="over"** on the **'feComposite'** filter primitive, matches the blending method used by **'feMerge'** and matches the simple alpha compositing technique used in SVG for all compositing outside of filter effects.

Example feBlend shows examples of the five blend modes.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="5cm" viewBox="0 0 500 500"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feBlend - Examples of feBlend modes</title>
  <desc>Five text strings blended into a gradient,
        with one text string for each of the five feBlend modes.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="300" y2="0">
      <stop offset="0" stop-color="#000000" />
      <stop offset=".33" stop-color="#ffffff" />
      <stop offset=".67" stop-color="#ff0000" />
      <stop offset="1" stop-color="#808080" />
    </linearGradient>
    <filter id="Normal">
      <feBlend mode="normal" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Multiply">
      <feBlend mode="multiply" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Screen">
      <feBlend mode="screen" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Darken">
      <feBlend mode="darken" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Lighten">
      <feBlend mode="lighten" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="498" height="498"/>
  <g enable-background="new" >
    <rect x="100" y="20" width="300" height="460" fill="url(#MyGradient)" />
```

```
      <g font-family="Verdana" font-size="75" fill="#888888" fill-opacity=".6" >
        <text x="50" y="90" filter="url(#Normal)" >Normal</text>
        <text x="50" y="180" filter="url(#Multiply)" >Multiply</text>
        <text x="50" y="270" filter="url(#Screen)" >Screen</text>
        <text x="50" y="360" filter="url(#Darken)" >Darken</text>
        <text x="50" y="450" filter="url(#Lighten)" >Lighten</text>
      </g>
    </g>
  </svg>
```




***Example feBlend***

*View this example as SVG (SVG-enabled browsers only)*

## 15.10 Filter primitive 'feColorMatrix'

This filter applies a matrix transformation:

```
| R' |     | a00 a01 a02 a03 a04 |   | R |
| G' |     | a10 a11 a12 a13 a14 |   | G |
| B' |  =  | a20 a21 a22 a23 a24 | * | B |
| A' |     | a30 a31 a32 a33 a34 |   | A |
| 1  |     |  0   0   0   0   1  |   | 1 |
```

on the RGBA color and alpha values of every pixel on the input graphics to produce a result with a new set of RGBA color and alpha values.

The calculations are performed on non-premultiplied color values. If the input graphics consists of premultiplied color values, those values are automatically converted into non-premultiplied color values for this operation.

These matrices often perform an identity mapping in the alpha channel. If that is the case, an implementation can avoid the costly undoing and redoing of the premultiplication for all pixels with A = 1.

```
<!ELEMENT feColorMatrix (animate|set)* >
<!ATTLIST feColorMatrix
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  type (matrix | saturate | hueRotate | luminanceToAlpha) "matrix"
  values CDATA #IMPLIED >
```

*Attribute definitions:*

**`type` = "*matrix* | *saturate* | *hueRotate* | *luminanceToAlpha*"**

Indicates the type of matrix operation. The keyword **matrix** indicates that a full 5x4 matrix of values will be provided. The other keywords represent convenience shortcuts to allow commonly used color operations to be performed without specifying a complete matrix.
*Animatable: yes.*

**`values` = "*list of <number>s*"**

The contents of **values** depends on the value of attribute **type**:

- For **type="matrix"**, **values** is a list of 20 matrix values (a00 a01 a02 a03 a04 a10 a11 ... a34), separated by whitespace and/or a comma. For example, the identity matrix could be expressed as:

```
type="matrix"
values="1 0 0 0 0  0 1 0 0 0  0 0 1 0 0  0 0 0 1 0"
```

- For **type="saturate"**, **values** is a single real number value (0 to 1). A **saturate** operation is equivalent to the following matrix operation:

```
| R' |     |0.213+0.787s  0.715-0.715s  0.072-0.072s 0  0 |   | R |
| G' |     |0.213-0.213s  0.715+0.285s  0.072-0.072s 0  0 |   | G |
| B' |  =  |0.213-0.213s  0.715-0.715s  0.072+0.928s 0  0 | * | B |
| A' |     |          0           0             0  1  0 |   | A |
| 1  |     |          0           0             0  0  1 |   | 1 |
```

- For **type="hueRotate"**, **values** is a single one real number value (degrees). A **hueRotate** operation is equivalent to the following matrix operation:

```
| R' |     | a00  a01  a02  0  0 |   | R |
| G' |     | a10  a11  a12  0  0 |   | G |
| B' |  =  | a20  a21  a22  0  0 | * | B |
| A' |     | 0    0    0    1  0 |   | A |
| 1  |     | 0    0    0    0  1 |   | 1 |
```

where the terms a00, a01, etc. are calculated as follows:

```
| a00 a01 a02 |    [+0.213 +0.715 +0.072]
| a10 a11 a12 | =  [+0.213 +0.715 +0.072] +
| a20 a21 a22 |    [+0.213 +0.715 +0.072]

                        [+0.787 -0.715 -0.072]
cos(hueRotate value) *  [-0.213 +0.285 -0.072] +
                        [-0.213 -0.715 +0.928]

                        [-0.213 -0.715+0.928]
sin(hueRotate value) *  [+0.143 +0.140-0.283]
                        [-0.787 +0.715+0.072]
```

Thus, the upper left term of the hue matrix turns out to be:

```
.213 + cos(hueRotate value)*.787 - sin(hueRotate value)*.213
```

- For **type="luminanceToAlpha"**, **values** is not applicable. A **luminanceToAlpha** operation is equivalent to the following matrix operation:

```
   | R' |     |      0       0       0  0  0 |   | R |
   | G' |     |      0       0       0  0  0 |   | G |
   | B' |  =  |      0       0       0  0  0 | * | B |
```

```
| A' |     | 0.2125   0.7154   0.0721  0  0 |   | A |
| 1  |     |      0        0        0  0  1 |   | 1 |
```

If the attribute is not specified, then the default behavior depends on the value of attribute **type**. If **type="matrix"**, then this attribute defaults to the identity matrix. If **type="saturate"**, then this attribute defaults to the value **1**, which results in the identify matrix. If **type="hueRotate"**, then this attribute defaults to the value **0**, which results in the identify matrix.
*Animatable: yes.*

***Attributes defined elsewhere:***
%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

Example feColorMatrix shows examples of the four types of feColorMatrix operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="5cm" viewBox="0 0 800 500"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feColorMatrix - Examples of feColorMatrix operations</title>
  <desc>Five text strings showing the effects of feColorMatrix:
        an unfiltered text string acting as a reference,
        use of the feColorMatrix matrix option to convert to grayscale,
        use of the feColorMatrix saturate option,
        use of the feColorMatrix hueRotate option,
        and use of the feColorMatrix luminanceToAlpha option.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="500" y2="0">
      <stop offset="0" stop-color="#ff00ff" />
      <stop offset=".33" stop-color="#88ff88" />
      <stop offset=".67" stop-color="#2020ff" />
      <stop offset="1" stop-color="#d00000" />
    </linearGradient>
    <filter id="Matrix" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="matrix" in="SourceGraphic"
           values=".33 .33 .33 0 0
                   .33 .33 .33 0 0
                   .33 .33 .33 0 0
                   .33 .33 .33 0 0"/>
    </filter>
    <filter id="Saturate40" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="saturate" in="SourceGraphic" values="40%"/>
    </filter>
    <filter id="HueRotate90" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="hueRotate" in="SourceGraphic" values="90"/>
    </filter>
    <filter id="LuminanceToAlpha" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="luminanceToAlpha" in="SourceGraphic" result="a"/>
      <feComposite in="SourceGraphic" in2="a" operator="in" />
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="798" height="498"/>
  <g font-family="Verdana" font-size="75"
           font-weight="bold" fill="url(#MyGradient)" >
    <rect x="100" y="0" width="500" height="20" />
    <text x="100" y="90">Unfiltered</text>
    <text x="100" y="190" filter="url(#Matrix)" >Matrix</text>
    <text x="100" y="290" filter="url(#Saturate40)" >Saturate</text>
    <text x="100" y="390" filter="url(#HueRotate90)" >HueRotate</text>
    <text x="100" y="490" filter="url(#LuminanceToAlpha)" >Luminance</text>
  </g>
</svg>
```


*Example feColorMatrix*

*View this example as SVG (SVG-enabled browsers only)*

## 15.11 Filter primitive 'feComponentTransfer'

This filter primitive performs component-wise remapping of data as follows:

```
R' = feFuncR( R )
G' = feFuncG( G )
B' = feFuncB( B )
A' = feFuncA( A )
```

for every pixel. It allows operations like brightness adjustment, contrast adjustment, color balance or thresholding.

The calculations are performed on non-premultiplied color values. If the input graphics consists of premultiplied color values, those values are automatically converted into non-premultiplied color values for this operation. (Note that the undoing and redoing of the premultiplication can be avoided if feFuncA is the identity transform and all alpha values on the source graphic are set to 1.)

```
<!ELEMENT feComponentTransfer (feFuncR?,feFuncG?,feFuncB?,feFuncA?) >
<!ATTLIST feComponentTransfer
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >

<!ENTITY % component_transfer_function_attributes
  "type (identity | table | discrete | linear | gamma) #REQUIRED
   tableValues CDATA #IMPLIED
   slope %Number; #IMPLIED
   intercept %Number; #IMPLIED
   amplitude %Number; #IMPLIED
   exponent %Number; #IMPLIED
   offset %Number; #IMPLIED" >

<!ELEMENT feFuncR (animate|set)* >
<!ATTLIST feFuncR
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feFuncG (animate|set)* >
<!ATTLIST feFuncG
  %stdAttrs;
  %component_transfer_function_attributes; >
```

```
<!ELEMENT feFuncB (animate|set)* >
<!ATTLIST feFuncB
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feFuncA (animate|set)* >
<!ATTLIST feFuncA
  %stdAttrs;
  %component_transfer_function_attributes; >
```

**The specification of the transfer functions is defined by the sub-elements to 'feComponentTransfer':**

**'feFuncR'**, transfer function for red component of the input graphic
**'feFuncG'**, transfer function for green component of the input graphic
**'feFuncB'**, transfer function for blue component of the input graphic
**'feFuncA'**, transfer function for alpha component of the input graphic

The attributes below apply to sub-elements **'feFuncR'**, **'feFuncG'**, **'feFuncB'** and **'feFuncA'** define the transfer functions.

*Attribute definitions:*

**`type` = "*identity* | *table* | *discrete* | *linear* | *gamma*"**

Indicates the type of component transfer function. The type of function determines the applicability of the other attributes.

- For **identity**:

  $$C' = C$$

- For **table**, the function is defined by linear interpolation into a lookup table by attribute **tableValues**, which provides a list of *n+1* values (i.e., $v_0$ to $v_n$) in order to identify *n* interpolation ranges. Interpolations use the following formula.

  For a value `C` pick a k such that:

  $$k/N <= C < (k+1)/N$$

  The result `C'` is given by:

  $$C' = v_k + (C - k/N)*N * (v_{k+1} - v_k)$$

- For **discrete**, the function is defined by the step function defined by attribute **tableValues**, which provides a list of *n* values (i.e., $v_0$ to $v_{n-1}$) in order to identify a step function consisting of *n* steps. The step function is defined by the following formula.

  For a value `C` pick a k such that:

  $$k/N <= C < (k+1)/N$$

  The result `C'` is given by:

  $$C' = v_k$$

- For **linear**, the function is defined by the following linear equation:

```
C' = slope * C + intercept
```

- For **gamma**, the function is defined by the following exponential function:

```
C' = amplitude * pow(C, exponent) + offset
```

*Animatable: yes.*

**`tableValues` = "*(list of <number>s)*"**
When **type="table"**, the list of <number>s *v0,v1,...vn*, separated by white space and/or a comma, which define the lookup table. An empty list results in an identity transfer function.
If the attribute is not specified, then the effect is as if an empty list were provided.
*Animatable: yes.*

**`slope` = "*<number>*"**
When **type="linear"**, the slope of the linear function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`intercept` = "*<number>*"**
When **type="linear"**, the intercept of the linear function.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**`amplitude` = "*<number>*"**
When **type="gamma"**, the amplitude of the gamma function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`exponent` = "*<number>*"**
When **type="gamma"**, the exponent of the gamma function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`offset` = "*<number>*"**
When **type="gamma"**, the offset of the gamma function.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

Example feComponentTransfer shows examples of the four types of feComponentTransfer operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feComponentTransfer - Examples of feComponentTransfer operations</title>
  <desc>Four text strings showing the effects of feComponentTransfer:
        an identity function acting as a reference,
        use of the feComponentTransfer table option,
        use of the feComponentTransfer linear option,
        and use of the feComponentTransfer gamma option.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="600" y2="0">
      <stop offset="0" stop-color="#ff0000" />
      <stop offset=".33" stop-color="#00ff00" />
      <stop offset=".67" stop-color="#0000ff" />
      <stop offset="1" stop-color="#000000" />
    </linearGradient>
    <filter id="Identity" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="identity"/>
        <feFuncG type="identity"/>
        <feFuncB type="identity"/>
        <feFuncA type="identity"/>
      </feComponentTransfer>
    </filter>
    <filter id="Table" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
```

```
        <feFuncR type="table" tableValues="0 0 1 1"/>
        <feFuncG type="table" tableValues="1 1 0 0"/>
        <feFuncB type="table" tableValues="0 1 1 0"/>
      </feComponentTransfer>
    </filter>
    <filter id="Linear" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="linear" slope=".5" intercept=".25"/>
        <feFuncG type="linear" slope=".5" intercept="0"/>
        <feFuncB type="linear" slope=".5" intercept=".5"/>
      </feComponentTransfer>
    </filter>
    <filter id="Gamma" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="gamma" amplitude="2" exponent="5" offset="0"/>
        <feFuncG type="gamma" amplitude="2" exponent="3" offset="0"/>
        <feFuncB type="gamma" amplitude="2" exponent="1" offset="0"/>
      </feComponentTransfer>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="798" height="398"/>
  <g font-family="Verdana" font-size="75"
          font-weight="bold" fill="url(#MyGradient)" >
    <rect x="100" y="0" width="600" height="20" />
    <text x="100" y="90">Identity</text>
    <text x="100" y="190" filter="url(#Table)" >TableLookup</text>
    <text x="100" y="290" filter="url(#Linear)" >LinearFunc</text>
    <text x="100" y="390" filter="url(#Gamma)" >GammaFunc</text>
  </g>
</svg>
```


***Example feComponentTransfer***

*View this example as SVG (SVG-enabled browsers only)*

## 15.12 Filter primitive 'feComposite'

This filter performs the combination of the two input images pixel-wise in image space using one of the Porter-Duff [PORTERDUFF] compositing operations: *over, in, atop, out, xor*. Additionally, a component-wise *arithmetic* operation (with the result clamped between [0..1]) can be applied.

The *arithmetic* operation is useful for combining the output from the **'feDiffuseLighting'** and **'feSpecularLighting'** filters with texture data. It is also useful for implementing *dissolve*. If the *arithmetic* operation is chosen, each result pixel is computed using the following formula:

```
result = k1*i1*i2 + k2*i1 + k3*i2 + k4
```

For this filter primitive, the extent of the resulting image might grow as described in the section that describes the filter primitive subregion.

```
<!ELEMENT feComposite (animate|set)* >
<!ATTLIST feComposite
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  in2 CDATA #REQUIRED
  operator (over | in | out | atop | xor | arithmetic) "over"
  k1 %Number; #IMPLIED
  k2 %Number; #IMPLIED
  k3 %Number; #IMPLIED
  k4 %Number; #IMPLIED >
```

*Attribute definitions:*

**`operator` = "*over* | *in* | *out* | *atop* | *xor* | *arithmetic*"**
: The compositing operation that is to be performed. All of the **operator** types except **arithmetic** match the correspond operation as described in [PORTERDUFF]. The **arithmetic** operator is described above.
*Animatable: yes.*

**`k1` = "*<number>*"**
: Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`k2` = "*<number>*"**
: Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`k3` = "*<number>*"**
: Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`k4` = "*<number>*"**
: Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`in2` = "*(see in attribute)*"**
: The second input image to the compositing operation. This attribute can take on the same values as the **in** attribute.
*Animatable: yes.*

***Attributes defined elsewhere:***
: %stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

Example feComposite shows examples of the six types of feComposite operations. It also shows two different techniques to using the BackgroundImage as part of the compositing operation.

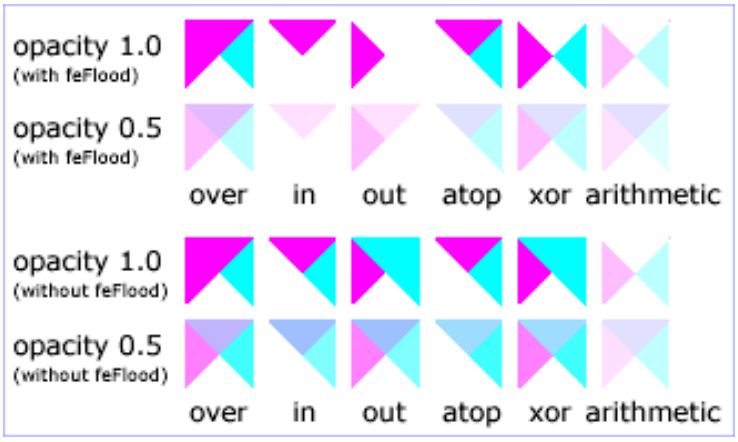
The first two rows render bluish triangles into the background. A filter is applied which composites reddish triangles into the bluish triangles using one of the compositing operations. The result from compositing is drawn onto an opaque white temporary surface, and then that result is written to the canvas. (The opaque white temporary surface obliterates the original bluish triangle.)

The last two rows apply the same compositing operations of reddish triangles into bluish triangles. However, the compositing result is directly blended into the canvas (the opaque white temporary surface technique is not used). In some cases, the results are different than when a temporary opaque white surface is used. The original bluish triangle from the background shines through wherever the compositing operation results in completely transparent pixel. In other cases, the result from compositing is blended into the bluish triangle, resulting in a different final color value.

```
<?xml version="1.0"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
         "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="330" height="195" viewBox="0 0 1100 650"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Example feComposite - Examples of feComposite operations</title>
  <desc>Four rows of six pairs of overlapping triangles depicting
        the six different feComposite operators under different
        opacity values and different clearing of the background.</desc>
        <defs>
    <desc>Define two sets of six filters for each of the six compositing operators.
          The first set wipes out the background image by flooding with opaque white.
          The second set does not wipe out the background, with the result
          that the background sometimes shines through and is other cases
          is blended into itself (i.e., "double-counting").</desc>
    <filter id="overFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="over" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="inFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="in" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="outFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="out" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="atopFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="atop" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="xorFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="xor" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="arithmeticFlood" filterUnits="objectBoundingBox"
            x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" result="comp"
                   operator="arithmetic" k1=".5" k2=".5" k3=".5" k4=".5"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="overNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="over" result="comp"/>
    </filter>
    <filter id="inNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="in" result="comp"/>
    </filter>
    <filter id="outNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="out" result="comp"/>
    </filter>
    <filter id="atopNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="atop" result="comp"/>
    </filter>
    <filter id="xorNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="xor" result="comp"/>
    </filter>
    <filter id="arithmeticNoFlood" filterUnits="objectBoundingBox"
            x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" result="comp"
                   operator="arithmetic" k1=".5" k2=".5" k3=".5" k4=".5"/>
    </filter>
    <path id="Blue100" d="M 0 0 L 100 0 L 100 100 z" fill="#00ffff" />
    <path id="Red100" d="M 0 0 L 0 100 L 100 0 z" fill="#ff00ff" />
    <path id="Blue50" d="M 0 125 L 100 125 L 100 225 z" fill="#00ffff" fill-opacity=".5" />
    <path id="Red50" d="M 0 125 L 0 225 L 100 125 z" fill="#ff00ff" fill-opacity=".5" />
    <g id="TwoBlueTriangles">
      <use xlink:href="#Blue100"/>
      <use xlink:href="#Blue50"/>
    </g>
    <g id="BlueTriangles">
      <use transform="translate(275,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(400,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(525,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(650,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(775,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(900,25)" xlink:href="#TwoBlueTriangles"/>
    </g>
  </defs>

  <rect fill="none" stroke="blue" x="1" y="1" width="1098" height="648"/>
  <g font-family="Verdana" font-size="40" shape-rendering="crispEdges">
    <desc>Render the examples using the filters that draw on top of
          an opaque white surface, thus obliterating the background.</desc>
    <g enable-background="new">
      <text x="15" y="75">opacity 1.0</text>
      <text x="15" y="115" font-size="27">(with feFlood)</text>
      <text x="15" y="200">opacity 0.5</text>
      <text x="15" y="240" font-size="27">(with feFlood)</text>
      <use xlink:href="#BlueTriangles"/>
      <g transform="translate(275,25)">
        <use xlink:href="#Red100" filter="url(#overFlood)" />
```

```
      <use xlink:href="#Red50" filter="url(#overFlood)" />
      <text x="5" y="275">over</text>
    </g>
    <g transform="translate(400,25)">
      <use xlink:href="#Red100" filter="url(#inFlood)" />
      <use xlink:href="#Red50" filter="url(#inFlood)" />
      <text x="35" y="275">in</text>
    </g>
    <g transform="translate(525,25)">
      <use xlink:href="#Red100" filter="url(#outFlood)" />
      <use xlink:href="#Red50" filter="url(#outFlood)" />
      <text x="15" y="275">out</text>
    </g>
    <g transform="translate(650,25)">
      <use xlink:href="#Red100" filter="url(#atopFlood)" />
      <use xlink:href="#Red50" filter="url(#atopFlood)" />
      <text x="10" y="275">atop</text>
    </g>
    <g transform="translate(775,25)">
      <use xlink:href="#Red100" filter="url(#xorFlood)" />
      <use xlink:href="#Red50" filter="url(#xorFlood)" />
      <text x="15" y="275">xor</text>
    </g>
    <g transform="translate(900,25)">
      <use xlink:href="#Red100" filter="url(#arithmeticFlood)" />
      <use xlink:href="#Red50" filter="url(#arithmeticFlood)" />
      <text x="-25" y="275">arithmetic</text>
    </g>
  </g>
  <g transform="translate(0,325)" enable-background="new">
  <desc>Render the examples using the filters that do not obliterate
        the background, thus sometimes causing the background to continue
        to appear in some cases, and in other cases the background
        image blends into itself ("double-counting").</desc>
    <text x="15" y="75">opacity 1.0</text>
    <text x="15" y="115" font-size="27">(without feFlood)</text>
    <text x="15" y="200">opacity 0.5</text>
    <text x="15" y="240" font-size="27">(without feFlood)</text>
    <use xlink:href="#BlueTriangles"/>
    <g transform="translate(275,25)">
      <use xlink:href="#Red100" filter="url(#overNoFlood)" />
      <use xlink:href="#Red50" filter="url(#overNoFlood)" />
      <text x="5" y="275">over</text>
    </g>
    <g transform="translate(400,25)">
      <use xlink:href="#Red100" filter="url(#inNoFlood)" />
      <use xlink:href="#Red50" filter="url(#inNoFlood)" />
      <text x="35" y="275">in</text>
    </g>
    <g transform="translate(525,25)">
      <use xlink:href="#Red100" filter="url(#outNoFlood)" />
      <use xlink:href="#Red50" filter="url(#outNoFlood)" />
      <text x="15" y="275">out</text>
    </g>
    <g transform="translate(650,25)">
      <use xlink:href="#Red100" filter="url(#atopNoFlood)" />
      <use xlink:href="#Red50" filter="url(#atopNoFlood)" />
      <text x="10" y="275">atop</text>
    </g>
    <g transform="translate(775,25)">
      <use xlink:href="#Red100" filter="url(#xorNoFlood)" />
      <use xlink:href="#Red50" filter="url(#xorNoFlood)" />
      <text x="15" y="275">xor</text>
    </g>
    <g transform="translate(900,25)">
      <use xlink:href="#Red100" filter="url(#arithmeticNoFlood)" />
      <use xlink:href="#Red50" filter="url(#arithmeticNoFlood)" />
      <text x="-25" y="275">arithmetic</text>
    </g>
  </g>
 </g>
</svg>
```




***Example feComposite***

*View this example as SVG (SVG-enabled browsers only)*

## 15.13 Filter primitive 'feConvolveMatrix'

feConvolveMatrix applies a matrix convolution filter effect. A convolution combines pixels in the input image with neighboring pixels to produce a resulting image. A wide variety of imaging operations can be achieved through convolutions, including blurring, edge detection, sharpening, embossing and beveling.

A matrix convolution is based on an n-by-m matrix (the convolution kernel) which describes how a given pixel value in the input image is combined with its neighboring pixel values to produce a resulting pixel value. Each result pixel is determined by applying the kernel matrix to the corresponding source pixel and its neighboring pixels. The basic convolution formula which is applied to each color value for a given pixel is:

$$\begin{aligned} RESULT_{X,Y} = ( & \\ & SUM_{I=0 \text{ to } [orderY-1]} \{ \\ & \quad SUM_{J=0 \text{ to } [orderX-1]} \{ \\ & \quad\quad SOURCE_{X-targetX+J,\ Y-targetY+I} * kernelMatrix_{orderX-J-1,\ orderY-I-1} \\ & \quad \} \\ & \} \\ ) & / divisor + bias \end{aligned}$$

where "orderX" and "orderY" represent the X and Y values for the **order** attribute, "targetX" represents the value of the **targetX** attribute, "targetY" represents the value of the **targetY** attribute, "kernelMatrix" represents the value of the **kernelMatrix** attribute, "divisor" represents the value of the **divisor** attribute, and "bias" represents the value of the **bias** attribute.

Note in the above formulas that the values in the kernel matrix are applied such that the kernel matrix is rotated 180 degrees relative to the source and destination images in order to match convolution theory as described in many computer graphics textbooks.

To illustrate, suppose you have a input image which is 5 pixels by 5 pixels, whose color values for one of the color channels are as follows:

```
  0  20  40 235 235
100 120 140 235 235
200 220 240 235 235
225 225 255 255 255
225 225 255 255 255
```

and you define a 3-by-3 convolution kernel as follows:

```
1 2 3
4 5 6
7 8 9
```

Let's focus on the color value at the second row and second column of the image (source pixel value is 120). Assuming the simplest case (where the input image's pixel grid aligns perfectly with the kernel's pixel grid) and assuming default values for attributes **divisor**, **targetX** and **targetY**, then resulting color value will be:

```
(9*  0 + 8* 20 + 7* 40 +
6*100 + 5*120 + 4*140 +
3*200 + 2*220 + 1*240) / (9+8+7+6+5+4+3+2+1)
```

Because they operate on pixels, matrix convolutions are inherently resolution-dependent. To make **'feConvolveMatrix** produce resolution-independent results, an explicit value should be provided for either the **filterRes** attribute on the **'filter'** element and/or attribute **kernelUnitLength**.

**kernelUnitLength**, in combination with the other attributes, defines an implicit pixel grid in the filter effects coordinate system (i.e., the coordinate system established by the **primitiveUnits** attribute). If the pixel grid established by **kernelUnitLength** is not scaled to match the pixel grid established by attribute **filterRes** (implicitly or explicitly), then the input image will be temporarily rescaled to match its pixels with **kernelUnitLength**. The convolution happens on the resampled image. After applying the convolution, the image is resampled back to the original resolution.

When the image must be resampled to match the coordinate system defined by **kernelUnitLength** prior to convolution, or resampled to match the device coordinate system after convolution, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting. Note that implementations might choose approaches that minimize or eliminate resampling when not necessary to produce proper results, such as when the document is zoomed out such that **kernelUnitLength** is considerably smaller than a device pixel.

```
<!ELEMENT feConvolveMatrix (animate|set)* >
<!ATTLIST feConvolveMatrix
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  order %NumberOptionalNumber; #REQUIRED
  kernelMatrix CDATA #REQUIRED
  divisor %Number; #IMPLIED
  bias %Number; #IMPLIED
  targetX %Integer; #IMPLIED
  targetY %Integer; #IMPLIED
  edgeMode (duplicate|wrap|none) "duplicate"
  kernelUnitLength %NumberOptionalNumber; #IMPLIED
  preserveAlpha %Boolean; #IMPLIED >
```

*Attribute definitions:*

**`order` = "<number-optional-number>"**

Indicates the number of cells in each dimension for **kernelMatrix**. The values provided must be <integer>s greater than zero. The first number, <orderX>, indicates the number of columns in the matrix. The second number, <orderY>, indicates the number of rows in the matrix. If <orderY> is not provided, it defaults to <orderX>. A typical value is order="3". It is recommended that only small values (e.g., 3) be used; higher values may result in very high CPU overhead and usually do not produce results that justify the impact on performance.
If the attribute is not specified, the effect is as if a value of "3" were specified.
*Animatable: yes.*

**`kernelMatrix` = "<list of numbers>"**

The list of <number>s that make up the kernel matrix for the convolution. Values are separated by space

characters and/or a comma. The number of entries in the list must equal <orderX> times <orderY>.
*Animatable: yes.*

**`divisor` = "<number>"**

After applying the **kernelMatrix** to the input image to yield a number, that number is divided by **divisor** to yield the final destination color value. A divisor that is the sum of all the matrix values tends to have an evening effect on the overall color intensity of the result. It is an error to specify a divisor of zero. The default value is the sum of all values in kernelMatrix, with the exception that if the sum is zero, then the divisor is set to 1.
*Animatable: yes.*

**`bias` = "<number>"**

After applying the **kernelMatrix** to the input image to yield a number and applying the **divisor**, the **bias** attribute is added to each component. One application of **bias** is when it is desirable to have .5 gray value be the zero response of the filter. If **bias** is not specified, then the effect is as if a value of zero were specified.
*Animatable: yes.*

**`targetX` = "<integer>"**

Determines the positioning in X of the convolution matrix relative to a given target pixel in the input image. The leftmost column of the matrix is column number zero. The value must be such that: 0 <= targetX < orderX. By default, the convolution matrix is centered in X over each pixel of the input image (i.e., targetX = floor ( orderX / 2 )).
*Animatable: yes.*

**`targetY` = "<integer>"**

Determines the positioning in Y of the convolution matrix relative to a given target pixel in the input image. The topmost row of the matrix is row number zero. The value must be such that: 0 <= targetY < orderY. By default, the convolution matrix is centered in Y over each pixel of the input image (i.e., targetY = floor ( orderY / 2 )).
*Animatable: yes.*

**`edgeMode` = "duplicate | wrap | none"**

Determines how to extend the input image as necessary with color values so that the matrix operations can be applied when the kernel is positioned at or near the edge of the input image.

"duplicate" indicates that the input image is extended along each of its borders as necessary by duplicating the color values at the given edge of the input image.

```
Original N-by-M image, where m=M-1 and n=N-1:
          11 12 ... 1m 1M
          21 22 ... 2m 2M
          .. .. ... .. ..
          n1 n2 ... nm nM
          N1 N2 ... Nm NM

Extended by two pixels using "duplicate":
  11 11   11 12 ... 1m 1M   1M 1M
  11 11   11 12 ... 1m 1M   1M 1M

  11 11   11 12 ... 1m 1M   1M 1M
  21 21   21 22 ... 2m 2M   2M 2M
  .. ..   .. .. ... .. ..   .. ..
  n1 n1   n1 n2 ... nm nM   nM nM
  N1 N1   N1 N2 ... Nm NM   NM NM

  N1 N1   N1 N2 ... Nm NM   NM NM
  N1 N1   N1 N2 ... Nm NM   NM NM
```

"wrap" indicates that the input image is extended by taking the color values from the opposite edge of the image.

```
Extended by two pixels using "wrap":
  nm nM   n1 n2 ... nm nM   n1 n2
  Nm NM   N1 N2 ... Nm NM   N1 N2

  1m 1M   11 12 ... 1m 1M   11 12
  2m 2M   21 22 ... 2m 2M   21 22
  .. ..   .. .. ... .. ..   .. ..
  nm nM   n1 n2 ... nm nM   n1 n2
  Nm NM   N1 N2 ... Nm NM   N1 N2

  1m 1M   11 12 ... 1m 1M   11 12
  2m 2M   21 22 ... 2m 2M   21 22
```

"none" indicates that the input image is extended with pixel values of zero for R, G, B and A.

*Animatable: yes.*

**`kernelUnitLength` = "<number-optional-number>"**

The first number is the <dx> value. The second number is the <dy> value. If the <dy> value is not specified, it defaults to the same value as <dx>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) between successive columns and rows, respectively, in the **kernelMatrix**. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary offscreen images aligns with the pixel grid of the kernel.
A negative or zero value is an error (see Error processing).
*Animatable: yes.*

**`preserveAlpha` = "false | true"**

A value of **false** indicates that the convolution will apply to all channels, including the alpha channel.
A value of **true** indicates that the convolution will only apply to the color channels. In this case, the filter will temporarily unpremultiply the color component values, apply the kernel, and then re-premultiply at the end.
If **preserveAlpha** is not specified, then the effect is as if a value of **false** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

## 15.14 Filter primitive 'feDiffuseLighting'

This filter primitive lights an image using the alpha channel as a bump map. The resulting image is an RGBA opaque image based on the light color with alpha = 1.0 everywhere. The lighting calculation follows the standard diffuse component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map.

The light map produced by this filter primitive can be combined with a texture image using the multiply term of the *arithmetic* **'feComposite'** compositing method. Multiple light sources can be simulated by adding several of these light maps together before applying it to the texture image.

The formulas below make use of 3x3 filters. Because they operate on pixels, such filters are inherently resolution-dependent. To make **'feDiffuseLighting'** produce resolution-independent results, an explicit value should be provided for either the **filterRes** attribute on the **'filter'** element and/or attribute **kernelUnitLength**.

**kernelUnitLength**, in combination with the other attributes, defines an implicit pixel grid in the filter effects coordinate system (i.e., the coordinate system established by the **primitiveUnits** attribute). If the pixel grid established by **kernelUnitLength** is not scaled to match the pixel grid established by attribute **filterRes** (implicitly or explicitly), then the input image will be temporarily rescaled to match its pixels with **kernelUnitLength**. The 3x3 filters are applied to the resampled image. After applying the filter, the image is resampled back to its original resolution.

When the image must be resampled, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting. Note that implementations might choose approaches that minimize or eliminate resampling when not necessary to produce proper results, such as when the document is zoomed out such that **kernelUnitLength** is considerably smaller than a device pixel.

For the formulas that follow, the $\mathtt{Norm}(A_x,A_y,A_z)$ function is defined as:

$$\mathtt{Norm}(A_x,A_y,A_z) = \mathtt{sqrt}(A_x\mathtt{^2}+A_y\mathtt{^2}+A_z\mathtt{^2})$$

The resulting RGBA image is computed as follows:

$$D_r = k_d * N.L * L_r$$
$$D_g = k_d * N.L * L_g$$
$$D_b = k_d * N.L * L_b$$
$$D_a = 1.0$$

where

$k_d$ = diffuse lighting constant
N = surface normal unit vector, a function of x and y
L = unit vector pointing from surface to light, a function of x and y in the point and spot light cases
$L_r$,$L_g$,$L_b$ = RGB components of light, a function of x and y in the spot light case

N is a function of x and y and depends on the surface gradient as follows:

The surface described by the input alpha image $A_{in}$ (x,y) is:

$$Z\ (x,y) = surfaceScale * A_{in}\ (x,y)$$

Surface normal is calculated using the Sobel gradient 3x3 filter. Different filter kernels are used depending on whether the given pixel is on the interior or an edge. For each case, the formula is:

$$N_x\ (x,y) = - surfaceScale * FACTOR_x *$$
$$(K_x(0,0)*I(x-dx,y-dy) + K_x(1,0)*I(x,y-dy) + K_x(2,0)*I(x+dx,y-dy) +$$
$$K_x(0,1)*I(x-dx,y) + K_x(1,1)*I(x,y) + K_x(2,1)*I(x+dx,y) +$$
$$K_x(0,2)*I(x-dx,y+dy) + K_x(1,2)*I(x,y+dy) + K_x(2,2)*I(x+dx,y+dy))$$
$$N_y\ (x,y) = - surfaceScale * FACTOR_y *$$
$$(K_y(0,0)*I(x-dx,y-dy) + K_y(1,0)*I(x,y-dy) + K_y(2,0)*I(x+dx,y-dy) +$$
$$K_y(0,1)*I(x-dx,y) + K_y(1,1)*I(x,y) + K_y(2,1)*I(x+dx,y) +$$
$$K_y(0,2)*I(x-dx,y+dy) + K_y(1,2)*I(x,y+dy) + K_y(2,2)*I(x+dx,y+dy))$$
$$N_z\ (x,y) = 1.0$$

$$N = (N_x, N_y, N_z) / Norm((N_x,N_y,N_z))$$

In these formulas, the `dx` and `dy` values (e.g., `I(x-dx,y-dy)`), represent deltas relative to a given `(x,y)` position for the purpose of estimating the slope of the surface at that point. These deltas are determined by the value (explicit or implicit) of attribute **kernelUnitLength**.

| Top/left corner: | Top row: | Top/right corner: |
|---|---|---|
| $FACTOR_x = 2/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$<br>$FACTOR_y = 2/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & -1 \\ 0 & 2 & 1 \end{vmatrix}$ | $FACTOR_x = 1/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$<br>$FACTOR_y = 1/(2*dy)$<br>$K_y =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & -1 \\ 1 & 2 & 1 \end{vmatrix}$ | $FACTOR_x = 2/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$<br>$FACTOR_y = 2/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & 0 \\ 1 & 2 & 0 \end{vmatrix}$ |

| Left column: | Interior pixels: | Right column: |
|---|---|---|
| $FACTOR_x=1/(2*dx)$<br>$K_x =$<br>$\begin{vmatrix} 0 & -1 & 1 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$<br>$FACTOR_y=1/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} 0 & -2 & -1 \\ 0 & 0 & 0 \\ 0 & 2 & 1 \end{vmatrix}$ | $FACTOR_x=1/(4*dx)$<br>$K_x =$<br>$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$<br>$FACTOR_y=1/(4*dy)$<br>$K_y =$<br>$\begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$ | $FACTOR_x=1/(2*dx)$<br>$K_x =$<br>$\begin{vmatrix} -1 & 1 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$<br>$FACTOR_y=1/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} -1 & -2 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 0 \end{vmatrix}$ |
| Bottom/left corner: | Bottom row: | Bottom/right corner: |
| $FACTOR_x=2/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} 0 & -1 & 1 \\ 0 & -2 & 2 \\ 0 & 0 & 0 \end{vmatrix}$<br>$FACTOR_y=2/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} 0 & -2 & -1 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \end{vmatrix}$ | $FACTOR_x=1/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 0 & 0 & 0 \end{vmatrix}$<br>$FACTOR_y=1/(2*dy)$<br>$K_y =$<br>$\begin{vmatrix} -1 & -2 & -1 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{vmatrix}$ | $FACTOR_x=2/(3*dx)$<br>$K_x =$<br>$\begin{vmatrix} -1 & 1 & 0 \\ -2 & 2 & 0 \\ 0 & 0 & 0 \end{vmatrix}$<br>$FACTOR_y=2/(3*dy)$<br>$K_y =$<br>$\begin{vmatrix} -1 & -2 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{vmatrix}$ |

L, the unit vector from the image sample to the light, is calculated as follows:

For Infinite light sources it is constant:

$$L_x = \cos(azimuth)*\cos(elevation)$$
$$L_y = \sin(azimuth)*\cos(elevation)$$
$$L_z = \sin(elevation)$$

For Point and spot lights it is a function of position:

$$L_x = Light_x - x$$
$$L_y = Light_y - y$$
$$L_z = Light_z - Z(x,y)$$

$$L = (L_x, L_y, L_z) / Norm(L_x, L_y, L_z)$$

where $Light_x$, $Light_y$, and $Light_z$ are the input light position.

$L_r$,$L_g$,$L_b$, the light color vector, is a function of position in the spot light case only:

$$L_r = Light_r*pow((-L.S),specularExponent)$$
$$L_g = Light_g*pow((-L.S),specularExponent)$$
$$L_b = Light_b*pow((-L.S),specularExponent)$$

where S is the unit vector pointing from the light to the point (pointsAtX, pointsAtY, pointsAtZ) in the x-y plane:

$$S_x = pointsAtX - Light_x$$
$$S_y = pointsAtY - Light_y$$
$$S_z = pointsAtZ - Light_z$$

$$S = (S_x, S_y, S_z) / Norm(S_x, S_y, S_z)$$

If L.S is positive, no light is present. ($L_r = L_g = L_b = 0$)

```
<!ELEMENT feDiffuseLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feDiffuseLighting
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FilterPrimitives;
  %PresentationAttributes-LightingEffects;
  %filter_primitive_attributes_with_in;
  surfaceScale %Number; #IMPLIED
  diffuseConstant %Number; #IMPLIED
  kernelUnitLength %NumberOptionalNumber; #IMPLIED >
```

*Attribute definitions:*

**`surfaceScale` = "*<number>*"**
: height of surface when $A_{in} = 1$.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`diffuseConstant` = "*<number>*"**
: kd in Phong lighting model. In SVG, this can be any non-negative number.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`kernelUnitLength` = "<number-optional-number>"**
: The first number is the <dx> value. The second number is the <dy> value. If the <dy> value is not specified, it defaults to the same value as <dx>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) for `dx` and `dy`, respectively, in the surface normal calculation formulas.
By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the `dx` and `dy` values should represent very small deltas relative to a given `(x,y)` position, which might be implemented in some cases as one pixel in the intermediate image offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. Discussion of intermediate images are in the Introduction and in the description of attribute **filterRes**.
A negative or zero value is an error (see Error processing).
*Animatable: yes.*

***Attributes defined elsewhere:***
: %stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-Color;, %PresentationAttributes-FilterPrimitives;, %PresentationAttributes-LightingEffects;.

The light source is defined by one of the child elements **'feDistantLight'**, **'fePointLight'** or **'feSpotLight'**. The light color is specified by property **'lighting-color'**.

## 15.15 Filter primitive 'feDisplacementMap'

This filter primitive uses the pixels values from the image from **in2** to spatially displace the image from **in**. This is the transformation to be performed:

```
P'(x,y) <- P( x + scale * ((XC(x,y) - .5), y + scale * (YC(x,y) - .5))
```

where P(x,y) is the input image, **in**, and P'(x,y) is the destination. XC(x,y) and YC(x,y) are the component values of the designated by the *xChannelSelector* and *yChannelSelector*. For example, to use the R component of **in2** to control

displacement in x and the G component of Image2 to control displacement in y, set *xChannelSelector* to "R" and *yChannelSelector* to "G".

The displacement map defines the inverse of the mapping performed.

The calculations using the pixel values from **in2** are performed using non-premultiplied color values. If the image from **in2** consists of premultiplied color values, those values are automatically converted into non-premultiplied color values before performing this operation.

This filter can have arbitrary non-localized effect on the input which might require substantial buffering in the processing pipeline. However with this formulation, any intermediate buffering needs can be determined by *scale* which represents the maximum range of displacement in either x or y.

When applying this filter, the source pixel location will often lie between several source pixels. In this case it is recommended that high quality viewers apply an interpolent on the surrounding pixels, for example bilinear or bicubic, rather than simply selecting the nearest source pixel. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting.

The **'color-interpolation-filters'** property only applies to the **in2** source image and does not apply to the **in** source image.

```
<!ELEMENT feDisplacementMap (animate|set)* >
<!ATTLIST feDisplacementMap
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  in2 CDATA #REQUIRED
  scale %Number; #IMPLIED
  xChannelSelector (R | G | B | A) "A"
  yChannelSelector (R | G | B | A) "A" >
```

*Attribute definitions:*

**`scale` = "*<number>*"**
: Displacement scale factor. The amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
When the value of this attribute is **0**, this operation has no effect on the source image.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**`xChannelSelector` = "*R* | *G* | *B* | *A*"**
: Indicates which channel from **in2** to use to displace the pixels in **in** along the x-axis.
*Animatable: yes.*

**`yChannelSelector` = "*R* | *G* | *B* | *A*"**
: Indicates which channel from **in2** to use to displace the pixels in **in** along the y-axis.
*Animatable: yes.*

**`in2` = "*(see in attribute)*"**
: The second input image, which is used to displace the pixels in the image from attribute **in**. This attribute can take on the same values as the **in** attribute.
*Animatable: yes.*

***Attributes defined elsewhere:***
: %stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

## 15.16 Filter primitive 'feFlood'

This filter primitive creates a rectangle filled with the color and opacity values from properties **'flood-color'** and **'flood-opacity'**. The rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **'feFlood'** element.

```
<!ELEMENT feFlood (animate|set|animateColor)* >
<!ATTLIST feFlood
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-feFlood;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >
```

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes_with_in;, class, **style**, %PresentationAttributes-Color;, %PresentationAttributes-feFlood;, %PresentationAttributes-FilterPrimitives;.

The **'flood-color'** property indicates what color to use to flood the current filter primitive subregion. The keyword **currentColor** and ICC colors can be specified in the same manner as within a <paint> specification for the **'fill'** and **'stroke'** properties.

**'flood-color'**

| | |
|---|---|
| *Value:* | currentColor \| <color> [icc-color(<name>[,<icccolorvalue>]*)] \| inherit |
| *Initial:* | black |
| *Applies to:* | **'feFlood'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

The **'flood-opacity'** property defines the opacity value to use across the entire filter primitive subregion.

**'flood-opacity'**

| | |
|---|---|
| *Value:* | <alphavalue> \| inherit |
| *Initial:* | 1 |
| *Applies to:* | **'feFlood'** elements |
| *Inherited:* | no |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

## 15.17 Filter primitive 'feGaussianBlur'

This filter primitive performs a Gaussian blur on the input image.

The Gaussian blur kernel is an approximation of the normalized convolution:

$$H(x) = \exp(-x^2/ (2s^2)) / \sqrt{2* \pi * s^2}$$

where 's' is the standard deviation specified by **stdDeviation**.

The value of **stdDeviation** can be either one or two numbers. If two numbers are provided, the first number represents a standard deviation value along the x-axis of the current coordinate system and the second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.

Even if only one value is provided for **stdDeviation**, this can be implemented as a separable convolution.

For larger values of 's' (s >= 2.0), an approximation can be used: Three successive box-blurs build a piece-wise quadratic convolution kernel, which approximates the Gaussian kernel to within roughly 3%.

```
let d = floor(s * 3*sqrt(2*pi)/4 + 0.5)
```

... if d is odd, use three box-blurs of size 'd', centered on the output pixel.

... if d is even, two box-blurs of size 'd' (the first one centered on the pixel boundary between the output pixel and the one to the left, the second one centered on the pixel boundary between the output pixel and the one to the right) and one box blur of size 'd+1' centered on the output pixel.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, **SourceAlpha**. The implementation may notice this and optimize the single channel case. If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

```
<!ELEMENT feGaussianBlur (animate|set)* >
<!ATTLIST feGaussianBlur
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  stdDeviation %NumberOptionalNumber; #IMPLIED >
```

*Attribute definitions:*

**`stdDeviation` = "*<number-optional-number>*"**

The standard deviation for the blur operation. If two <number>s are provided, the first number represents a standard deviation value along the x-axis of the coordinate system established by attribute **primitiveUnits** on the **'filter'** element. The second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

The example at the start of this chapter makes use of the **feGaussianBlur** filter primitive to create a drop shadow effect.

## 15.18 Filter primitive 'feImage'

This filter primitive refers to a graphic external to this filter element, which is loaded or rendered into an RGBA raster and becomes the result of the filter primitive.

This filter primitive can refer to an external image or can be a reference to another piece of SVG. It produces an image similar to the built-in image source **SourceGraphic** except that the graphic comes from an external source.

If the **xlink:href** references a stand-alone image resource such as a JPEG, PNG or SVG file, then the image resource is rendered according to the behavior of the **'image'** element; otherwise, the referenced resource is rendered according to the behavior of the **'use'** element. In either case, the current user coordinate system depends on the value of attribute **primitiveUnits** on the **'filter'** element.

When the referenced image must be resampled to match the device coordinate system, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting.

```
<!ELEMENT feImage (animate|set|animateTransform)* >
<!ATTLIST feImage
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All; >
```

***Attributes defined elsewhere:***

%stdAttrs;, %xlinkRefAttrsEmbed;, xlink:href, %langSpaceAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-All;, %filter_primitive_attributes;.

## 15.19 Filter primitive 'feMerge'

This filter primitive composites input image layers on top of each other using the *over* operator with *Input1* (corresponding to the first **'feMergeNode'** child element) on the bottom and the last specified input, *InputN* (corresponding to the last **'feMergeNode'** child element), on top.

Many effects produce a number of intermediate layers in order to create the final output image. This filter allows us to collapse those into a single image. Although this could be done by using n-1 Composite-filters, it is more convenient to have this common operation available in this form, and offers the implementation some additional flexibility.

Each 'feMerge' element can have any number of 'feMergeNode' subelements, each of which has an **in** attribute.

The canonical implementation of feMerge is to render the entire effect into one RGBA layer, and then render the resulting layer on the output device. In certain cases (in particular if the output device itself is a continuous tone device), and since merging is associative, it might be a sufficient approximation to evaluate the effect one layer at a time and render each layer individually onto the output device bottom to top.

If the topmost image input is **SourceGraphic** and this **'feMerge'** is the last filter primitive in the filter, the implementation is encouraged to render the layers up to that point, and then render the **SourceGraphic** directly from its vector description on top.

```
<!ELEMENT feMerge (feMergeNode)* >
<!ATTLIST feMerge
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes; >

<!ELEMENT feMergeNode (animate|set)* >
<!ATTLIST feMergeNode
  %stdAttrs;
  in CDATA #IMPLIED >
```

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes;, in, %PresentationAttributes-FilterPrimitives;.

The example at the start of this chapter makes use of the **feMerge** filter primitive to composite two intermediate filter results together.

## 15.20 Filter primitive 'feMorphology'

This filter primitive performs "fattening" or "thinning" of artwork. It is particularly useful for fattening or thinning an alpha channel.

The dilation (or erosion) kernel is a rectangle with a width of 2*_x-radius_ and a height of 2*_y-radius_. In dilation, the output pixel is the individual component-wise maximum of the corresponding R,G,B,A values in the input image's kernel rectangle. In erosion, the output pixel is the individual component-wise minimum of the corresponding R,G,B,A values in the input image's kernel rectangle.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, **SourceAlpha**. In that case, the implementation might want to optimize the single channel case.

If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

Because **'feMorphology'** operates on premultipied color values, it will always result in color values less than or equal to the alpha channel.

```
<!ELEMENT feMorphology (animate|set)* >
<!ATTLIST feMorphology
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  operator (erode | dilate) "erode"
  radius %NumberOptionalNumber; #IMPLIED >
```

*Attribute definitions:*

**`operator` = "*erode* | *dilate*"**

A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic.
*Animatable: yes.*

**`radius` = "*<number-optional-number>*"**

The radius (or radii) for the operation. If two <number>s are provided, the first number represents a x-radius and the second value represents a y-radius. If one number is provided, then that value is used for both X and Y. The values are in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).

If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

Example feMorphology shows examples of the four types of feMorphology operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="7cm" viewBox="0 0 700 500"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feMorphology - Examples of erode and dilate</title>
  <desc>Five text strings drawn as outlines.
        The first is unfiltered. The second and third use 'erode'.
        The fourth and fifth use 'dilate'.</desc>
  <defs>
    <filter id="Erode3">
      <feMorphology operator="erode" in="SourceGraphic" radius="3" />
    </filter>
    <filter id="Erode6">
      <feMorphology operator="erode" in="SourceGraphic" radius="6" />
    </filter>
    <filter id="Dilate3">
      <feMorphology operator="dilate" in="SourceGraphic" radius="3" />
    </filter>
    <filter id="Dilate6">
      <feMorphology operator="dilate" in="SourceGraphic" radius="6" />
    </filter>
  </defs>
  <rect fill="none" stroke="blue" stroke-width="2"
        x="1" y="1" width="698" height="498"/>
  <g enable-background="new" >
    <g font-family="Verdana" font-size="75"
              fill="none" stroke="black" stroke-width="6" >
      <text x="50" y="90">Unfiltered</text>
      <text x="50" y="180" filter="url(#Erode3)" >Erode radius 3</text>
      <text x="50" y="270" filter="url(#Erode6)" >Erode radius 6</text>
      <text x="50" y="360" filter="url(#Dilate3)" >Dilate radius 3</text>
      <text x="50" y="450" filter="url(#Dilate6)" >Dilate radius 6</text>
    </g>
  </g>
</svg>
```


***Example feMorphology***

*View this example as SVG (SVG-enabled browsers only)*

## 15.21 Filter primitive 'feOffset'

This filter primitive offsets the input image relative to its current position in the image space by the specified vector.

This is important for effects like drop shadows.

When applying this filter, the destination location may be offset by a fraction of a pixel in device space. In this case a high quality viewer should make use of appropriate interpolation techniques, for example bilinear or bicubic. This is especially recommended for dynamic viewers where this interpolation provides visually smoother movement of images. For static viewers this is less of a concern. Close attention should be made to the **'image-rendering'** property setting to determine the authors intent.

```
<!ELEMENT feOffset (animate|set)* >
<!ATTLIST feOffset
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  dx %Number; #IMPLIED
  dy %Number; #IMPLIED >
```

*Attribute definitions:*

**dx = "*<number>*"**
: The amount to offset the input graphic along the x-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**dy = "*<number>*"**
: The amount to offset the input graphic along the y-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***
: %stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

The example at the start of this chapter makes use of the **feOffset** filter primitive to offset the drop shadow from the original source graphic.

## 15.22 Filter primitive 'feSpecularLighting'

This filter primitive lights a source graphic using the alpha channel as a bump map. The resulting image is an RGBA image based on the light color. The lighting calculation follows the standard specular component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map. The result of the lighting calculation is added. The filter primitive assumes that the viewer is at infinity in the z direction (i.e., the unit vector in the eye direction is (0,0,1) everywhere).

This filter primitive produces an image which contains the specular reflection part of the lighting calculation. Such a map is intended to be combined with a texture using the *add* term of the *arithmetic* **'feComposite'** method. Multiple light sources can be simulated by adding several of these light maps before applying it to the texture image.

The resulting RGBA image is computed as follows:

$$S_r = k_s * pow(N.H, specularExponent) * L_r$$

```
Sg = ks * pow(N.H, specularExponent) * Lg
Sb = ks * pow(N.H, specularExponent) * Lb
Sa = max(Sr, Sg, Sb)
```

where

$k_s$ = specular lighting constant
N = surface normal unit vector, a function of x and y
H = "halfway" unit vector between eye unit vector and light unit vector

$L_r$,$L_g$,$L_b$ = RGB components of light

See **'feDiffuseLighting'** for definition of N and ($L_r$, $L_g$, $L_b$).

The definition of H reflects our assumption of the constant eye vector E = (0,0,1):

```
H = (L + E) / Norm(L+E)
```

where L is the light unit vector.

Unlike the **'feDiffuseLighting'**, the **'feSpecularLighting'** filter produces a non-opaque image. This is due to the fact that the specular result ($S_r$,$S_g$,$S_b$,$S_a$) is meant to be added to the textured image. The alpha channel of the result is the max of the color components, so that where the specular light is zero, no additional coverage is added to the image and a fully white highlight will add opacity.

The **'feDiffuseLighting'** and **'feSpecularLighting'** filters will often be applied together. An implementation may detect this and calculate both maps in one pass, instead of two.

```
<!ELEMENT feSpecularLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feSpecularLighting
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FilterPrimitives;
  %PresentationAttributes-LightingEffects;
  %filter_primitive_attributes_with_in;
  surfaceScale %Number; #IMPLIED
  specularConstant %Number; #IMPLIED
  specularExponent %Number; #IMPLIED
  kernelUnitLength %NumberOptionalNumber; #IMPLIED >
```

*Attribute definitions:*

**`surfaceScale`** = "***<number>***"
: height of surface when $A_{in}$ = 1.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`specularConstant`** = "***<number>***"
: ks in Phong lighting model. In SVG, this can be any non-negative number.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`specularExponent`** = "***<number>***"
: Exponent for specular term, larger is more "shiny". Range 1.0 to 128.0.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes_with_in;, kernelUnitLength, %PresentationAttributes-Color;, %PresentationAttributes-FilterPrimitives;, %PresentationAttributes-LightingEffects;.

The light source is defined by one of the child elements **'feDistantLight'**, **'fePointLight'** or **'feDistantLight'**. The light color is specified by property **'lighting-color'**.

The example at the start of this chapter makes use of the **feSpecularLighting** filter primitive to achieve a highly reflective, 3D glowing effect.

## 15.23 Filter primitive 'feTile'

This filter primitive fills a target rectangle with a repeated, tiled pattern of an input image. The target rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **'feTile'** element.

Typically, the input image has been defined with its own filter primitive subregion in order to define a reference tile. **'feTile'** replicates the reference tile in both X and Y to completely fill the target rectangle. The top/left corner of each given tile is at location `(x+i*width,y+j*height)`, where `(x,y)` represents the top/left of the input image's filter primitive subregion, `width` and `height` represent the width and height of the input image's filter primitive subregion, and `i` and `j` can be any integer value. In most cases, the input image will have a smaller filter primitive subregion than the **'feTile'** in order to achieve a repeated pattern effect.

Implementers must take appropriate measures in constructing the tiled image to avoid artifacts between tiles, particularly in situations where the user to device transform includes shear and/or rotation. Unless care is taken, interpolation can lead to edge pixels in the tile having opacity values lower or higher than expected due to the interaction of painting adjacent tiles which each have partial overlap with particular pixels.

```
<!ELEMENT feTile (animate|set)* >
<!ATTLIST feTile
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >
```

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

## 15.24 Filter primitive 'feTurbulence'

This filter primitive creates an image using the Perlin turbulence function. It allows the synthesis of artificial textures like clouds or marble. For a detailed description the of the Perlin turbulence function, see "Texturing and Modeling", Ebert et al, AP Professional, 1994. The resulting image will fill the entire filter primitive subregion for this filter primitive.

It is possible to create bandwidth-limited noise by synthesizing only one octave.

The C code below shows the exact algorithm used for this filter effect.

For fractalSum, you get a turbFunctionResult that is aimed at a range of -1 to 1 (the actual result might exceed this range in some cases). To convert to a color value, use the formula `colorValue = ((turbFunctionResult * 255) + 255) / 2`, then clamp to the range 0 to 255.

For turbulence, you get a turbFunctionResult that is aimed at a range of 0 to 1 (the actual result might exceed this range in some cases). To convert to a color value, use the formula `colorValue = (turbFunctionResult * 255)`, then clamp to the range 0 to 255.

The following order is used for applying the pseudo random numbers. An initial seed value is computed based on attribute **seed**. Then the implementation computes the lattice points for R, then continues getting additional pseudo random numbers relative to the last generated pseudo random number and computes the lattice points for G, and so on for B and A.

The generated color and alpha values are in the color space determined by the value of property **'color-interpolation-filters'**:

```
/* Produces results in the range [1, 2**31 - 2].
Algorithm is: r = (a * r) mod m
where a = 16807 and m = 2**31 - 1 = 2147483647
See [Park & Miller], CACM vol. 31 no. 10 p. 1195, Oct. 1988
To test: the algorithm should produce the result 1043618065
as the 10,000th generated number if the original seed is 1.
*/
#define RAND_m 2147483647 /* 2**31 - 1 */
#define RAND_a 16807 /* 7**5; primitive root of m */
#define RAND_q 127773 /* m / a */
#define RAND_r 2836 /* m % a */

long setup_seed(long lSeed)
{
  if (lSeed <= 0) lSeed = -(lSeed % (RAND_m - 1)) + 1;
  if (lSeed > RAND_m - 1) lSeed = RAND_m - 1;
  return lSeed;
}

long random(long lSeed)
{
  long result;
  result = RAND_a * (lSeed % RAND_q) - RAND_r * (lSeed / RAND_q);
  if (result <= 0) result += RAND_m;
  return result;
}

#define BSize 0x100
#define BM 0xff
#define PerlinN 0x1000
#define NP 12 /* 2^PerlinN */
#define NM 0xfff
static uLatticeSelector[BSize + BSize + 2];
static double fGradient[4][BSize + BSize + 2][2];
struct StitchInfo
{
  int nWidth; // How much to subtract to wrap for stitching.
  int nHeight;
  int nWrapX; // Minimum value to wrap.
  int nWrapY;
};

static void init(long lSeed)
{
  double s;
  int i, j, k;
  lSeed = setup_seed(lSeed);
  for(k = 0; k < 4; k++)
  {
    for(i = 0; i < BSize; i++)
    {
      uLatticeSelector[i] = i;
      for (j = 0; j < 2; j++)
        fGradient[k][i][j] = (double)(((lSeed = random(lSeed)) % (BSize + BSize)) - BSize) / BSize;
      s = double(sqrt(fGradient[k][i][0] * fGradient[k][i][0] + fGradient[k][i][1] * fGradient[k][i][1]));
      fGradient[k][i][0] /= s;
      fGradient[k][i][1] /= s;
    }
  }
  while(--i)
  {
    k = uLatticeSelector[i];
    uLatticeSelector[i] = uLatticeSelector[j = (lSeed = random(lSeed)) % BSize];
    uLatticeSelector[j] = k;
  }
  for(i = 0; i < BSize + 2; i++)
  {
    uLatticeSelector[BSize + i] = uLatticeSelector[i];
    for(k = 0; k < 4; k++)
      for(j = 0; j < 2; j++)
        fGradient[k][BSize + i][j] = fGradient[k][i][j];
  }
}

#define s_curve(t) ( t * t * (3. - 2. * t) )
#define lerp(t, a, b) ( a + t * (b - a) )
double noise2(int nColorChannel, double vec[2], StitchInfo *pStitchInfo)
{
  int bx0, bx1, by0, by1, b00, b10, b01, b11;
  double rx0, rx1, ry0, ry1, *q, sx, sy, a, b, t, u, v;
```

```
  register i, j;
  t = vec[0] + PerlinN;
  bx0 = (int)t;
  bx1 = bx0+1;
  rx0 = t - (int)t;
  rx1 = rx0 - 1.0f;
  t = vec[1] + PerlinN;
  by0 = (int)t;
  by1 = by0+1;
  ry0 = t - (int)t;
  ry1 = ry0 - 1.0f;

  // If stitching, adjust lattice points accordingly.
  if(pStitchInfo != NULL)
  {
    if(bx0 >= pStitchInfo->nWrapX)
      bx0 -= pStitchInfo->nWidth;
    if(bx1 >= pStitchInfo->nWrapX)
      bx1 -= pStitchInfo->nWidth;
    if(by0 >= pStitchInfo->nWrapY)
      by0 -= pStitchInfo->nHeight;
    if(by1 >= pStitchInfo->nWrapY)
      by1 -= pStitchInfo->nHeight;
  }

  bx0 &= BM;
  bx1 &= BM;
  by0 &= BM;
  by1 &= BM;

  i = uLatticeSelector[bx0];
  j = uLatticeSelector[bx1];
  b00 = uLatticeSelector[i + by0];
  b10 = uLatticeSelector[j + by0];
  b01 = uLatticeSelector[i + by1];
  b11 = uLatticeSelector[j + by1];
  sx = double(s_curve(rx0));
  sy = double(s_curve(ry0));
  q = fGradient[nColorChannel][b00]; u = rx0 * q[0] + ry0 * q[1];
  q = fGradient[nColorChannel][b10]; v = rx1 * q[0] + ry0 * q[1];
  a = lerp(sx, u, v);
  q = fGradient[nColorChannel][b01]; u = rx0 * q[0] + ry1 * q[1];
  q = fGradient[nColorChannel][b11]; v = rx1 * q[0] + ry1 * q[1];
  b = lerp(sx, u, v);
  return lerp(sy, a, b);
}

double turbulence(int nColorChannel, double *point, double fBaseFreqX, double fBaseFreqY,
          int nNumOctaves, bool bFractalSum, bool bDoStitching,
          double fTileX, double fTileY, double fTileWidth, double fTileHeight)
{
  StitchInfo stitch;
  StitchInfo *pStitchInfo = NULL; // Not stitching when NULL.

  // Adjust the base frequencies if necessary for stitching.
  if(bDoStitching)
  {
    // When stitching tiled turbulence, the frequencies must be adjusted
    // so that the tile borders will be continuous.
    if(fBaseFreqX != 0.0)
    {
      double fLoFreq = double(floor(fTileWidth * fBaseFreqX)) / fTileWidth;
      double fHiFreq = double(ceil(fTileWidth * fBaseFreqX)) / fTileWidth;
      if(fBaseFreqX / fLoFreq < fHiFreq / fBaseFreqX)
        fBaseFreqX = fLoFreq;
      else
        fBaseFreqX = fHiFreq;
    }

    if(fBaseFreqY != 0.0)
    {
      double fLoFreq = double(floor(fTileHeight * fBaseFreqY)) / fTileHeight;
      double fHiFreq = double(ceil(fTileHeight * fBaseFreqY)) / fTileHeight;
      if(fBaseFreqY / fLoFreq < fHiFreq / fBaseFreqY)
        fBaseFreqY = fLoFreq;
      else
        fBaseFreqY = fHiFreq;
    }

    // Set up initial stitch values.
    pStitchInfo = &stitch;
    stitch.nWidth = int(fTileWidth * fBaseFreqX + 0.5f);
    stitch.nWrapX = fTileX * fBaseFreqX + PerlinN + stitch.nWidth;
    stitch.nHeight = int(fTileHeight * fBaseFreqY + 0.5f);
    stitch.nWrapY = fTileY * fBaseFreqY + PerlinN + stitch.nHeight;
  }

  double fSum = 0.0f;
  double vec[2];
  vec[0] = point[0] * fBaseFreqX;
  vec[1] = point[1] * fBaseFreqY;
  double ratio = 1;
  for(int nOctave = 0; nOctave < nNumOctaves; nOctave++)
  {
    if(bFractalSum)
      fSum += double(noise2(nColorChannel, vec, pStitchInfo) / ratio);
```

```
          else
            fSum += double(fabs(noise2(nColorChannel, vec, pStitchInfo)) / ratio);

          vec[0] *= 2;
          vec[1] *= 2;
          ratio *= 2;

          if(pStitchInfo != NULL)
          {
            // Update stitch values. Subtracting PerlinN before the multiplication and
            // adding it afterward simplifies to subtracting it once.
            stitch.nWidth *= 2;
            stitch.nWrapX = 2 * stitch.nWrapX - PerlinN;
            stitch.nHeight *= 2;
            stitch.nWrapY = 2 * stitch.nWrapY - PerlinN;
          }
        }
        return fSum;
      }
```

```
<!ELEMENT feTurbulence (animate|set)* >
<!ATTLIST feTurbulence
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes;
  baseFrequency %NumberOptionalNumber; #IMPLIED
  numOctaves %Integer; #IMPLIED
  seed %Number; #IMPLIED
  stitchTiles (stitch | noStitch) "noStitch"
  type (fractalNoise | turbulence) "turbulence" >
```

*Attribute definitions:*

**`baseFrequency` = "*<number-optional-number>*"**

The base frequency (frequencies) parameter(s) for the noise function. If two <number>s are provided, the first number represents a base frequency in the X direction and the second value represents a base frequency in the Y direction. If one number is provided, then that value is used for both X and Y.
A negative value for base frequency is an error (see Error processing).
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**`numOctaves` = "*<integer>*"**

The numOctaves parameter for the noise function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
*Animatable: yes.*

**`seed` = "*<number>*"**

The starting number for the pseudo random number generator.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
*Animatable: yes.*

**`stitchTiles` = "*stitch* | *noStitch*"**

If **stitchTiles="noStitch"**, no attempt it made to achieve smooth transitions at the border of tiles which contain a turbulence function. Sometimes the result will show clear discontinuities at the tile borders.
If **stitchTiles="stitch"**, then the user agent will automatically adjust baseFrequency-x and baseFrequency-y values such that the feTurbulence node's width and height (i.e., the width and height of the current subregion) contains an integral number of the Perlin tile width and height for the first octave. The baseFrequency will be adjusted up or down depending on which way has the smallest relative (not absolute) change as follows: Given the frequency, calculate `lowFreq=floor(width*frequency)/width` and `hiFreq=ceil(width*frequency)/width`. If frequency/lowFreq < hiFreq/frequency then use lowFreq, else use hiFreq. While generating turbulence values, generate lattice vectors as normal for Perlin Noise, except for those lattice points that lie on the right or bottom edges of the active area (the size of the resulting tile). In those cases, copy the lattice vector from the opposite edge of the active area.
*Animatable: yes.*

**`type` = "*fractalNoise* | *turbulence*"**

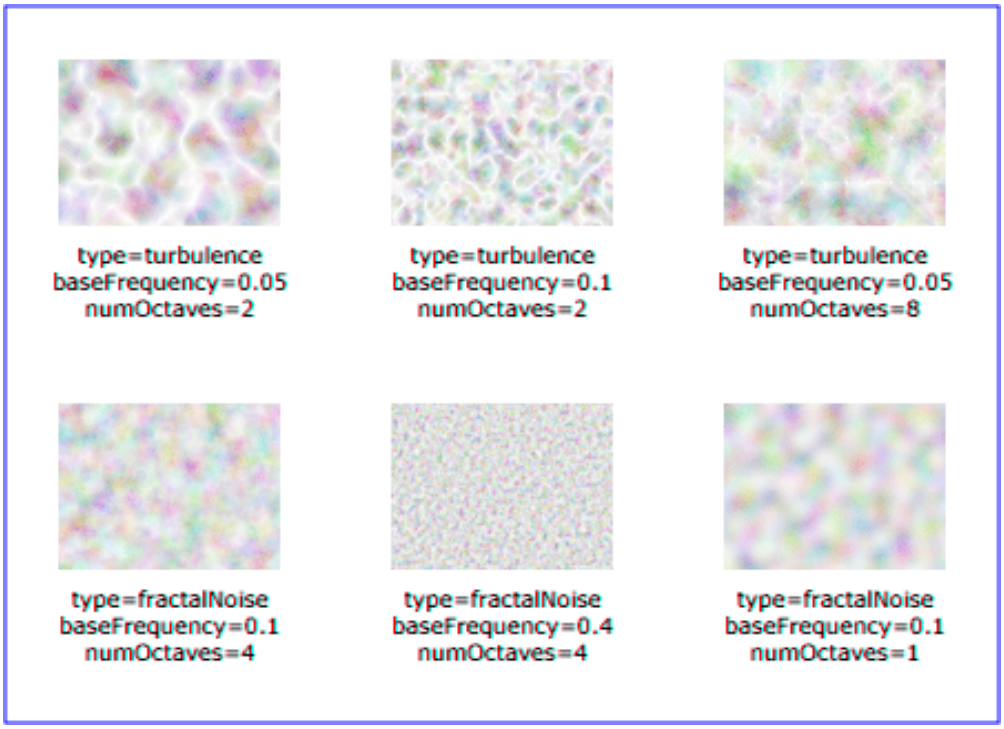
Indicates whether the filter primitive should perform a noise or turbulence function.
*Animatable: yes.*

***Attributes defined elsewhere:***

%stdAttrs;, %filter_primitive_attributes;, %PresentationAttributes-FilterPrimitives;.

Example feTurbulence shows the effects of various parameter settings for feTurbulence.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="450px" height="325px" viewBox="0 0 450 325"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feTurbulence - Examples of feTurbulence operations</title>
  <desc>Six rectangular areas showing the effects of
        various parameter settings for feTurbulence.</desc>
  <g  font-family="Verdana" text-anchor="middle" font-size="10" >
    <defs>
      <filter id="Turb1" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.05" numOctaves="2"/>
      </filter>
      <filter id="Turb2" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.1" numOctaves="2"/>
      </filter>
      <filter id="Turb3" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.05" numOctaves="8"/>
      </filter>
      <filter id="Turb4" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.1" numOctaves="4"/>
      </filter>
      <filter id="Turb5" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.4" numOctaves="4"/>
      </filter>
      <filter id="Turb6" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.1" numOctaves="1"/>
      </filter>
    </defs>

    <rect x="1" y="1" width="448" height="323"
          fill="none" stroke="blue" stroke-width="1"  />

    <rect x="25" y="25" width="100" height="75" filter="url(#Turb1)"  />
    <text x="75" y="117">type=turbulence</text>
    <text x="75" y="129">baseFrequency=0.05</text>
    <text x="75" y="141">numOctaves=2</text>

    <rect x="175" y="25" width="100" height="75" filter="url(#Turb2)"  />
    <text x="225" y="117">type=turbulence</text>
    <text x="225" y="129">baseFrequency=0.1</text>
    <text x="225" y="141">numOctaves=2</text>

    <rect x="325" y="25" width="100" height="75" filter="url(#Turb3)"  />
    <text x="375" y="117">type=turbulence</text>
    <text x="375" y="129">baseFrequency=0.05</text>
    <text x="375" y="141">numOctaves=8</text>

    <rect x="25" y="180" width="100" height="75" filter="url(#Turb4)"  />
    <text x="75" y="272">type=fractalNoise</text>
    <text x="75" y="284">baseFrequency=0.1</text>
    <text x="75" y="296">numOctaves=4</text>

    <rect x="175" y="180" width="100" height="75" filter="url(#Turb5)"  />
    <text x="225" y="272">type=fractalNoise</text>
    <text x="225" y="284">baseFrequency=0.4</text>
    <text x="225" y="296">numOctaves=4</text>

    <rect x="325" y="180" width="100" height="75" filter="url(#Turb6)"  />
    <text x="375" y="272">type=fractalNoise</text>
    <text x="375" y="284">baseFrequency=0.1</text>
    <text x="375" y="296">numOctaves=1</text>
  </g>
</svg>
```




***Example feTurbulence***

*View this example as SVG (SVG-enabled browsers only)*

## 15.25 DOM interfaces

The following interfaces are defined below: SVGFilterElement, SVGFilterPrimitiveStandardAttributes, SVGFEBlendElement, SVGFEColorMatrixElement, SVGFEComponentTransferElement, SVGComponentTransferFunctionElement, SVGFEFuncRElement, SVGFEFuncGElement, SVGFEFuncBElement, SVGFEFuncAElement, SVGFECompositeElement, SVGFEConvolveMatrixElement, SVGFEDiffuseLightingElement, SVGFEDistantLightElement, SVGFEPointLightElement, SVGFESpotLightElement, SVGFEDisplacementMapElement, SVGFEFloodElement, SVGFEGaussianBlurElement, SVGFEImageElement, SVGFEMergeElement, SVGFEMergeNodeElement, SVGFEMorphologyElement, SVGFEOffsetElement, SVGFESpecularLightingElement, SVGFETileElement, SVGFETurbulenceElement.

### Interface SVGFilterElement

The **SVGFilterElement** interface corresponds to the **'filter'** element.

**IDL Definition**

```
interface SVGFilterElement :
                SVGElement,
                SVGURIReference,
                SVGLangSpace,
                SVGExternalResourcesRequired,
```

```
                SVGStylable,
                SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration filterUnits;
  readonly attribute SVGAnimatedEnumeration primitiveUnits;
  readonly attribute SVGAnimatedLength      x;
  readonly attribute SVGAnimatedLength      y;
  readonly attribute SVGAnimatedLength      width;
  readonly attribute SVGAnimatedLength      height;
  readonly attribute SVGAnimatedInteger    filterResX;
  readonly attribute SVGAnimatedInteger    filterResY;

  void setFilterRes ( in unsigned long filterResX, in unsigned long filterResY );
};
```

**Attributes**

**readonly SVGAnimatedEnumeration filterUnits**
Corresponds to attribute **filterUnits** on the given **'filter'** element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedEnumeration primitiveUnits**
Corresponds to attribute **primitiveUnits** on the given **'filter'** element. Takes one of the constants defined in **SVGUnitTypes**.

**readonly SVGAnimatedLength x**
Corresponds to attribute **x** on the given **'filter'** element.

**readonly SVGAnimatedLength y**
Corresponds to attribute **y** on the given **'filter'** element.

**readonly SVGAnimatedLength width**
Corresponds to attribute **width** on the given **'filter'** element.

**readonly SVGAnimatedLength height**
Corresponds to attribute **height** on the given **'filter'** element.

**readonly SVGAnimatedInteger filterResX**
Corresponds to attribute **filterRes** on the given **'filter'** element. Contains the X component of attribute **filterRes**.

**readonly SVGAnimatedInteger filterResY**
Corresponds to attribute **filterRes** on the given **'filter'** element. Contains the Y component (possibly computed automatically) of attribute **filterRes**.

**Methods**

**setFilterRes**
Sets the values for attribute **filterRes**.

**Parameters**

| | |
|---|---|
| in unsigned long filterResX | The X component of attribute **filterRes**. |
| in unsigned long filterResY | The Y component of attribute **filterRes**. |

**No Return Value**
**No Exceptions**

## Interface SVGFilterPrimitiveStandardAttributes

This interface defines the set of DOM attributes that are common across the filter interfaces.

**IDL Definition**

```
interface SVGFilterPrimitiveStandardAttributes : SVGStylable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};
```

**Attributes**

**readonly SVGAnimatedLength x**
Corresponds to attribute **x** on the given element.
**readonly SVGAnimatedLength y**
Corresponds to attribute **y** on the given element.
**readonly SVGAnimatedLength width**
Corresponds to attribute **width** on the given element.
**readonly SVGAnimatedLength height**
Corresponds to attribute **height** on the given element.
**readonly SVGAnimatedString result**
Corresponds to attribute **result** on the given element.

## Interface SVGFEBlendElement

The **SVGFEBlendElement** interface corresponds to the **'feBlend'** element.

**IDL Definition**

```
interface SVGFEBlendElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Blend Mode Types
  const unsigned short SVG_FEBLEND_MODE_UNKNOWN  = 0;
  const unsigned short SVG_FEBLEND_MODE_NORMAL   = 1;
  const unsigned short SVG_FEBLEND_MODE_MULTIPLY = 2;
  const unsigned short SVG_FEBLEND_MODE_SCREEN   = 3;
  const unsigned short SVG_FEBLEND_MODE_DARKEN   = 4;
  const unsigned short SVG_FEBLEND_MODE_LIGHTEN  = 5;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration mode;
};
```

**Definition group Blend Mode Types**

**Defined constants**

| | |
|---|---|
| SVG_FEBLEND_MODE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_FEBLEND_MODE_NORMAL | Corresponds to value **normal**. |
| SVG_FEBLEND_MODE_MULTIPLY | Corresponds to value **multiply**. |
| SVG_FEBLEND_MODE_SCREEN | Corresponds to value **screen**. |
| SVG_FEBLEND_MODE_DARKEN | Corresponds to value **darken**. |
| SVG_FEBLEND_MODE_LIGHTEN | Corresponds to value **lighten**. |

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feBlend'** element.
**readonly SVGAnimatedString in2**
Corresponds to attribute **in2** on the given **'feBlend'** element.
**readonly SVGAnimatedEnumeration mode**
Corresponds to attribute **mode** on the given **'feBlend'** element. Takes one of the Blend Mode Types.

## Interface SVGFEColorMatrixElement

The **SVGFEColorMatrixElement** interface corresponds to the **'feColorMatrix'** element.

**IDL Definition**

```
interface SVGFEColorMatrixElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Color Matrix Types
  const unsigned short SVG_FECOLORMATRIX_TYPE_UNKNOWN          = 0;
  const unsigned short SVG_FECOLORMATRIX_TYPE_MATRIX           = 1;
  const unsigned short SVG_FECOLORMATRIX_TYPE_SATURATE         = 2;
  const unsigned short SVG_FECOLORMATRIX_TYPE_HUEROTATE        = 3;
  const unsigned short SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA = 4;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  values;
};
```

**Definition group Color Matrix Types**

**Defined constants**

| | |
|---|---|
| SVG_FECOLORMATRIX_TYPE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_FECOLORMATRIX_TYPE_MATRIX | Corresponds to value **matrix**. |
| SVG_FECOLORMATRIX_TYPE_SATURATE | Corresponds to value **saturate**. |
| SVG_FECOLORMATRIX_TYPE_HUEROTATE | Corresponds to value **hueRotate**. |
| SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA | Corresponds to value **luminanceToAlpha**. |

**Attributes**

**readonly SVGAnimatedString in1**

Corresponds to attribute **in** on the given **'feColorMatrix'** element.

**readonly SVGAnimatedEnumeration type**

Corresponds to attribute **type** on the given **'feColorMatrix'** element. Takes one of the Color Matrix Types.

**readonly SVGAnimatedNumberList values**

Corresponds to attribute **values** on the given **'feColorMatrix'** element.

Provides access to the contents of the **values** attribute.

## Interface SVGFEComponentTransferElement

The **SVGFEComponentTransferElement** interface corresponds to the **'feComponentTransfer'** element.

**IDL Definition**

```
interface SVGFEComponentTransferElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
};
```

**Attributes**

**readonly SVGAnimatedString in1**

Corresponds to attribute **in** on the given **'feComponentTransfer'** element.

## Interface SVGComponentTransferFunctionElement

This interface defines a base interface used by the component transfer function interfaces.

**IDL Definition**

```
interface SVGComponentTransferFunctionElement : SVGElement {
  // Component Transfer Types
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN  = 0;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_IDENTITY = 1;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_TABLE    = 2;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE    = 3;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_LINEAR   = 4;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_GAMMA    = 5;

  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  tableValues;
  readonly attribute SVGAnimatedNumber      slope;
  readonly attribute SVGAnimatedNumber      intercept;
  readonly attribute SVGAnimatedNumber      amplitude;
  readonly attribute SVGAnimatedNumber      exponent;
  readonly attribute SVGAnimatedNumber      offset;
};
```

**Definition group Component Transfer Types**

**Defined constants**

| | |
|---|---|
| SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_FECOMPONENTTRANSFER_TYPE_IDENTITY | Corresponds to value **identity**. |
| SVG_FECOMPONENTTRANSFER_TYPE_TABLE | Corresponds to value **table**. |
| SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE | Corresponds to value **discrete**. |
| SVG_FECOMPONENTTRANSFER_TYPE_LINEAR | Corresponds to value **linear**. |
| SVG_FECOMPONENTTRANSFER_TYPE_GAMMA | Corresponds to value **gamma**. |

**Attributes**

**readonly SVGAnimatedEnumeration type**
Corresponds to attribute **type** on the given element. Takes one of the Component Transfer Types.

**readonly SVGAnimatedNumberList tableValues**
Corresponds to attribute **tableValues** on the given element.

**readonly SVGAnimatedNumber slope**
Corresponds to attribute **slope** on the given element.

**readonly SVGAnimatedNumber intercept**
Corresponds to attribute **intercept** on the given element.

**readonly SVGAnimatedNumber amplitude**
Corresponds to attribute **amplitude** on the given element.

**readonly SVGAnimatedNumber exponent**
Corresponds to attribute **exponent** on the given element.

**readonly SVGAnimatedNumber offset**
Corresponds to attribute **offset** on the given element.

## Interface SVGFEFuncRElement

The **SVGFEFuncRElement** interface corresponds to the **'feFuncR'** element.

**IDL Definition**

```
interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {};
```

## Interface SVGFEFuncGElement

The **SVGFEFuncGElement** interface corresponds to the **'feFuncG'** element.

**IDL Definition**

```
interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {};
```

## Interface SVGFEFuncBElement

The **SVGFEFuncBElement** interface corresponds to the **'feFuncB'** element.

**IDL Definition**

```
interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {};
```

## Interface SVGFEFuncAElement

The **SVGFEFuncAElement** interface corresponds to the **'feFuncA'** element.

**IDL Definition**

```
interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {};
```

## Interface SVGFECompositeElement

The **SVGFECompositeElement** interface corresponds to the **'feComposite'** element.

**IDL Definition**

```
interface SVGFECompositeElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Composite Operators
  const unsigned short SVG_FECOMPOSITE_OPERATOR_UNKNOWN    = 0;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OVER       = 1;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_IN         = 2;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OUT        = 3;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ATOP       = 4;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_XOR        = 5;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ARITHMETIC = 6;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedNumber      k1;
  readonly attribute SVGAnimatedNumber      k2;
  readonly attribute SVGAnimatedNumber      k3;
  readonly attribute SVGAnimatedNumber      k4;
};
```

**Definition group Composite Operators**

**Defined constants**

| | |
|---|---|
| SVG_FECOMPOSITE_OPERATOR_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_FECOMPOSITE_OPERATOR_OVER | Corresponds to value **over**. |
| SVG_FECOMPOSITE_OPERATOR_IN | Corresponds to value **in**. |
| SVG_FECOMPOSITE_OPERATOR_OUT | Corresponds to value **out**. |
| SVG_FECOMPOSITE_OPERATOR_ATOP | Corresponds to value **atop**. |
| SVG_FECOMPOSITE_OPERATOR_XOR | Corresponds to value **xor**. |
| SVG_FECOMPOSITE_OPERATOR_ARITHMETIC | Corresponds to value **arithmetic**. |

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feComposite'** element.

**readonly SVGAnimatedString in2**
Corresponds to attribute **in2** on the given **'feComposite'** element.

**readonly SVGAnimatedEnumeration operator**
Corresponds to attribute **operator** on the given **'feComposite'** element. Takes one of the Composite Operators.

**readonly SVGAnimatedNumber k1**
Corresponds to attribute **k1** on the given **'feComposite'** element.

**readonly SVGAnimatedNumber k2**
Corresponds to attribute **k2** on the given **'feComposite'** element.

**readonly SVGAnimatedNumber k3**
Corresponds to attribute **k3** on the given **'feComposite'** element.

**readonly SVGAnimatedNumber k4**
Corresponds to attribute **k4** on the given **'feComposite'** element.

## Interface SVGFEConvolveMatrixElement

The **SVGFEConvolveMatrixElement** interface corresponds to the **'feConvolveMatrix'** element.

**IDL Definition**

```
interface SVGFEConvolveMatrixElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Edge Mode Values
  const unsigned short SVG_EDGEMODE_UNKNOWN   = 0;
  const unsigned short SVG_EDGEMODE_DUPLICATE = 1;
  const unsigned short SVG_EDGEMODE_WRAP      = 2;
  const unsigned short SVG_EDGEMODE_NONE      = 3;

  readonly attribute SVGAnimatedInteger     orderX;
  readonly attribute SVGAnimatedInteger     orderY;
  readonly attribute SVGAnimatedNumberList  kernelMatrix;
  readonly attribute SVGAnimatedNumber      divisor;
  readonly attribute SVGAnimatedNumber      bias;
  readonly attribute SVGAnimatedInteger     targetX;
  readonly attribute SVGAnimatedInteger     targetY;
  readonly attribute SVGAnimatedEnumeration edgeMode;
  readonly attribute SVGAnimatedLength      kernelUnitLengthX;
  readonly attribute SVGAnimatedLength      kernelUnitLengthY;
  readonly attribute SVGAnimatedBoolean     preserveAlpha;
};
```

**Definition group Edge Mode Values**

**Defined constants**

| | |
|---|---|
| SVG_EDGEMODE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_EDGEMODE_DUPLICATE | Corresponds to value **duplicate**. |
| SVG_EDGEMODE_WRAP | Corresponds to value **wrap**. |
| SVG_EDGEMODE_NONE | Corresponds to value **none**. |

**Attributes**

**readonly SVGAnimatedInteger orderX**
Corresponds to attribute **order** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedInteger orderY**
Corresponds to attribute **order** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedNumberList kernelMatrix**
Corresponds to attribute **kernelMatrix** on the given element.

**readonly SVGAnimatedNumber divisor**
Corresponds to attribute **divisor** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedNumber bias**
Corresponds to attribute **bias** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedInteger targetX**
Corresponds to attribute **targetX** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedInteger targetY**
Corresponds to attribute **targetY** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedEnumeration edgeMode**
Corresponds to attribute **edgeMode** on the given **'feConvolveMatrix'** element. Takes one of the Edge Mode Types.

**readonly SVGAnimatedLength kernelUnitLengthX**
Corresponds to attribute **kernelUnitLength** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedLength kernelUnitLengthY**
Corresponds to attribute **kernelUnitLength** on the given **'feConvolveMatrix'** element.

**readonly SVGAnimatedBoolean preserveAlpha**
Corresponds to attribute **preserveAlpha** on the given **'feConvolveMatrix'** element.

## Interface SVGFEDiffuseLightingElement

The **SVGFEDiffuseLightingElement** interface corresponds to the **'feDiffuseLighting'** element.

**IDL Definition**

```
interface SVGFEDiffuseLightingElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
};
```

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feDiffuseLighting'** element.

**readonly SVGAnimatedNumber surfaceScale**
Corresponds to attribute **surfaceScale** on the given **'feDiffuseLighting'** element.

**readonly SVGAnimatedNumber diffuseConstant**
Corresponds to attribute **diffuseConstant** on the given **'feDiffuseLighting'** element.

## Interface SVGFEDistantLightElement

The **SVGFEDistantLightElement** interface corresponds to the **'feDistantLight'** element.

**IDL Definition**

```
interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};
```

**Attributes**

**readonly SVGAnimatedNumber azimuth**
Corresponds to attribute **azimuth** on the given **'feDistantLight'** element.

**readonly SVGAnimatedNumber elevation**
Corresponds to attribute **elevation** on the given **'feDistantLight'** element.

## Interface SVGFEPointLightElement

The **SVGFEPointLightElement** interface corresponds to the **'fePointLight'** element.

**IDL Definition**

```
interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};
```

**Attributes**

**readonly SVGAnimatedNumber x**
Corresponds to attribute **x** on the given **'fePointLight'** element.
**readonly SVGAnimatedNumber y**
Corresponds to attribute **y** on the given **'fePointLight'** element.
**readonly SVGAnimatedNumber z**
Corresponds to attribute **z** on the given **'fePointLight'** element.

## Interface SVGFESpotLightElement

The **SVGFESpotLightElement** interface corresponds to the **'feSpotLight'** element.

**IDL Definition**

```
interface SVGFESpotLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
  readonly attribute SVGAnimatedNumber pointsAtX;
  readonly attribute SVGAnimatedNumber pointsAtY;
  readonly attribute SVGAnimatedNumber pointsAtZ;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber limitingConeAngle;
};
```

**Attributes**
**readonly SVGAnimatedNumber x**
Corresponds to attribute **x** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber y**
Corresponds to attribute **y** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber z**
Corresponds to attribute **z** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber pointsAtX**
Corresponds to attribute **pointsAtX** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber pointsAtY**
Corresponds to attribute **pointsAtY** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber pointsAtZ**
Corresponds to attribute **pointsAtZ** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber specularExponent**
Corresponds to attribute **specularExponent** on the given **'feSpotLight'** element.
**readonly SVGAnimatedNumber limitingConeAngle**
Corresponds to attribute **limitingConeAngle** on the given **'feSpotLight'** element.

## Interface SVGFEDisplacementMapElement

The **SVGFEDisplacementMapElement** interface corresponds to the **'feDisplacementMap'** element.

**IDL Definition**

```
interface SVGFEDisplacementMapElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Channel Selectors
  const unsigned short SVG_CHANNEL_UNKNOWN = 0;
  const unsigned short SVG_CHANNEL_R       = 1;
  const unsigned short SVG_CHANNEL_G       = 2;
```

```
    const unsigned short SVG_CHANNEL_B       = 3;
    const unsigned short SVG_CHANNEL_A       = 4;

    readonly attribute SVGAnimatedString      in1;
    readonly attribute SVGAnimatedString      in2;
    readonly attribute SVGAnimatedNumber      scale;
    readonly attribute SVGAnimatedEnumeration xChannelSelector;
    readonly attribute SVGAnimatedEnumeration yChannelSelector;
  };
```

**Definition group Channel Selectors**

**Defined constants**

| | |
|---|---|
| SVG_CHANNEL_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_CHANNEL_R | Corresponds to value **R**. |
| SVG_CHANNEL_G | Corresponds to value **G**. |
| SVG_CHANNEL_B | Corresponds to value **B**. |
| SVG_CHANNEL_A | Corresponds to value **A**. |

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feDisplacementMap'** element.

**readonly SVGAnimatedString in2**
Corresponds to attribute **in2** on the given **'feDisplacementMap'** element.

**readonly SVGAnimatedNumber scale**
Corresponds to attribute **scale** on the given **'feDisplacementMap'** element.

**readonly SVGAnimatedEnumeration xChannelSelector**
Corresponds to attribute **xChannelSelector** on the given **'feDisplacementMap'** element. Takes one of the Channel Selectors.

**readonly SVGAnimatedEnumeration yChannelSelector**
Corresponds to attribute **yChannelSelector** on the given **'feDisplacementMap'** element. Takes one of the Channel Selectors.

## Interface SVGFEFloodElement

The **SVGFEFloodElement** interface corresponds to the **'feFlood'** element.

**IDL Definition**

```
interface SVGFEFloodElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString      in1;
};
```

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feFlood'** element.

## Interface SVGFEGaussianBlurElement

The **SVGFEGaussianBlurElement** interface corresponds to the **'feGaussianBlur'** element.

**IDL Definition**

```
interface SVGFEGaussianBlurElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;

  void setStdDeviation ( in float stdDeviationX, in float stdDeviationY );
};
```

**Attributes**

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feGaussianBlur'** element.

**readonly SVGAnimatedNumber stdDeviationX**
Corresponds to attribute **stdDeviation** on the given **'feGaussianBlur'** element. Contains the X component of attribute **stdDeviation**.

**readonly SVGAnimatedNumber stdDeviationY**
Corresponds to attribute **stdDeviation** on the given **'feGaussianBlur'** element. Contains the Y component (possibly computed automatically) of attribute **stdDeviation**.

**Methods**

**setStdDeviation**
Sets the values for attribute **stdDeviation**.

**Parameters**

| | |
|---|---|
| in float stdDeviationX | The X component of attribute **stdDeviation**. |
| in float stdDeviationY | The Y component of attribute **stdDeviation**. |

**No Return Value**
**No Exceptions**

## Interface SVGFEImageElement

The **SVGFEImageElement** interface corresponds to the **'feImage'** element.

**IDL Definition**

```
interface SVGFEImageElement :
               SVGElement,
               SVGURIReference,
               SVGLangSpace,
               SVGExternalResourcesRequired,
               SVGFilterPrimitiveStandardAttributes {};
```

## Interface SVGFEMergeElement

The **SVGFEMergeElement** interface corresponds to the **'feMerge'** element.

**IDL Definition**

```
interface SVGFEMergeElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {};
```

## Interface SVGFEMergeNodeElement

The **SVGFEMergeNodeElement** interface corresponds to the **'feMergeNode'** element.

### IDL Definition

```
interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};
```

### Attributes

**readonly SVGAnimatedString in1**

Corresponds to attribute **in** on the given **'feMergeNode'** element.

## Interface SVGFEMorphologyElement

The **SVGFEMorphologyElement** interface corresponds to the **'feMorphology'** element.

### IDL Definition

```
interface SVGFEMorphologyElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Morphology Operators
  const unsigned short SVG_MORPHOLOGY_OPERATOR_UNKNOWN = 0;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_ERODE   = 1;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_DILATE  = 2;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedLength      radiusX;
  readonly attribute SVGAnimatedLength      radiusY;
};
```

### Definition group Morphology Operators

#### Defined constants

| | |
|---|---|
| SVG_MORPHOLOGY_OPERATOR_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_MORPHOLOGY_OPERATOR_ERODE | Corresponds to value **erode**. |
| SVG_MORPHOLOGY_OPERATOR_DILATE | Corresponds to value **dilate**. |

### Attributes

**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feMorphology'** element.
**readonly SVGAnimatedEnumeration operator**
Corresponds to attribute **operator** on the given **'feMorphology'** element. Takes one of the Morphology Operators.
**readonly SVGAnimatedLength radiusX**
Corresponds to attribute **radius** on the given **'feMorphology'** element.
**readonly SVGAnimatedLength radiusY**
Corresponds to attribute **radius** on the given **'feMorphology'** element.

## Interface SVGFEOffsetElement

The **SVGFEOffsetElement** interface corresponds to the **'feOffset'** element.

**IDL Definition**

```
interface SVGFEOffsetElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};
```

**Attributes**
**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feOffset'** element.
**readonly SVGAnimatedNumber dx**
Corresponds to attribute **dx** on the given **'feOffset'** element.
**readonly SVGAnimatedNumber dy**
Corresponds to attribute **dy** on the given **'feOffset'** element.

## Interface SVGFESpecularLightingElement

The **SVGFESpecularLightingElement** interface corresponds to the **'feSpecularLighting'** element.

**IDL Definition**

```
interface SVGFESpecularLightingElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
};
```

**Attributes**
**readonly SVGAnimatedString in1**
Corresponds to attribute **in** on the given **'feSpecularLighting'** element.
**readonly SVGAnimatedNumber surfaceScale**

Corresponds to attribute **surfaceScale** on the given **'feSpecularLighting'** element.

**readonly SVGAnimatedNumber specularConstant**

Corresponds to attribute **specularConstant** on the given **'feSpecularLighting'** element.

**readonly SVGAnimatedNumber specularExponent**

Corresponds to attribute **specularExponent** on the given **'feSpecularLighting'** element.

## Interface SVGFETileElement

The **SVGFETileElement** interface corresponds to the **'feTile'** element.

### IDL Definition

```
interface SVGFETileElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
};
```

### Attributes

**readonly SVGAnimatedString in1**

Corresponds to attribute **in** on the given **'feTile'** element.

## Interface SVGFETurbulenceElement

The **SVGFETurbulenceElement** interface corresponds to the **'feTurbulence'** element.

### IDL Definition

```
interface SVGFETurbulenceElement :
               SVGElement,
               SVGFilterPrimitiveStandardAttributes {

  // Turbulence Types
  const unsigned short SVG_TURBULENCE_TYPE_UNKNOWN      = 0;
  const unsigned short SVG_TURBULENCE_TYPE_FRACTALNOISE = 1;
  const unsigned short SVG_TURBULENCE_TYPE_TURBULENCE   = 2;
  // Stitch Options
  const unsigned short SVG_STITCHTYPE_UNKNOWN  = 0;
  const unsigned short SVG_STITCHTYPE_STITCH   = 1;
  const unsigned short SVG_STITCHTYPE_NOSTITCH = 2;

  readonly attribute SVGAnimatedNumber      baseFrequencyX;
  readonly attribute SVGAnimatedNumber      baseFrequencyY;
  readonly attribute SVGAnimatedInteger     numOctaves;
  readonly attribute SVGAnimatedNumber      seed;
  readonly attribute SVGAnimatedEnumeration stitchTiles;
  readonly attribute SVGAnimatedEnumeration type;
};
```

### Definition group Turbulence Types

#### Defined constants

| | |
|---|---|
| SVG_TURBULENCE_TYPE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_TURBULENCE_TYPE_FRACTALNOISE | Corresponds to value **fractalNoise**. |

| | |
|---|---|
| SVG_TURBULENCE_TYPE_TURBULENCE | Corresponds to value **turbulence**. |

**Definition group Stitch Options**

**Defined constants**

| | |
|---|---|
| SVG_STITCHTYPE_UNKNOWN | The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type. |
| SVG_STITCHTYPE_STITCH | Corresponds to value **stitch**. |
| SVG_STITCHTYPE_NOSTITCH | Corresponds to value **noStitch**. |

**Attributes**

**readonly SVGAnimatedNumber baseFrequencyX**
Corresponds to attribute **baseFrequencyX** on the given **'feTurbulence'** element.

**readonly SVGAnimatedNumber baseFrequencyY**
Corresponds to attribute **baseFrequencyY** on the given **'feTurbulence'** element.

**readonly SVGAnimatedInteger numOctaves**
Corresponds to attribute **numOctaves** on the given **'feTurbulence'** element.

**readonly SVGAnimatedNumber seed**
Corresponds to attribute **seed** on the given **'feTurbulence'** element.

**readonly SVGAnimatedEnumeration stitchTiles**
Corresponds to attribute **stitchTiles** on the given **'feTurbulence'** element. Takes one of the Stitching Options.

**readonly SVGAnimatedEnumeration type**
Corresponds to attribute **type** on the given **'feTurbulence'** element. Takes one of the Turbulence Types.

---

04 September 2001

# 16 Interactivity

## Contents



## 16.1 Introduction

SVG content can be interactive (i.e., responsive to user-initiated events) by utilizing the following features in the SVG language:

- User-initiated actions such as button presses on the pointing device (e.g., a mouse) can cause animations or scripts to execute.
- The user can initiate hyperlinks to new Web pages (see Links out of SVG content: the **'a'** element) by actions such as mouse clicks when the pointing device is positioned over particular graphics elements.
- In many cases, depending on the value of the **zoomAndPan** attribute on the **'svg'** element and on the characteristics of the user agent, users are able to zoom into and pan around SVG content.
- User movements of the pointing device can cause changes to the cursor that shows the current position of the pointing device.

This chapter describes:

- information about events, including under which circumstances events are triggered
- how to indicate whether a given document can be zoomed and panned
- how to specify which cursors to use

Related information can be found in other chapters:

- hyperlinks are discussed in Links
- scripting and event attributes are discussed in Scripting
- SVG's relationship to DOM2 events is discussed in Relationship with DOM2 event model
- animation is discussed in Animation

## 16.2 Complete list of supported events

The following aspects of SVG are affected by events:

- Using SVG Document Object Model (DOM), a script can register DOM2 event listeners so that script can be invoked when a given event occurs.
- SVG includes event attributes on selected elements which define script that can be executed when a given event occurs in association with the given element.
- SVG's animation elements can be defined to begin or end based on events.

The following table lists all of the events which are recognized and supported in SVG. The *Event name* in the first column is the name to use within SVG's animation elements to define the events which can start or end animations. The *DOM2 name* in the third column is the name to use when defining DOM2 event listeners. The *Event attribute name* in the fifth column contains the corresponding name of the event attributes that can be attached to elements in the SVG language.

| Event name | Description | DOM2 name | DOM2 category | Event attribute name |
|---|---|---|---|---|
| focusin | Occurs when an element receives focus, such as when a **'text'** becomes selected. | DOMFocusIn | UIEvent | onfocusin |

| | | | | |
|---|---|---|---|---|
| focusout | Occurs when an element loses focus, such as when a **'text'** becomes unselected. | DOMFocusOut | UIEvent | onfocusout |
| activate | Occurs when an element is activated, for instance, thru a mouse click or a keypress. A numerical argument is provided to give an indication of the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter). | DOMActivate | UIEvent | onactivate |
| click | Occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: `mousedown`, `mouseup`, `click`. If multiple clicks occur at the same screen location, the sequence | (same) | MouseEvent | onclick |

| | | | | |
|---|---|---|---|---|
| | repeats with the `detail` attribute incrementing with each repetition. | | | |
| mousedown | Occurs when the pointing device button is pressed over an element. | (same) | MouseEvent | onmousedown |
| mouseup | Occurs when the pointing device button is released over an element. | (same) | MouseEvent | onmouseup |
| mouseover | Occurs when the pointing device is moved onto an element. | (same) | MouseEvent | onmouseover |
| mousemove | Occurs when the pointing device is moved while it is over an element. | (same) | MouseEvent | onmousemove |
| mouseout | Occurs when the pointing device is moved away from an element. | (same) | MouseEvent | onmouseout |
| DOMSubtreeModified | This is a general event for notification of all changes to the document. It can be used instead of the more specific events listed below. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |

| | | | | |
|---|---|---|---|---|
| DOMNodeInserted | Fired when a node has been added as a child of another node. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |
| DOMNodeRemoved | Fired when a node is being removed from another node. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |
| DOMNodeRemovedFromDocument | Fired when a node is being removed from a document, either through direct removal of the Node or removal of a subtree in which it is contained. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |

| | | | | |
|---|---|---|---|---|
| DOMNodeInsertedIntoDocument | Fired when a node is being inserted into a document, either through direct insertion of the Node or insertion of a subtree in which it is contained. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |
| DOMAttrModified | Fired after an attribute has been modified on a node. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |
| DOMCharacterDataModified | Fired after CharacterData within a node has been modified but the node itself has not been inserted or deleted. (The normative definition of this event is the description in the DOM2 specification.) | (same) | MutationEvent | none |

| | | | | |
|---|---|---|---|---|
| SVGLoad | The event is triggered at the point at which the user agent has fully parsed the element and its descendants and is ready to act appropriately upon that element, such as being ready to render the element to the target device. Referenced external resources that are required must be loaded, parsed and ready to render before the event is triggered. Optional external resources are not required to be ready for the event to be triggered. | (same) | none | onload |
| SVGUnload | Only applicable to outermost **'svg'** elements. The unload event occurs when the DOM implementation removes a document from a window or frame. | (same) | none | onunload |

| | | | | |
|---|---|---|---|---|
| SVGAbort | The abort event occurs when page loading is stopped before an element has been allowed to load completely. | (same) | none | onabort |
| SVGError | The error event occurs when an element does not load properly or when an error occurs during script execution. | (same) | none | onerror |
| SVGResize | The resize event occurs when a document view is resized. (Only applicable to outermost **'svg'** elements.) | (same) | none | onresize |
| SVGScroll | The scroll event occurs when a document view is shifted in X or Y or both, such as when the document view is scrolled or panned. (Only applicable to outermost **'svg'** elements.) | (same) | none | onscroll |

| | | | | |
|---|---|---|---|---|
| SVGZoom | Occurs when the document changes its zoom level based on user interaction. (Only applicable to outermost **'svg'** elements.) | none | none | onzoom |
| beginEvent | Occurs when an animation element begins. For details, see the description of Interface TimeEvent in the SMIL Animation specification. | none | none | onbegin |
| endEvent | Occurs when an animation element ends. For details, see the description of Interface TimeEvent in the SMIL Animation specification. | none | none | onend |
| repeatEvent | Occurs when an animation element repeats. It is raised each time the element repeats, after the first iteration. For details, see the description of Interface TimeEvent in the SMIL Animation specification. | none | none | onrepeat |

As in DOM2 Key events, the SVG specification does not provide a key event set. An event set designed for use with keyboard input devices will be included in a later version of the DOM and SVG specifications.

A **SVGLoad** event is dispatched only to the element to which the event applies; it is not dispatched to its ancestors. For example, if an **'image'** element and its parent **'g'** element both have event listeners for **SVGLoad** events, when the **'image'** element has been loaded, only its event listener will be invoked. (The **'g'** element's event listener will indeed get invoked, but the invocation will happen when the **'g'** itself has been loaded.)

Details on the parameters passed to event listeners for the event types from DOM2 can be found in the DOM2 specification. For other event types, the parameters passed to event listeners are described elsewhere in this specification.

## 16.3 User interface events

On user agents which support interactivity, it is common for authors to define SVG documents such that they are responsive to user interface events. Among the set of possible user events are pointer events, keyboard events, and document events.

In response to user interface (UI) events, the author might start an animation, perform a hyperlink to another Web page, highlight part of the document (e.g., change the color of the graphics elements which are under the pointer), initiate a "roll-over" (e.g., cause some previously hidden graphics elements to appear near the pointer) or launch a script which communicates with a remote database.

For all UI event-related features defined as part of the SVG language via event attributes or animation, the event model corresponds to the *event bubbling* model described in DOM2 [DOM2-EVBUBBLE]. The *event capture* model from DOM2 [DOM2-EVCAPTURE] can only be established from DOM method calls.

## 16.4 Pointer events

User interface events that occur because of user actions performed on a pointer device are called pointer events.

Many systems support pointer devices such as a mouse or trackball. On systems which use a mouse, pointer events consist of actions such as mouse movements and mouse clicks. On systems with a different pointer device, the pointing device often emulates the behavior of the mouse by providing a mechanism for equivalent user actions, such as a button to press which is equivalent to a mouse click.

For each pointer event, the SVG user agent determines the *target element* of a given pointer event. The target element is the topmost graphics element whose relevant graphical content is under the pointer at the time of the event. (See property **'pointer-events'** for a description of how

to determine whether an element's relevant graphical content is under the pointer, and thus in which circumstances that graphic element can be the target element for a pointer event.) When an element is not displayed (i.e., when the **'display'** property on that element or one of its ancestors has a value of **none**), that element cannot be the target of pointer events.

The event is either initially dispatched to the *target element*, to one of the target element's ancestors, or not dispatched, depending on the following:

- If there are no graphics elements whose relevant graphics content is under the pointer (i.e., there is no target element), the event is not dispatched.
- Otherwise, there is a target element. If there is an ancestor of the target element which has specified an event handler with event capturing [DOM2-EVCAPTURE] for the given event, then the event is dispatched to that ancestor element.
- Otherwise, if the target element has an appropriate event handler for the given event, the event is dispatched to the target element.
- Otherwise, each ancestor of the target element (starting with its immediate parent) is checked to see if it has an appropriate event handler. If an ancestor is found with an appropriate event handler, the event is dispatched to that ancestor element.
- Otherwise, the event is discarded.

When event bubbling [DOM2-EVBUBBLE] is active, bubbling occurs up to all direct ancestors of the target element. Descendant elements receive events before their ancestors. Thus, if a **'path'** element is a child of a **'g'** element and they both have event listeners for **click** events, then the event will be dispatched to the **'path'** element before the **'g'** element.

When event capturing [DOM2-EVCAPTURE] is active, ancestor elements receive events before their descendants.

After an event is initially dispatched to a particular element, unless an appropriate action has been taken to prevent further processing (e.g., by invoking the **preventCapture()** or **preventBubble()** DOM method call), the event will be passed to the appropriate event handlers (if any) for that element's ancestors (in the case of event bubbling) or that element's descendants (in the case of event capture) for further processing.

## 16.5 Processing order for user interface events

The processing order for user interface events is as follows:

- Event handlers assigned to the topmost graphics element under the pointer (and the various ancestors of that graphics element via potential event bubbling [DOM2-EVBUBBLE]) receive the event first. If none of the activation event handlers take an explicit action to prevent further processing of the given event (e.g., by invoking the **preventDefault()** DOM method), then the event is passed on for:
- Processing of any relevant dynamic pseudo-classes (i.e., :hover, :active and :focus) [CSS2-DYNPSEUDO], after which the event is passed on for:
- (For those user interface events which invoke hyperlinks, such as mouse clicks in some user agents) Link processing. If a hyperlink is invoked in response to a user interface

event, the hyperlink typically will disable further activation event processing (e.g., often, the link will define a hyperlink to another Web page). If link processing does not disable further processing of the given event, then the event is passed on for:

- (For those user interface events which can select text, such as mouse clicks and drags on **'text'** elements) Text selection processing. When a text selection operation occurs, typically it will disable further processing of the given event; otherwise, the event is passed on for:
- Document-wide event processing, such as user agent facilities to allow zooming and panning of an SVG document fragment.

## 16.6 The 'pointer-events' property

In different circumstances, authors may want to control under what circumstances particular graphic elements can become the target of pointer events. For example, the author might want a given element to receive pointer events only when the pointer is over the stroked perimeter of a given shape. In other cases, the author might want a given element to ignore pointer events under all circumstances so that graphical elements underneath the given element will become the target of pointer events.

For example, suppose a circle with a **'stroke'** of **red** (i.e., the outline is solid red) and a **'fill'** of **none** (i.e., the interior is not painted) is rendered directly on top of a rectangle with a **'fill'** of **blue**. The author might want the circle to be the target of pointer events only when the pointer is over the perimeter of the circle. When the pointer is over the interior of the circle, the author might want the underlying rectangle to be the target element of pointer events.

The **'pointer-events'** property specifies under what circumstances a given graphics element can be the target element for a pointer event. It affects the circumstances under which the following are processed:

- user interface events such as mouse clicks
- dynamic pseudo-classes (i.e., :hover, :active and :focus) [CSS2-DYNPSEUDO]
- hyperlinks (see Links out of SVG content: the **'a'** element)

**'pointer-events'**

| | |
|---|---|
| *Value:* | visiblePainted \| visibleFill \| visibleStroke \| visible \| painted \| fill \| stroke \| all \| none \| inherit |
| *Initial:* | visiblePainted |
| *Applies to:* | graphics elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual |
| *Animatable:* | yes |

**visiblePainted**

The given element can be the target element for pointer events when the **'visibility'**

property is set to **visible** and when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the **'fill'** property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the **'stroke'** property is set to a value other than 'none'.

**visibleFill**
: The given element can be the target element for pointer events when the **'visibility'** property is set to **visible** and when the pointer is over the interior (i.e., fill) of the element. The value of the **'fill'** property does not effect event processing.

**visibleStroke**
: The given element can be the target element for pointer events when the **'visibility'** property is set to **visible** and when the pointer is over the perimeter (i.e., stroke) of the element. The value of the **'stroke'** property does not effect event processing.

**visible**
: The given element can be the target element for pointer events when the **'visibility'** property is set to **visible** and the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element. The values of the **'fill'** and **'stroke'** do not effect event processing.

**painted**
: The given element can be the target element for pointer events when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the **'fill'** property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the **'stroke'** property is set to a value other than 'none'. The value of the **'visibility'** property does not effect event processing.

**fill**
: The given element can be the target element for pointer events when the pointer is over the interior (i.e., fill) of the element. The values of the **'fill'** and **'visibility'** properties do not effect event processing.

**stroke**
: The given element can be the target element for pointer events when the pointer is over the perimeter (i.e., stroke) of the element. The values of the **'stroke'** and **'visibility'** properties do not effect event processing.

**all**
: The given element can be the target element for pointer events whenever the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element. The values of the **'fill'**, **'stroke'** and **'visibility'** properties do not effect event processing.

**none**
: The given element does not receive pointer events.

For text elements, hit detection is performed on a character cell basis:

- The value **visiblePainted** means that the text string can receive events anywhere within the character cell if either the **'fill'** property is set to a value other than **none** or the **'stroke'** property is set to a value other than **none**, with the additional requirement that the **'visibility'** property is set to **visible**.
- The values **visibleFill**, **visibleStroke** and **visible** are equivalent and indicate that the text string can receive events anywhere within the character cell if the **'visibility'** property is set to **visible**. The values of the **'fill'** and **'stroke'** properties do not effect event processing.
- The value **painted** means that the text string can receive events anywhere within the

character cell if either the **'fill'** property is set to a value other than **none** or the **'stroke'** property is set to a value other than **none**. The value of the **'visibility'** property does not effect event processing.
- The values **fill**, **stroke** and **all** are equivalent and indicate that the text string can receive events anywhere within the character cell. The values of the **'fill'**, **'stroke'** and **'visibility'** properties do not effect event processing.
- The value **none** indicates that the given text does not receive pointer events.

For raster images, hit detection is either performed on a whole-image basis (i.e., the rectangular area for the image is one of the determinants for whether the image receives the event) or on a per-pixel basic (i.e., the alpha values for pixels under the pointer help determine whether the image receives the event):

- The value **visiblePainted** means that the raster image can receive events anywhere within the bounds of the image if any pixel from the raster image which is under the pointer is not fully transparent, with the additional requirement that the **'visibility'** property is set to **visible**.
- The values **visibleFill**, **visibleStroke** and **visible** are equivalent and indicate that the image can receive events anywhere within the rectangular area for the image if the **'visibility'** property is set to **visible**.
- The value **painted** means that the raster image can receive events anywhere within the bounds of the image if any pixel from the raster image which is under the pointer is not fully transparent. The value of the **'visibility'** property does not effect event processing.
- The values **fill**, **stroke** and **all** are equivalent and indicate that the image can receive events anywhere within the rectangular area for the image. The value of the **'visibility'** property does not effect event processing.
- The value **none** indicates that the image does not receive pointer events.

Note that for raster images, the values of properties **'opacity'**, **'fill-opacity'**, **'stroke-opacity'**, **'fill'** and **'stroke'** do not effect event processing.

## 16.7 Magnification and panning

Magnification represents a complete, uniform transformation on an SVG document fragment, where the magnify operation scales all graphical elements by the same amount. A magnify operation has the effect of a supplemental scale and translate transformation placed at the outermost level on the SVG document fragment (i.e., outside the outermost **'svg'** element).

Panning represents a translation (i.e., a shift) transformation on an SVG document fragment in response to a user interface action.

SVG user agents that operate in interaction-capable user environments are required to support the ability to magnify and pan.

The outermost **'svg'** element in an SVG document fragment has attribute **zoomAndPan**, which takes the possible values of *disable* and *magnify*, with the default being *magnify*.

If *disable*, the user agent shall disable any magnification and panning controls and not allow the user to magnify or pan on the given document fragment.

If *magnify*, in environments that support user interactivity, the user agent shall provide controls to allow the user to perform a "magnify" operation on the document fragment.

If a **zoomAndPan** attribute is assigned to an inner **'svg'** element, the **zoomAndPan** setting on the inner **'svg'** element will have no effect on the SVG user agent.

*Animatable: no.*

## 16.8 Cursors

### 16.8.1 Introduction to cursors

Some interactive display environments provide the ability to modify the appearance of the pointer, which is also known as the *cursor*. Three types of cursors are available:

- Standard built-in cursors
- Platform-specific custom cursors
- Platform-independent custom cursors

The **'cursor'** property is used to specify which cursor to use. The **'cursor'** property can be used to reference standard built-in cursors by specifying a keyword such as *crosshair* or a custom cursor. Custom cursors are referenced via a <uri> and can point to either an external resource such as a platform-specific cursor file or to a **'cursor'** element, which can be used to define a platform-independent cursor.

### 16.8.2 The 'cursor' property

**'cursor'**

| | |
|---|---|
| *Value:* | [ [<uri> ,]* [ auto \| crosshair \| default \| pointer \| move \| e-resize \| ne-resize \| nw-resize \| n-resize \| se-resize \| sw-resize \| s-resize \| w-resize\| text \| wait \| help ] ] \| inherit |
| *Initial:* | auto |
| *Applies to:* | container elements and graphics elements |
| *Inherited:* | yes |
| *Percentages:* | N/A |
| *Media:* | visual, interactive |
| *Animatable:* | yes |

This property specifies the type of cursor to be displayed for the pointing device. Values have the following meanings:

**auto**

The UA determines the cursor to display based on the current context.

**crosshair**
A simple crosshair (e.g., short line segments resembling a "+" sign).

**default**
The platform-dependent default cursor. Often rendered as an arrow.

**pointer**
The cursor is a pointer that indicates a link.

**move**
Indicates something is to be moved.

**e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize**
Indicate that some edge is to be moved. For example, the 'se-resize' cursor is used when the movement starts from the south-east corner of the box.

**text**
Indicates text that can be selected. Often rendered as an I-bar.

**wait**
Indicates that the program is busy. Often rendered as a watch or hourglass.

**help**
Help is available for the object under the cursor. Often rendered as a question mark or a balloon.

**<uri>**
The user agent retrieves the cursor from the resource designated by the URI. If the user agent cannot handle the first cursor of a list of cursors, it shall attempt to handle the second, etc. If the user agent cannot handle any user-defined cursor, it must use the generic cursor at the end of the list.

```
P { cursor : url("mything.cur"), url("second.csr"), text; }
```

The **'cursor'** property for SVG is identical to the **'cursor'** property defined in the "Cascading Style Sheets (CSS) level 2" specification [CSS2], with the exception that SVG user agents must support cursors defined by the **'cursor'** element.

## 16.8.3 The 'cursor' element

The **'cursor'** element can be used to define a platform-independent custom cursor. A recommended approach for defining a platform-independent custom cursor is to create a PNG [PNG01] image and define a **'cursor'** element that references the PNG image and identifies the exact position within the image which is the pointer position (i.e., the hot spot).

The PNG format is recommended because it supports the ability to define a transparency mask via an alpha channel. If a different image format is used, this format should support the definition of a transparency mask (two options: provide an explicit alpha channel or use a particular pixel color to indicate transparency). If the transparency mask can be determined, the mask defines the shape of the cursor; otherwise, the cursor is an opaque rectangle. Typically, the other pixel information (e.g., the R, G and B channels) defines the colors for those parts of the cursor which are not masked out. Note that cursors usually contain at least two colors so that the cursor can be visible over most backgrounds.

```
<!ELEMENT cursor (%descTitleMetadata;) >
<!ATTLIST cursor
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED >
```

*Attribute definitions:*

**x = "<coordinate>"**
: The *x-coordinate* of the position in the cursor's coordinate system which represents the precise position that is being pointed to.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>"**
: The *y-coordinate* of the position in the cursor's coordinate system which represents the precise position that is being pointed to.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`xlink:href` = "<uri>"**
: A URI reference to the file or element which provides the image of the cursor.
*Animatable: yes.*

***Attributes defined elsewhere:***
: **%stdAttrs;, %testAttrs;, %xlinkRefAttrs;, externalResourcesRequired.**

SVG user agents are required to support PNG format images as targets of the **xlink:href** property.

## 16.9 DOM interfaces

The following interfaces are defined below: SVGCursorElement.

### Interface SVGCursorElement

The **SVGCursorElement** interface corresponds to the **'cursor'** element.

### IDL Definition

```
interface SVGCursorElement :
                SVGElement,
                SVGURIReference,
                SVGTests,
                SVGExternalResourcesRequired {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
};
```

### Attributes

**readonly SVGAnimatedLength x**

Corresponds to attribute **x** on the given **'cursor'** element.

**readonly SVGAnimatedLength y**

Corresponds to attribute **y** on the given **'cursor'** element.

---

04 September 2001

# 17 Linking

## Contents



## 17.1 Links out of SVG content: the 'a' element

SVG provides an **'a'** element, analogous to HTML's **'a'** element, to indicate links (also known as *hyperlinks* or *Web links*). SVG uses XLink ([XLink]) for all link definitions.

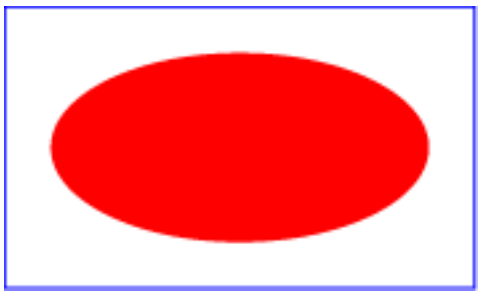
SVG 1.0 only requires that user agents support XLink's notion of simple links. Each simple link associates exactly two resources, one local and one remote, with an arc going from the former to the latter.

A simple link is defined for each separate rendered element contained within the **'a'** element; thus, if the **'a'** element contains three **'circle'** elements, a link is created for each circle. For each rendered element within an **'a'** element, the given rendered element is the local resource (the source anchor for the link).

The remote resource (the destination for the link) is defined by a URI specified by the XLink **href** attribute on the **'a'** element. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, an element within the current document, an element within a different document, etc.). By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may visit these resources.

Example link01 assigns a link to an ellipse.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="3cm" viewBox="0 0 5 3"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Example link01 - a link on an ellipse
  </desc>
  <rect x=".01" y=".01" width="4.98" height="2.98"
        fill="none" stroke="blue"  stroke-width=".03"/>
  <a xlink:href="http://www.w3.org">
    <ellipse cx="2.5" cy="1.5" rx="2" ry="1"
             fill="red" />
  </a>
</svg>
```



***Example link01***

*View this example as SVG (SVG-enabled browsers only)*

If the above SVG file is viewed by a user agent that supports both SVG and HTML, then clicking on the ellipse will cause the current window or frame to be replaced by the W3C home page.

```
<!ENTITY % aExt "" >
<!ELEMENT a       (#PCDATA|desc|title|metadata|defs|
                   path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|view|switch|a|altGlyphDef|
                   script|style|symbol|marker|clipPath|mask|
                   linearGradient|radialGradient|pattern|filter|cursor|font|
                   animate|set|animateMotion|animateColor|animateTransform|
                   color-profile|font-face
                   %ceExt;%aExt;)* >
<!ATTLIST a
  %stdAttrs;
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
  xlink:type (simple) #FIXED "simple"
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
```

```
  xlink:show (new|replace) 'replace'
  xlink:actuate (onRequest) #FIXED 'onRequest'
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  target %LinkTarget; #IMPLIED >
```

*Attribute definitions:*

**`xmlns` [:*prefix*] = "*resource-name*"**
: Standard XML attribute for identifying an XML namespace. This attribute makes the XLink [XLink] namespace available to the current element. Refer to the "Namespaces in XML" Recommendation [XML-NS].

  *Animatable: no.*

**`xlink:type` = 'simple'**
: (See generic description of **xlink:type** attribute.)

**`xlink:role` = '<uri>'**
: (See generic description of **xlink:role** attribute.)

**`xlink:arcrole` = '<uri>'**
: (See generic description of **xlink:arcrole** attribute.)

**`xlink:title` = '<string>'**
: (See generic description of **xlink:title** attribute.)

**`xlink:show` = 'new | replace'**
: Indicates whether, upon activation of the link, traversing to the ending resource should load it in a new window, frame, pane, or other relevant presentation context or load it in the same window, frame, pane, or other relevant presentation context in which the starting resource was loaded. Refer to the "XML Linking Language (XLink)" [XLink].

  *Animatable: no.*

**`xlink:actuate` = 'onRequest'**
: An application should traverse from the starting resource to the ending resource only on a post-loading event triggered for the purpose of traversal. Refer to the "XML Linking Language (XLink)" [XLink].

  *Animatable: no.*

**`xlink:href` = "<uri>"**
: The location of the referenced object, expressed as a URI reference. Refer to the "XML Linking Language (XLink)" [XLink].

  *Animatable: yes.*

**`target` = "*<frame-target>*"**
: This attribute has applicability when there are multiple possible targets for the ending resource, such as when the parent document is a multi-frame HTML or XHTML document. This attribute specifies the name of the target location (e.g., an HTML or

XHTML frame) into which a document is to be opened when the link is activated. For more information on targets, refer to the appropriate HTML or XHTML specifications.
*Animatable: yes.*

***Attributes defined elsewhere:***
**%stdAttrs; %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired style, %PresentationAttributes-All;.**

## 17.2 Linking into SVG content: URI fragments and SVG views

### 17.2.1 Introduction: URI fragments and SVG views

On the Internet, resources are identified using URIs (Uniform Resource Identifiers) [URI]. For example, an SVG file called MyDrawing.svg located at http://example.com might have the following URI:

```
http://example.com/MyDrawing.svg
```

A URI can also address a particular element within an XML document by including a **URI fragment identifier** as part of the URI. A URI which includes a URI fragment identifier consists of an optional **base URI**, followed by a "#" character, followed by the URI fragment identifier. For example, the following URI can be used to specify the element whose ID is "Lamppost" within file MyDrawing.svg:

```
http://example.com/MyDrawing.svg#Lamppost
```

Because SVG content often represents a picture or drawing of something, a common need is to link into a particular **view** of the document, where a view indicates the initial transformations so as to present a closeup of a particular section of the document.

### 17.2.2 SVG fragment identifiers

To link into a particular view of an SVG document, the URI fragment identifier needs to be a correctly formed **SVG fragment identifier**. An SVG fragment identifier defines the meaning of the "selector" or "fragment identifier" portion of URIs that locate resources of MIME media type "image/svg+xml".

An SVG fragment identifier can come in three forms:

- Shorthand *bare name* form of addressing (e.g., **MyDrawing.svg#MyView**). This form of addressing, which allows addressing an SVG element by its ID, is compatible with the fragment addressing mechanism for older versions of HTML and the shorthand bare name formulation in "XML Pointer Language (XPointer)" [XPTR]. (The bare name form

of addressing **#MyElement** is equivalent to the XPointer formulation **#xpointer(id('MyView'))**.)

- XPointer-compatible ID reference (e.g., **MyDrawing.svg#xpointer(id('MyView'))**). This form of addressing, which also allows addressing an SVG element by its ID, is compatible with "XML Pointer Language (XPointer)" [XPTR] syntax and the XPath syntax for referencing IDs.
- **SVG view specification** (e.g., **MyDrawing.svg#svgView(viewBox(0,200,1000,1000))**). This form of addressing specifies the desired view of the document (e.g., the region of the document to view, the initial zoom level) completely within the SVG fragment specification. The contents of the SVG view specification are the five parameter specifications, viewBox(...), preserveAspectRatio(...), transform(...), zoomAndPan(...) and viewTarget(...), whose parameters have the same meaning as the corresponding attributes on a **'view'** element, or, in the case of transform(...), the same meaning as the corresponding attribute has on a **'g'** element).

An SVG fragment identifier is defined as follows:

```
SVGFragmentIdentifier ::= BareName |
                          XPointerIDRef |
                          SVGViewSpec

BareName ::= XML_Name

SVGViewSpec ::= 'svgView(' SVGViewAttributes ')'

SVGViewAttributes ::= SVGViewAttribute |
                      SVGViewAttribute ';' SVGViewAttributes

SVGViewAttribute ::= viewBoxSpec |
                     preserveAspectRatioSpec |
                     transformSpec |
                     zoomAndPanSpec |
                     viewTargetSpec

viewBoxSpec ::= 'viewBox(' ViewBoxParams ')'

preserveAspectRatioSpec = 'preserveAspectRatio(' AspectParams ')'

transformSpec ::= 'transform(' TransformParams ')'

zoomAndPanSpec ::= 'zoomAndPan(' ZoomAndPanParams ')'

viewTargetSpec ::= 'viewTarget(' ViewTargetParams ')'
```

where:

- **XPointerIDRef** conforms to the rules for referencing IDs in XPointer (see [XPTR] and XPath syntax for referencing IDs). For example, `xpointer(id('MyView'))`.
- **ViewBoxParams** corresponds to the parameter values for the **viewBox** attribute on the **'view'** element. For example, **viewBox(0,0,200,200)**.
- **AspectParams** corresponds to the parameter values for the **preserveAspectRatio** attribute on the **'view'** element. For example, **preserveAspectRatio(xMidYMid)**.
- **TransformParams** corresponds to the parameter values for the **transform** attribute that is available on many elements. For example, **transform(scale(5))**.

- **ZoomAndPanParams** corresponds to the parameter values for the **zoomAndPan** attribute on the **'view'** element. For example, **zoomAndPan(magnify)**.
- **ViewTargetParams** corresponds to the parameter values for the **viewTarget** attribute on the **'view'** element. For example, **viewTarget(MyElementID)**.

Spaces are not allowed in fragment specifications; thus, commas are used to separate numeric values within an SVG view specification (e.g., **#svgView(viewBox(0,0,200,200))**) and semicolons are used to separate attributes (e.g., **#svgView(viewBox(0,0,200,200);preserveAspectRatio(none))**).

When a source document performs a link into an SVG document via an HTML [HTML4] anchor element (i.e., **<a href=...>** element in HTML) or an XLink specification [XLINK], then the SVG fragment identifier specifies the initial view into the SVG document, as follows:

- If no SVG fragment identifier is provided (e.g, the specified URI did not contain a "#" character, such as **MyDrawing.svg**), then the initial view into the SVG document is established using the view specification attributes (i.e., viewBox, etc.) on the outermost **'svg'** element.
- If the SVG fragment identifier addresses a **'view'** element within an SVG document (e.g., **MyDrawing.svg#MyView** or **MyDrawing.svg#xpointer(id('MyView'))**) then the closest ancestor **'svg'** element is displayed in the viewport. Any view specification attributes included on the given **'view'** element override the corresponding view specification attributes on the closest ancestor **'svg'** element.
- If the SVG fragment identifier addresses specific SVG view (e.g., **MyDrawing.svg#svgView(viewBox(0,200,1000,1000))**), then the document fragment defined by the closest ancestor **'svg'** element is displayed in the viewport using the SVG view specification provided by the SVG fragment identifier.
- If the SVG fragment identifier addresses any element other than a **'view'** element, then the document defined by the closest ancestor **'svg'** element is displayed in the viewport using the view specification attributes on that **'svg'** element.

### 17.2.3 Predefined views: the 'view' element

The **'view'** element is defined as follows:

```
<!ENTITY % viewExt "" >
<!ELEMENT view (%descTitleMetadata;%viewExt;) >
<!ATTLIST view
  %stdAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  zoomAndPan (disable | magnify) 'magnify'
  viewTarget CDATA #IMPLIED >
```

*Attribute definitions:*

**`viewTarget` = "*XML_Name [XML_NAME]**"**

Indicates the target object associated with the view. If provided, then the target element(s) will be highlighted.
*Animatable: no.*

***Attributes defined elsewhere:***

**%stdAttrs;, viewBox, preserveAspectRatio, zoomAndPan externalResourcesRequired.**

## 17.3 DOM interfaces

The following interfaces are defined below: SVGAElement, SVGViewElement.

### Interface SVGAElement

The **SVGAElement** interface corresponds to the **'a'** element.

**IDL Definition**

```
interface SVGAElement :
                SVGElement,
                SVGURIReference,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedString target;
};
```

**Attributes**

**readonly SVGAnimatedString target**

Corresponds to attribute **target** on the given **'a'** element.

### Interface SVGViewElement

The **SVGViewElement** interface corresponds to the **'view'** element.

### IDL Definition

```
interface SVGViewElement :
                SVGElement,
                SVGExternalResourcesRequired,
                SVGFitToViewBox,
                SVGZoomAndPan {

  readonly attribute SVGStringList viewTarget;
};
```

### Attributes

**readonly SVGStringList viewTarget**

Corresponds to attribute **viewTarget** on the given **'view'** element. A list of DOMString values which contain the names listed in the **viewTarget** attribute. Each of the DOMString values can be associated with the corresponding element using the getElementById() method call.

---

04 September 2001

# 18 Scripting

## Contents



.

## 18.1 Specifying the scripting language

### 18.1.1 Specifying the default scripting language

The **contentScriptType** attribute on the **'svg'** element specifies the default scripting language for the given document fragment.

**`contentScriptType` = "*content-type*"**
: Identifies the default scripting language for the given document. This attribute sets the scripting language used to process the value strings in event attributes. The value ***content-type*** specifies a media type, per [RFC2045]. The default value is "text/ecmascript".
*Animatable: no.*

### 18.1.2 Local declaration of a scripting language

It is also possible to specify the scripting language for each individual **'script'** element by specifying a **type** attribute on the **'script'** element.

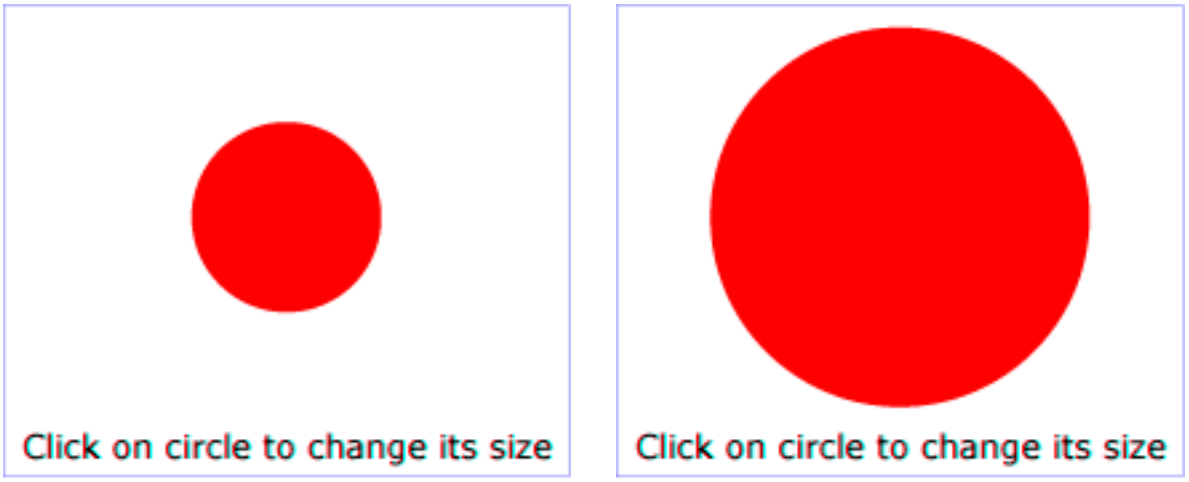
## 18.2 The 'script' element

A 'script' element is equivalent to the 'script' element in HTML and thus is the place for scripts (e.g., ECMAScript). Any functions defined within any 'script' element have a "global" scope across the entire current document.

Example script01 defines a function `circle_click` which is called by the **onclick** event attribute on the **'circle'** element. The drawing below on the left is the initial image. The drawing below on the right shows the result after clicking on the circle.

Note that this example demonstrates the use of the **onclick** event attribute for explanatory purposes. The example presupposes the presence of an input device with the same behavioral characteristics as a mouse, which will not always be the case. To support the widest range of users, the **onactivate** event attribute should be used instead of the **onclick** event attribute.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="6cm" height="5cm" viewBox="0 0 600 500"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example script01 - invoke an ECMAScript function from an onclick event
  </desc>
  <!-- ECMAScript to change the radius with each click -->
  <script type="text/ecmascript"> <![CDATA[
    function circle_click(evt) {
      var circle = evt.target;
      var currentRadius = circle.getAttribute("r");
      if (currentRadius == 100)
        circle.setAttribute("r", currentRadius*2);
      else
        circle.setAttribute("r", currentRadius*0.5);
    }
  ]]> </script>

  <!-- Outline the drawing area with a blue line -->
  <rect x="1" y="1" width="598" height="498" fill="none" stroke="blue"/>

  <!-- Act on each click event -->
  <circle onclick="circle_click(evt)" cx="300" cy="225" r="100"
          fill="red"/>

  <text x="300" y="480"
        font-family="Verdana" font-size="35" text-anchor="middle">
    Click on circle to change its size
  </text>
</svg>
```




***Example script01***

*View this example as SVG (SVG-enabled browsers only)*

```
<!ELEMENT script (#PCDATA) >
<!ATTLIST script
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  type %ContentType; #REQUIRED >
```

*Attribute definitions:*

**`type` = "*content-type*"**
Identifies the scripting language for the given **'script'** element. The value ***content-type*** specifies a media type, per [RFC2045]. *Animatable: no.*

***Attributes defined elsewhere:***
**%stdAttrs;, %xlinkRefAttrs;, href, externalResourcesRequired.**

## 18.3 Event handling

Events can cause scripts to execute when either of the following has occurred:

- Event attributes such as "onclick" or "onload" are assigned to particular elements,

where the value of the event attributes contains script which is executed when the given event occurs.
- Event listeners as described in "Document Object Model Events" [DOM2-EVENTS] are defined which are invoked when a given event happens on a given object

Related sections of the spec:

- User interface events describes how an SVG user agent handles events such as pointer movements events (e.g., mouse movement) and activation events (e.g., mouse click).
- Relationship with DOM2 events describes what parts of DOM are supported by SVG and how to register event listeners

## 18.4 Event attributes

The following event attributes are available on many SVG elements.

The complete list of events that are part of the SVG language and SVG DOM and descriptions of those events is provided in Complete list of supported events.

**Event attributes on graphics and container elements**

```
<!ENTITY % graphicsElementEvents
  "onfocusin %Script; #IMPLIED
   onfocusout %Script; #IMPLIED
   onactivate %Script; #IMPLIED
   onclick %Script; #IMPLIED
   onmousedown %Script; #IMPLIED
   onmouseup %Script; #IMPLIED
   onmouseover %Script; #IMPLIED
   onmousemove %Script; #IMPLIED
   onmouseout %Script; #IMPLIED
   onload %Script; #IMPLIED" >
```

**Document-level event attributes**

```
<!ENTITY % documentEvents
  "onunload %Script; #IMPLIED
   onabort %Script; #IMPLIED
   onerror %Script; #IMPLIED
   onresize %Script; #IMPLIED
   onscroll %Script; #IMPLIED
   onzoom %Script; #IMPLIED" >
```

**Animation event attributes**

```
<!ENTITY % animationEvents
  "onbegin %Script; #IMPLIED
   onend %Script; #IMPLIED
   onrepeat %Script; #IMPLIED" >
```

*Animatable: no.*

## 18.5 DOM interfaces

The following interfaces are defined below: SVGScriptElement, SVGEvent, SVGZoomEvent.

### Interface SVGScriptElement

The **SVGScriptElement** interface corresponds to the **'script'** element.

**IDL Definition**

```
interface SVGScriptElement :
                SVGElement,
                SVGURIReference,
                SVGExternalResourcesRequired {

           attribute DOMString type;
                       // raises DOMException on setting
};
```

**Attributes**

**DOMString type**

Corresponds to attribute **type** on the given **'script'** element.

**Exceptions on setting**

| | |
|---|---|
| DOMException | NO_MODIFICATION_ALLOWED_ERR: Raised on an attempt to change the value of a readonly attribute. |

### Interface SVGEvent

The SVG event set contains a list of special event types which are available in SVG.

A DOM consumer can use the hasFeature of the DOMImplementation interface to determine whether the SVG event set has been implemented by a DOM implementation. The feature string for this event set is "SVGEvents". This string is also used with the createEvent method.

The SVG events use the base DOM Event interface to pass contextual information.

The different types of such events that can occur are:

**SVGLoad**

See SVGLoad event.

- Bubbles: No
- Cancelable: No
- Context Info: None

**SVGUnload**

See SVGUnload event.

- Bubbles: No
- Cancelable: No
- Context Info: None

**SVGAbort**

See SVGAbort event.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

**SVGError**

See SVGError event.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

**SVGResize**

See SVGResize event.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

**SVGScroll**

See SVGScroll event.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

**IDL Definition**

```
interface SVGEvent : events::Event {};
```

## Interface SVGZoomEvent

A DOM consumer can use the hasFeature of the DOMImplementation interface to determine whether the SVG zoom event set has been implemented by a DOM implementation. The feature string for this event set is "SVGZoomEvents". This string is also used with the createEvent method.

The zoom event handler occurs before the zoom event is processed. The remainder of the DOM represents the previous state of the document. The document will be updated upon normal return from the event handler.

The UI event type for a zoom event is:

**SVGZoom**

The zoom event occurs when the user initiates an action which causes the current view of the SVG document fragment to be rescaled. Event handlers are only recognized on **'svg'** elements. See SVGZoom event.

- Bubbles: Yes
- Cancelable: No
- Context Info: zoomRectScreen, previousScale, previousTranslate, newScale, newTranslate, screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedNode.
  (screenX, screenY, clientX and clientY indicate the center of the zoom area, with clientX and clientY in viewport coordinates for the corresponding **'svg'** element. relatedNode is the corresponding **'svg'** element.)

**IDL Definition**

```
interface SVGZoomEvent : events::UIEvent {
  readonly attribute SVGRect zoomRectScreen;
  readonly attribute float previousScale;
  readonly attribute SVGPoint previousTranslate;
  readonly attribute float newScale;
  readonly attribute SVGPoint newTranslate;
};
```

**Attributes**

**readonly SVGRect zoomRectScreen**

The specified zoom rectangle in screen units.

The object itself and its contents are both readonly.

**readonly float previousScale**
The scale factor from previous zoom operations that was in place before the zoom operation occurred.

**readonly SVGPoint previousTranslate**

The translation values from previous zoom operations that were in place before the zoom operation occurred.

The object itself and its contents are both readonly.

**readonly float newScale**
The scale factor that will be in place after the zoom operation has been processed.

**readonly SVGPoint newTranslate**

The translation values that will be in place after the zoom operation has been processed.

The object itself and its contents are both readonly.

---

04 September 2001

# 19 Animation

## Contents



## 19.1 Introduction

Because the Web is a dynamic medium, SVG supports the ability to change vector graphics over time. SVG content can be animated in the following ways:

- Using SVG's animation elements. SVG document fragments can describe time-based modifications to the document's elements. Using the various animation elements, you can define motion paths, fade-in or fade-out effects, and objects that grow, shrink, spin or change color.
- Using the SVG DOM. The SVG DOM conforms to key aspects of the "Document Object Model (DOM) Level 1" [DOM1] and "Document Object Model (DOM) Level 2" [DOM2] specifications. Every attribute and style sheet setting is accessible to scripting, and SVG offers a set of additional DOM interfaces to support efficient animation via scripting. As a result, virtually any kind of animation can be achieved. The timer facilities in scripting languages such as ECMAScript can be used to start up

and control the animations. (See example below.)

- SVG has been designed to allow future versions of SMIL [SMIL1] to use animated or static SVG content as media components.
- In the future, it is expected that future versions of SMIL will be modularized and that components of it could be used in conjunction with SVG and other XML grammars to achieve animation effects.

## 19.2 Animation elements

### 19.2.1 Overview

SVG's animation elements were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [SMIL1].

The SYMM Working Group, in collaboration with the SVG Working Group, has authored the SMIL Animation specification [SMILANIM], which represents a general-purpose XML animation feature set. SVG incorporates the animation features defined in the SMIL Animation specification and provides some SVG-specific extensions.

For an introduction to the approach and features available in any language that supports SMIL Animation, see SMIL Animation overview and SMIL Animation animation model. For the list of animation features which go beyond SMIL Animation, see SVG extensions to SMIL Animation.

### 19.2.2 Relationship to SMIL Animation

SVG is a host language in terms of SMIL Animation and therefore introduces additional constraints and features as permitted by that specification. Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for SVG's animation elements and attributes is the SMIL Animation [SMILANIM] specification.

SVG supports the following four animation elements which are defined in the SMIL Animation specification:

| | |
|---|---|
| **'animate'** | allows scalar attributes and properties to be assigned different values over time |
| **'set'** | a convenient shorthand for **'animate'**, which is useful for assigning animation values to non-numeric attributes and properties, such as the **'visibility'** property |
| **'animateMotion'** | moves an element along a motion path |
| **'animateColor'** | modifies the color value of particular attributes or properties over time |

Additionally, SVG includes the following compatible extensions to SMIL Animation:

| | |
|---|---|
| **'animateTransform'** | modifies one of SVG's transformation attributes over time, such as the **transform** attribute |
| **path** attribute | SVG allows any feature from SVG's path data syntax to be specified in a **path** attribute to the **'animateMotion'** element (SMIL Animation only allows a subset of SVG's path data syntax within a **path** attribute) |
| **'mpath'** element | SVG allows an **'animateMotion'** element to contain a child **'mpath'** element which references an SVG **'path'** element as the definition of the motion path |

| | |
|---|---|
| **keyPoints** attribute | SVG adds a **keyPoints** attribute to the **'animateMotion'** to provide precise control of the velocity of motion path animations |
| **rotate** attribute | SVG adds a **rotate** attribute to the **'animateMotion'** to control whether an object is automatically rotated so that its x-axis points in the same direction (or opposite direction) as the directional tangent vector of the motion path |

For compatibility with other aspects of the language, SVG uses URI references via an **xlink:href** attribute to identify the elements which are to be targets of the animations.

SMIL Animation requires that the host language define the meaning for **document begin** and the **document end**. Since an **'svg'** is sometimes the root of the XML document tree and other times can be a component of a parent XML grammar, the *document begin* for a given SVG document fragment is defined to be the exact time at which the **'svg'** element's onload event is triggered. The *document end* of an SVG document fragment is the point at which the document fragment has been released and is no longer being processed by the user agent.

For SVG, the term **presentation time** indicates the position in the timeline relative to the *document begin* of a given document fragment.

SVG defines more constrained error processing than is defined in the SMIL Animation [SMILANIM] specification. SMIL Animation defines error processing behavior where the document continues to run in certain error situations, whereas all animations within an SVG document fragment will stop in the event of any error within the document (see Error processing).

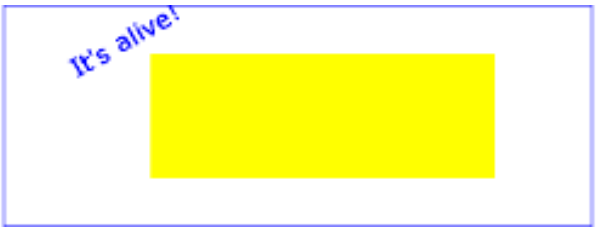
### 19.2.3 Animation elements example

Example anim01 below demonstrates each of SVG's five animation elements.

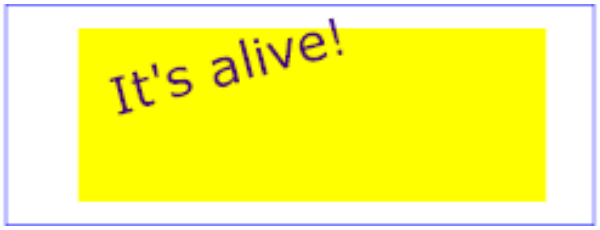
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="3cm"  viewBox="0 0 800 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example anim01 - demonstrate animation elements</desc>
  <rect x="1" y="1" width="798" height="298"
        fill="none" stroke="blue" stroke-width="2" />

  <!-- The following illustrates the use of the 'animate' element
        to animate a rectangles x, y, and width attributes so that
        the rectangle grows to ultimately fill the viewport. -->
  <rect id="RectElement" x="300" y="100" width="300" height="100"
        fill="rgb(255,255,0)"  >
    <animate attributeName="x" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="300" to="0" />
    <animate attributeName="y" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="100" to="0" />
    <animate attributeName="width" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="300" to="800" />
    <animate attributeName="height" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="100" to="300" />
  </rect>

  <!-- Set up a new user coordinate system so that
        the text string's origin is at (0,0), allowing
        rotation and scale relative to the new origin -->
  <g transform="translate(100,100)" >
```

```
    <!-- The following illustrates the use of the 'set', 'animateMotion',
         'animateColor' and 'animateTransform' elements. The 'text' element
         below starts off hidden (i.e., invisible). At 3 seconds, it:
           * becomes visible
           * continuously moves diagonally across the viewport
           * changes color from blue to dark red
           * rotates from -30 to zero degrees
           * scales by a factor of three. -->
    <text id="TextElement" x="0" y="0"
          font-family="Verdana" font-size="35.27" visibility="hidden"  >
      It's alive!
      <set attributeName="visibility" attributeType="CSS" to="visible"
           begin="3s" dur="6s" fill="freeze" />
      <animateMotion path="M 0 0 L 100 100"
           begin="3s" dur="6s" fill="freeze" />
      <animateColor attributeName="fill" attributeType="CSS"
           from="rgb(0,0,255)" to="rgb(128,0,0)"
           begin="3s" dur="6s" fill="freeze" />
      <animateTransform attributeName="transform" attributeType="XML"
           type="rotate" from="-30" to="0"
           begin="3s" dur="6s" fill="freeze" />
      <animateTransform attributeName="transform" attributeType="XML"
           type="scale" from="1" to="3" additive="sum"
           begin="3s" dur="6s" fill="freeze" />
    </text>
  </g>
</svg>
```

*At zero seconds*




*At three seconds*




*At six seconds*


*At nine seconds*

***Example anim01***

*View this example as SVG (SVG-enabled browsers only)*

The sections below describe the various animation attributes and elements.

## 19.2.4 Attributes to identify the target element for an animation

The following attributes are common to all animation elements and identify the target element for the animation.

```
<!ENTITY % animElementAttrs
 "%xlinkRefAttrs;
  xlink:href %URI; #IMPLIED" >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**

A URI reference to the element which is the target of this animation and which therefore will be modified over time.
The target element must be part of the current SVG document fragment.
<uri> must point to exactly one target element which is capable of being the target of the given animation. If <uri> points to multiple target elements, if the given target element is not capable of being a target of the given animation, or if the given target element is not part of the current SVG document fragment, then the document is in error (see Error processing).
If the **xlink:href** attribute is not provided, then the target element will be the immediate parent element of the current animation element.
Refer to the descriptions of the individual animation elements for any restrictions on what types of elements can be targets of particular types of animations.
Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: Specifying the animation target.

***Attributes defined elsewhere:***

**%xlinkRefAttrs;.**

## 19.2.5 Attributes to identify the target attribute or property for an animation

The following attributes identify the target attribute or property for the given target element whose value changes over time.

```
<!ENTITY % animAttributeAttrs
 "attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED" >
```

*Attribute definitions:*

**attributeName = <attributeName>**

Specifies the name of the target attribute. An XMLNS prefix may be used to indicate the XML namespace for the attribute. The prefix will be interpreted in the scope of the current (i.e., the referencing) animation element.
Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: Specifying the animation target.

**attributeType = "CSS | XML | auto"**

Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following (values are case-sensitive):

**"CSS"**
: This specifies that the value of "attributeName" is the name of a CSS property defined as animatable in this specification.

**"XML"**
: This specifies that the value of "attributeName" is the name of an XML attribute defined in the default XML namespace for the target element. If the value for `attributeName` has an XMLNS prefix, the implementation must use the associated namespace as defined in the scope of the target element. The attribute must be defined as animatable in this specification.

**"auto"**
: The implementation should match the `attributeName` to an attribute for the target element. The implementation must first search through the list of CSS properties for a matching property name, and if none is found, search the default XML namespace for the element.

The default value is "auto".

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: Specifying the animation target.

### 19.2.6 Attributes to control the timing of the animation

The following attributes are common to all animation elements and control the timing of the animation, including what causes the animation to start and end, whether the animation runs repeatedly, and whether to retain the end state the animation once the animation ends.

```
<!ENTITY % animTimingAttrs
 "begin CDATA #IMPLIED
  dur CDATA #IMPLIED
  end CDATA #IMPLIED
  min CDATA #IMPLIED
  max CDATA #IMPLIED
  restart (always | never | whenNotActive) 'always'
  repeatCount CDATA #IMPLIED
  repeatDur CDATA #IMPLIED
  fill (remove | freeze) 'remove'" >
```

In the syntax specifications that follow, optional white space is indicated as "S", defined as follows:

```
S ::= (#x20 | #x9 | #xD | #xA)*
```

*Attribute definitions:*

**begin : begin-value-list**
: Defines when the element should begin (i.e. become active).
The attribute value is a semicolon separated list of values.

**begin-value-list ::= begin-value (S? ";" S? begin-value-list )?**
: A semicolon separated list of begin values. The interpretation of a list of begin times is

detailed in SMIL Animation's section on "Evaluation of begin and end time lists".

**begin-value : ( offset-value | syncbase-value | event-value | repeat-value | accessKey-value | wallclock-sync-value | "indefinite" )**

Describes the element begin.

**offset-value ::= ( S? "+" | "-" S? )? ( Clock-value )**

For SMIL Animation, this describes the element begin as an offset from an implicit syncbase. For SVG, the implicit syncbase begin is defined to be relative to the document begin. SInce the document end in SVG is always undetermined, a negative offset value in SVG is always an error.

**syncbase-value ::= ( Id-value "." ( "begin" | "end" ) ) ( S? ("+"|"-") S? Clock-value )?**

Describes a **syncbase** and an optional offset from that syncbase. The element begin is defined relative to the begin or active end of another animation. A **syncbase** consists of an ID reference to another animation element followed by either `begin` or `end` to identify whether to synchronize with the beginning or active end of the referenced animation element.

**event-value ::= ( Id-value "." )? ( event-ref ) ( S? ("+"|"-") S? Clock-value )?**

Describes an event and an optional offset that determine the element begin. The animation begin is defined relative to the time that the event is raised. The list of event-symbols available for a given event-base element is the list of event attributes available for the given element as defined by the SVG DTD, with the one difference that the leading 'on' is removed from the event name (i.e., the animation event name is 'click', not 'onclick'). A list of all events supported by SVG can be found in Complete list of supported events. Details of event-based timing are described in SMIL Animation: Unifying Event-based and Scheduled Timing.

**repeat-value ::= ( Id-value "." )? "repeat(" integer ")" ( S? ("+"|"-") S? Clock-value )?**

Describes a qualified repeat event. The element begin is defined relative to the time that the repeat event is raised with the specified iteration value.

**accessKey-value ::= "accessKey(" character ")" ( S? ("+"|"-") S? Clock-value )?**

Describes an accessKey that determines the element begin. The element begin is defined relative to the time that the accessKey character is input by the user.

**"wallclock-sync-value : wallclock(" wallclock-value ")"**

Describes the element begin as a real-world clock time. The wallclock time syntax is based upon syntax defined in [ISO8601].

**"indefinite"**

The begin of the animation will be determined by a "beginElement()" method call or a hyperlink targeted to the element.
The animation DOM methods are described in DOM interfaces.
Hyperlink-based timing is described in SMIL Animation: Hyperlinks and timing.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'begin' attribute.

**dur : Clock-value | "media" | "indefinite"**

Specifies the simple duration.
The attribute value can be either of the following:

**Clock-value**

Specifies the length of the simple duration in presentation time. Value must be greater than 0.

**"media"**

Specifies the simple duration as the intrinsic media duration. This is only valid for elements that define media.
(For SVG's animation elements, if **"media"** is specified, the attribute will be ignored.)

**"indefinite"**

Specifies the simple duration as indefinite.

If the animation does not have a **dur** attribute, the simple duration is indefinite. Note that interpolation will not work if the simple duration is indefinite (although this may still be useful for **'set'** elements). Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'dur' attribute.

**end : end-value-list**

Defines an end value for the animation that can constrain the active duration. The attribute value is a semicolon separated list of values.

**end-value-list ::= end-value (S? ";" S? end-value-list )?**

A semicolon separated list of end values. The interpretation of a list of end times is detailed below.

**end-value : ( offset-value | syncbase-value | event-value | repeat-value | accessKey-value | wallclock-sync-value | "indefinite" )**

Describes the active end of the animation.

A value of **"indefinite"** specifies that the end of the animation will be determined by a "endElement()" method call (the animation DOM methods are described in DOM interfaces).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see description of SMIL Animation: 'end' attribute.

**min : Clock-value | "media"**

Specifies the minimum value of the active duration.

The attribute value can be either of the following:

**Clock-value**

Specifies the length of the minimum value of the active duration, measured in local time. Value must be greater than 0.

**"media"**

Specifies the minimum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's animation elements, if **"media"** is specified, the attribute will be ignored.)

The default value for **min** is "0". This does not constrain the active duration at all.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'min' attribute.

**max : Clock-value | "media"**

Specifies the maximum value of the active duration.

The attribute value can be either of the following:

**Clock-value**

Specifies the length of the maximum value of the active duration, measured in local time. Value must be greater than 0.

**"media"**

Specifies the maximum value of the active duration as the intrinsic media duration. This is only valid for elements that define media. (For SVG's animation elements, if **"media"** is specified, the attribute will be ignored.)

There is no default value for **max**. This does not constrain the active duration at all.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'max' attribute.

**restart : "always" | "whenNotActive" | "never"**

**always**

The animation can be restarted at any time.

This is the default value.

**whenNotActive**
The animation can only be restarted when it is not active (i.e. after the active end). Attempts to restart the animation during its active duration are ignored.

**never**
The element cannot be restarted for the remainder of the current simple duration of the parent time container. (In the case of SVG, since the parent time container is the SVG document fragment, then the animation cannot be restarted for the remainder of the document duration.)

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'restart' attribute.

**repeatCount : numeric value | "indefinite"**
Specifies the number of iterations of the animation function. It can have the following attribute values:

**numeric value**
This is a (base 10) "floating point" numeric value that specifies the number of iterations. It can include partial iterations expressed as fraction values. A fractional value describes a portion of the simple duration. Values must be greater than 0.

**"indefinite"**
The animation is defined to repeat indefinitely (i.e. until the document ends).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'repeatCount' attribute.

**repeatDur : Clock-value | "indefinite"**
Specifies the total duration for repeat. It can have the following attribute values:

**Clock-value**
Specifies the duration in presentation time to repeat the animation function **`f(t)`**.

**"indefinite"**
The animation is defined to repeat indefinitely (i.e. until the document ends).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'repeatDur' attribute.

**fill : "freeze" | "remove"**
This attribute can have the following values:

**freeze**
The animation effect F(t) is defined to freeze the effect value at the last value of the active duration. The animation effect is "frozen" for the remainder of the document duration (or until the animation is restarted - see SMIL Animation: Restarting animation ).

**remove**
The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end of the animation, the animation no longer affects the target (unless the animation is restarted - see SMIL Animation: Restarting animation ).
This is the default value.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'fill' attribute.

The SMIL Animation [SMILANIM] specification defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the SMIL Animation [SMILANIM] specification is the normative definition of the processing rules for the above attributes.

**Clock values**

Clock values have the same syntax as in SMIL Animation [SMILANIM], which is repeated here:

```
Clock-val         ::= Full-clock-val | Partial-clock-val
                      | Timecount-val
Full-clock-val    ::= Hours ":" Minutes ":" Seconds ("." Fraction)?
Partial-clock-val ::= Minutes ":" Seconds ("." Fraction)?
Timecount-val     ::= Timecount ("." Fraction)? (Metric)?
Metric            ::= "h" | "min" | "s" | "ms"
Hours             ::= DIGIT+; any positive number
Minutes           ::= 2DIGIT; range from 00 to 59
Seconds           ::= 2DIGIT; range from 00 to 59
Fraction      ::= DIGIT+
Timecount         ::= DIGIT+
2DIGIT        ::= DIGIT DIGIT
DIGIT         ::= [0-9]
```

For Timecount values, the default metric suffix is "s" (for seconds). No embedded white space is allowed in clock values, although leading and trailing white space characters will be ignored.

Clock values describe presentation time.

The following are examples of legal clock values:

- Full clock values:
  `02:30:03` = 2 hours, 30 minutes and 3 seconds
  `50:00:10.25` = 50 hours, 10 seconds and 250 milliseconds
- Partial clock value:
  `02:33` = 2 minutes and 33 seconds
  `00:10.5` = 10.5 seconds = 10 seconds and 500 milliseconds
- Timecount values:
  `3.2h` = 3.2 hours = 3 hours and 12 minutes
  `45min` = 45 minutes
  `30s` = 30 seconds
  `5ms` = 5 milliseconds
  `12.467` = 12 seconds and 467 milliseconds

Fractional values are just (base 10) floating point definitions of seconds. Thus:

```
00.5s = 500 milliseconds
00:00.005 = 5 milliseconds
```

### 19.2.7 Attributes that define animation values over time

The following attributes are common to elements **'animate'**, **'animateMotion'**, **'animateColor'** and **'animateTransform'**. These attributes define the values that are assigned to the target attribute or property over time. The attributes below provide control over the relative timing of keyframes and the interpolation method between discrete values.

```
<!ENTITY % animValueAttrs
 "calcMode (discrete | linear | paced | spline) 'linear'
  values CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED" >
```

*Attribute definitions:*

**`calcMode` = "discrete | linear | paced | spline"**

Specifies the interpolation mode for the animation. This can take any of the following values. The default mode is "linear", however if the attribute does not support linear interpolation (e.g. for strings), the `calcMode` attribute is ignored and discrete interpolation is used.

**`discrete`**

This specifies that the animation function will jump from one value to the next without any interpolation.

**`linear`**

Simple linear interpolation between values is used to calculate the animation function. Except for **'animateMotion'**, this is the default `calcMode`.

**`paced`**

Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). If "`paced`" is specified, any `keyTimes` or `keySplines` will be ignored. For **'animateMotion'**, this is the default `calcMode`.

**`spline`**

Interpolates from one value in the `values` list to the next according to a time function defined by a cubic Bézier spline. The points of the spline are defined in the `keyTimes` attribute, and the control points for each interval are defined in the `keySplines` attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'calcMode' attribute.

**values = "<list>"**

A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the `attributeType` domain. Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'values' attribute.

**`keyTimes` = "<list>"**

A semicolon-separated list of time values used to control the pacing of the animation. Each time in the list corresponds to a value in the `values` attribute list, and defines when the value is used in the animation function. Each time value in the `keyTimes` list is specified as a floating point value between 0 and 1 (inclusive), representing a proportional offset into the simple duration of the animation element.

If a list of `keyTimes` is specified, there must be exactly as many values in the `keyTimes` list as in the `values` list.

Each successive time value must be greater than or equal to the preceding time value.

The `keyTimes` list semantics depends upon the interpolation mode:

- For linear and spline animation, the first time value in the list must be 0, and the last time value in the list must be 1. The `keyTime` associated with each value defines when the value is set; values are interpolated between the `keyTimes`.
- For discrete animation, the first time value in the list must be 0. The time associated with each value defines when the value is set; the animation function uses that value until the next time defined in `keyTimes`.

If the interpolation mode is "paced", the `keyTimes` attribute is ignored.

If there are any errors in the `keyTimes` specification (bad values, too many or too few values), the document fragment is in error (see error processing).

If the simple duration is indefinite, any `keyTimes` specification will be ignored.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'keyTimes' attribute.

**`keySplines` = "<list>"**

A set of Bézier control points associated with the `keyTimes` list, defining a cubic Bézier function that controls interval pacing. The attribute value is a semicolon separated list of control point descriptions. Each control point description is a set of four values: `x1 y1 x2 y2`, describing the Bézier control points for one time segment. The `keyTimes` values that define the associated segment are the Bézier "anchor points", and the `keySplines` values are the control points. Thus, there must be one fewer sets of control points than there are `keyTimes`.

The values must all be in the range 0 to 1.

This attribute is ignored unless the `calcMode` is set to "spline".

If there are any errors in the `keySplines` specification (bad values, too many or too few values), the document fragment is in error (see error processing).

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'keySplines' attribute.

**from = "<value>"**

Specifies the starting value of the animation.
Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'from' attribute.

**to = "<value>"**

Specifies the ending value of the animation.
Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'to' attribute.

**by = "<value>"**

Specifies a relative offset value for the animation.
Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'by' attribute.

The SMIL Animation [SMILANIM] specification defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the SMIL Animation [SMILANIM] specification is the normative definition of the processing rules for the above attributes.

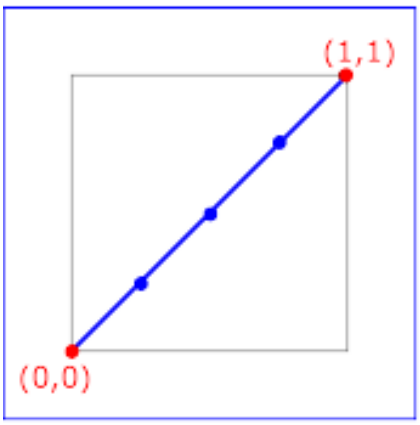
The animation values specified in the animation element must be legal values for the specified attribute. Leading and trailing white space, and white space before and after semicolon separators, will be ignored.

All values specified must be legal values for the specified attribute (as defined in the associated namespace). If any values are not legal, the document fragment is in error (see error processing).

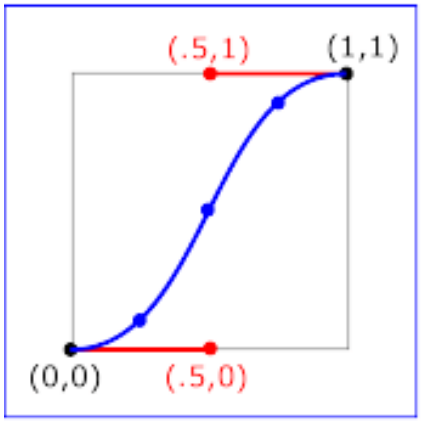
If a list of values is used, the animation will apply the values in order over the course of the animation. If a list of *values* is specified, any *from*, *to* and *by* attribute values are ignored.

The processing rules for the variants of *from/by/to* animations are described in Animation function values.

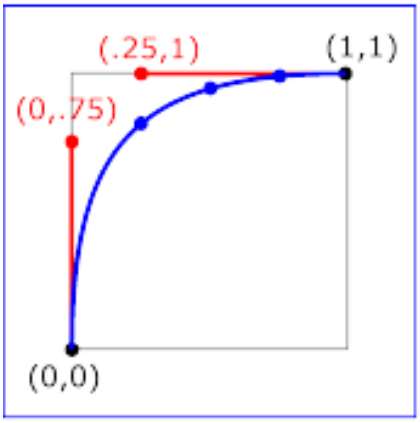
The following figure illustrates the interpretation of the `keySplines` attribute. Each diagram illustrates the effect of `keySplines` settings for a single interval (i.e. between the associated pairs of values in the `keyTimes` and `values` lists.). The horizontal axis can be thought of as the input value for the *unit progress* of interpolation within the interval - i.e. the pace with which interpolation proceeds along the given interval. The vertical axis is the resulting value for the *unit progress*, yielded by the `keySplines` function. Another way of describing this is that the horizontal axis is the input *unit time* for the interval, and the vertical axis is the output *unit time*. See also the section Timing and real-world clock times.



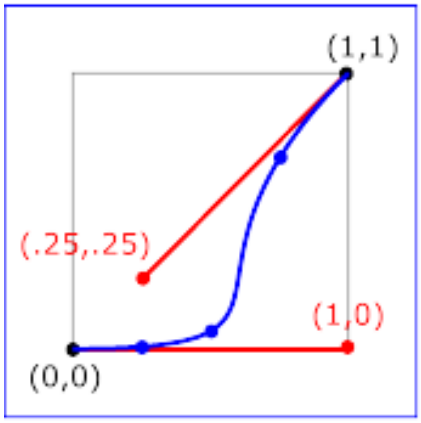

*keySplines="0 0 1 1" (the default)*




*keySplines=".5 0 .5 1"*




*keySplines="0 .75 .25 1"*




*keySplines="1 0 .25 .25"*

***Examples of keySplines***

To illustrate the calculations, consider the simple example:

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
     calcMode="spline" keySplines={as in table} />
```

Using the keySplines values for each of the four cases above, the approximate interpolated values as the animation proceeds are:

| keySplines values | Initial value | After 1s | After 2s | After 3s | Final value |
|---|---|---|---|---|---|
| 0 0 1 1 | 10.0 | 12.5 | 15.0 | 17.5 | 20.0 |
| .5 0 .5 1 | 10.0 | 11.0 | 15.0 | 19.0 | 20.0 |
| 0 .75 .25 1 | 10.0 | 18.0 | 19.3 | 19.8 | 20.0 |
| 1 0 .25 .25 | 10.0 | 10.1 | 10.6 | 16.9 | 20.0 |

For a formal definition of Bézier spline calculation, see [FOLEY-VANDAM].

### 19.2.8 Attributes that control whether animations are additive

It is frequently useful to define animation as an offset or delta to an attribute's value, rather than as absolute values. A simple "grow" animation can increase the width of an object by 10 pixels:

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
           additive="sum"/>
</rect>
```

It is frequently useful for repeated animations to build upon the previous results, accumulating with each interation. The following example causes the rectangle to continue to grow with each repeat of the animation:

```
<rect width="20px" ...>
  <animate attributeName="width" from="0px" to="10px" dur="10s"
           additive="sum" accumulate="sum" repeatCount="5"/>
</rect>
```

At the end of the first repetition, the rectangle has a width of 30 pixels. At the end of the second repetition, the rectangle has a width of 40 pixels. At the end of the fifth repetition, the rectangle has a width of 70 pixels.

For more information about additive animations, see SMIL Animation: Additive animation. For more information on cumulative animations, see SMIL Animation: Controlling behavior of repeating animation - Cumulative animation.

The following attributes are common to elements **'animate'**, **'animateMotion'**, **'animateColor'** and **'animateTransform'**.

```
<!ENTITY % animAdditionAttrs
 "additive       (replace | sum) 'replace'
  accumulate     (none | sum) 'none'" >
```

*Attribute definitions:*

**additive = "replace | sum"**

Controls whether or not the animation is additive.

**sum**

Specifies that the animation will add to the underlying value of the attribute and other lower priority animations.

**replace**

Specifies that the animation will override the underlying value of the attribute and other lower priority animations. This is the default, however the behavior is also affected by the animation value attributes **by** and **to**, as described in SMIL Animation: How from, to and by attributes affect additive behavior.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'additive' attribute.

**accumulate = "none | sum"**

Controls whether or not the animation is cumulative.

**sum**

Specifies that each repeat iteration after the first builds upon the last value of the previous iteration.

**none**

Specifies that repeat iterations are not cumulative. This is the default.

This attribute is ignored if the target attribute value does not support addition, or if the animation element does not repeat.

Cumulative animation is not defined for "*to animation*".

This attribute will be ignored if the animation function is specified with only the `to` attribute.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'accumulate' attribute.

## 19.2.9 Inheritance

SVG allows both attributes and properties to be animated. If a given attribute or property is inheritable by descendants, then animations on a parent element such as a **'g'** element has the effect of propagating the attribute or property animation values to descendant elements as the animation proceeds; thus, descendant elements can inherit animated attributes and properties from their ancestors.

## 19.2.10 The 'animate' element

The **'animate'** element is used to animate a single attribute or property over time. For example, to make a rectangle repeatedly fade away over 5 seconds, you can specify:

```
<rect>
  <animate attributeType="CSS" attributeName="opacity"
         from="1" to="0" dur="5s" repeatCount="indefinite" />
</rect>
```

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'animate' element.

```
<!ENTITY % animateExt "" >
<!ELEMENT animate (%descTitleMetadata;%animateExt;) >
<!ATTLIST animate
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >
```

***Attributes defined elsewhere:***

**%stdAttrs;, %testAttrs;, externalResourcesRequired, %animationEvents;, %animElementAttrs;, %animAttributeAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.**

For a list of attributes and properties that can be animated using the **'animate'** element, see Elements, attributes and properties that can be animated.

## 19.2.11 The 'set' element

The **'set'** element provides a simple means of just setting the value of an attribute for a specified duration. It supports all attribute types, including those that cannot reasonably be interpolated, such as string and boolean values. The **'set'** element is non-additive. The additive and accumulate attributes are not allowed, and will be ignored if specified.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'set' element.

```
<!ENTITY % setExt "" >
<!ELEMENT set (%descTitleMetadata;%setExt;) >
<!ATTLIST set
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  to CDATA #IMPLIED >
```

*Attribute definitions:*

**`to` = "<value>"**
: Specifies the value for the attribute during the duration of the **'set'** element. The argument value must match the attribute type.

***Attributes defined elsewhere:***
: **%stdAttrs;**, **%testAttrs;**, **externalResourcesRequired**, **%animationEvents;**, **%animElementAttrs;**, **%animAttributeAttrs;**, **%animTimingAttrs;**.

For a list of attributes and properties that can be animated using the **'set'** element, see Elements, attributes and properties that can be animated.

### 19.2.12 The 'animateMotion' element

The **'animateMotion'** element causes a referenced element to move along a motion path.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'animateMotion' element.

```
<!ENTITY % animateMotionExt "" >
<!ELEMENT animateMotion (%descTitleMetadata;,mpath? %animateMotionExt;) >
<!ATTLIST animateMotion
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animTimingAttrs;
  calcMode (discrete | linear | paced | spline) 'paced'
  values CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  %animAdditionAttrs;
  path CDATA #IMPLIED
  keyPoints CDATA #IMPLIED
  rotate CDATA #IMPLIED
  origin CDATA #IMPLIED >
```

*Attribute definitions:*

**`calcMode` = "discrete | linear | paced | spline"**
: Specifies the interpolation mode for the animation. Refer to general description of the **calcMode** attribute above. The only difference is that the default value for the **calcMode** for **'animateMotion'** is **paced**. See SMIL Animation: 'calcMode' attribute for 'animateMotion'.

**`path` = "<path-data>"**
: The motion path, expressed in the same format and interpreted the same way as the **d=** attribute on the **'path'** element. The effect of a motion path animation is to add a supplemental transformation matrix onto the CTM for the referenced object which causes a translation along the x- and y-axes of the current user coordinate system by the computed X and Y values computed over time.

**`keyPoints` = "<list-of-numbers>"**
: **keyPoints** takes a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the object shall move at the moment in time specified by corresponding **keyTimes** value. Distance calculations use the user agent's distance along the path algorithm. Each progress value in the list corresponds to a value in the `keyTimes` attribute list.
If a list of `keyPoints` is specified, there must be exactly as many values in the `keyPoints` list as in the `keyTimes` list.
If there are any errors in the `keyPoints` specification (bad values, too many or too few values), then the document is in error (see Error processing).

**`rotate` = "<angle> | auto | auto-reverse"**
: **auto** indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path. **auto-reverse** indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees. An actual angle value can also be given, which represents an angle relative to the x-axis of current user coordinate system. The **rotate** attribute adds a supplemental transformation matrix onto the CTM to apply a rotation transformation about the origin of the current user coordinate system. The rotation transformation is applied after the supplemental translation transformation that is computed due to the path attribute. The default value is 0.

**`origin` = "default"**
: The **origin** attribute is defined in the SMIL Animation specification [SMILANIM-ATTR-ORIGIN]. It has no effect in SVG.

***Attributes defined elsewhere:***

**%stdAttrs;**, **%testAttrs;**, **externalResourcesRequired**, **%animationEvents;**, **%animElementAttrs;**, **%animTimingAttrs;**, **values**, **keyTimes**, **keySplines**, **from**, **to**, **by**, **%animAdditionAttrs;**.

```
<!ENTITY % mpathExt "" >
<!ELEMENT mpath (%descTitleMetadata;%mpathExt;) >
<!ATTLIST mpath
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  externalResourcesRequired %Boolean; #IMPLIED >
```

*Attribute definitions:*

**`xlink:href` = "<uri>"**

A URI reference to the **'path'** element which defines the motion path.
*Animatable: no.*

***Attributes defined elsewhere:***

**%stdAttrs;**,, **%xlinkRefAttrs;** **externalResourcesRequired.**

For **'animateMotion'**, the specified values for **from**, **by**, **to** and **values** consists of x, y coordinate pairs, with a single comma and/or white space separating the x coordinate from the y coordinate. For example, **from="33,15"** specifies an x coordinate value of **33** and a y coordinate value of **15**.

If provided, the **values** attribute must consists of a list of x, y coordinate pairs. Coordinate values are separated by at least one white space character or a comma. Additional white space around the separator is allowed. For example, **values="10,20;30,20;30,40"** or **values="10mm,20mm;30mm,20mm;30mm,40mm"**. Each coordinate represents a length. Attributes **from**, **by**, **to** and **values** specify a shape on the current canvas which represents the motion path.

Two options are available which allow definition of a motion path using any of SVG's path data commands:

- the **path** attribute defines a motion path directly on **'animateMotion'** element using any of SVG's path data commands.
- the **'mpath'** sub-element provides the ability to reference an external **'path'** element as the definition of the motion path.

Note that SVG's path data commands can only contain values in user space, whereas **from**, **by**, **to** and **values** can specify coordinates in user space or using unit identifiers. See Units.

The various (x,y) points of the shape provide a supplemental transformation matrix onto the CTM for the referenced object which causes a translation along the x- and y-axes of the current user coordinate system

by the (x,y) values of the shape computed over time. Thus, the referenced object is translated over time by the offset of the motion path relative to the origin of the current user coordinate system. The supplemental transformation is applied on top of any transformations due to the target element's **transform** attribute or any animations on that attribute due to **'animateTransform'** elements on the target element.

The **additive** and **accumulate** attributes apply to **'animateMotion'** elements. Multiple **'animateMotion'** elements all simultaneously referencing the same target element can be additive with respect to each other; however, the transformations which result from the **'animateMotion'** elements are always supplemental to any transformations due to the target element's **transform** attribute or any **'animateTransform'** elements.

The default calculation mode (**calcMode**) for **animateMotion** is "paced". This will produce constant velocity motion along the specified path. Note that while animateMotion elements can be additive, it is important to observe that the addition of two or more "paced" (constant velocity) animations might not result in a combined motion animation with constant velocity.

When a **path** is combined with "discrete", "linear" or "spline" **calcMode** settings, and if attribute **keyPoints** is not provided, the number of values is defined to be the number of points defined by the path, unless there are "move to" commands within the path. A "move to" command within the **path** (i.e. other than at the beginning of the **path** description) A "move to" command does not count as an additional point when dividing up the duration, or when associating `keyTimes`, `keySplines` and `keyPoints` values. When a **path** is combined with a "paced" **calcMode** setting, all "move to" commands are considered to have 0 length (i.e. they always happen instantaneously), and is not considered in computing the pacing.

For more flexibility in controlling the velocity along the motion path, the **keyPoints** attribute provides the ability to specify the progress along the motion path for each of the **keyTimes** specified values. If specified, **keyPoints** causes **keyTimes** to apply to the values in **keyPoints** rather than the points specified in the **values** attribute array or the points on the **path** attribute.

The override rules for **'animateMotion** are as follows. Regarding the definition of the motion path, the **'mpath'** element overrides the the **path** attribute, which overrides **values**, which overrides **from/by/to**. Regarding determining the points which correspond to the **keyTimes** attributes, the **keyPoints** attribute overrides **path**, which overrides **values**, which overrides **from/by/to**.

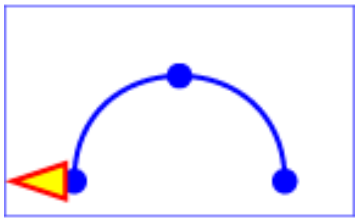
At any time *t* within a motion path animation of duration *dur*, the computed coordinate (x,y) along the motion path is determined by finding the point (x,y) which is *t/dur* distance along the motion path using the user agent's distance along the path algorithm.

The following example demonstrates the supplemental transformation matrices that are computed during a motion path animation.

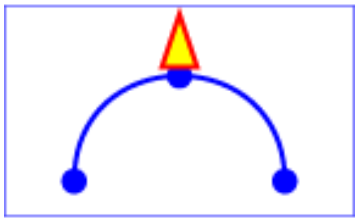
Example animMotion01 shows a triangle moving along a motion path.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="3cm"  viewBox="0 0 500 300"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example animMotion01 - demonstrate motion animation computations</desc>
  <rect x="1" y="1" width="498" height="298"
        fill="none" stroke="blue" stroke-width="2" />

  <!-- Draw the outline of the motion path in blue, along
         with three small circles at the start, middle and end. -->
  <path d="M100,250 C 100,50 400,50 400,250"
        fill="none" stroke="blue" stroke-width="7.06"  />
```

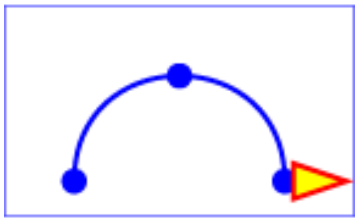
```
  <circle cx="100" cy="250" r="17.64" fill="blue"  />
  <circle cx="250" cy="100" r="17.64" fill="blue"  />
  <circle cx="400" cy="250" r="17.64" fill="blue"  />

  <!-- Here is a triangle which will be moved about the motion path.
       It is defined with an upright orientation with the base of
       the triangle centered horizontally just above the origin. -->
  <path d="M-25,-12.5 L25,-12.5 L 0,-87.5 z"
        fill="yellow" stroke="red" stroke-width="7.06"  >

    <!-- Define the motion path animation -->
    <animateMotion dur="6s" repeatCount="indefinite"
                   path="M100,250 C 100,50 400,50 400,250" rotate="auto" />
  </path>
</svg>
```



*At zero seconds*



*At three seconds*



*At six seconds*

***Example animMotion01***

*View this example as SVG (SVG-enabled browsers only)*

The following table shows the supplemental transformation matrices that are applied to achieve the effect of the motion path animation.

| | After 0s | After 3s | After 6s |
|---|---|---|---|
| Supplemental transform due to movement along motion path | translate(100,250) | translate(250,100) | translate(400,250) |
| Supplemental transform due to **rotate="auto"** | rotate(-90) | rotate(0) | rotate(90) |

For a list of elements that can be animated using the **'animateMotion'** element, see Elements, attributes and properties that can be animated.

## 19.2.13 The 'animateColor' element

The **'animateColor'** element specifies a color transformation over time.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this element is the SMIL Animation [SMILANIM] specification. In particular, see SMIL Animation: 'animateColor' element.

```
<!ENTITY % animateColorExt "" >
<!ELEMENT animateColor (%descTitleMetadata;%animateColorExt;) >
<!ATTLIST animateColor
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >
```

***Attributes defined elsewhere:***

**%stdAttrs;**, **%testAttrs;**, **externalResourcesRequired**, **%animationEvents;**, **%animElementAttrs;**, **%animAttributeAttrs;**, **%animTimingAttrs;**, **%animValueAttrs;**, **%animAdditionAttrs;**.

The **from**, **by** and **to** attributes take color values, where each color value is expressed using the following syntax (the same syntax as used in SVG's properties that can take color values):

```
<color> [icc-color(<name>[,<icccolorvalue>]*)]
```

The **values** attribute for the 'animateColor' element consists of a semicolon-separated list of color values, with each color value expressed in the above syntax.

Out of range color values can be provided, but user agent processing will be implementation dependent. User agents should clamp color values to allow color range values as late as possible, but note that system differences might preclude consistent behavior across different systems.

The **'color-interpolation'** property applies to color interpolations that result from **'animateColor'** animations.

For a list of attributes and properties that can be animated using the **'animateColor'** element, see Elements, attributes and properties that can be animated.

## 19.2.14 The 'animateTransform' element

The **'animateTransform'** element animates a transformation attribute on a target element, thereby allowing animations to control translation, scaling, rotation and/or skewing.

```
<!ENTITY % animateTransformExt "" >
<!ELEMENT animateTransform (%descTitleMetadata;%animateTransformExt;) >
<!ATTLIST animateTransform
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  type (translate | scale | rotate | skewX | skewY) "translate" >
```

*Attribute definitions:*

**type = "translate | scale | rotate | skewX | skewY"**
Indicates the type of transformation which is to have its values change over time.

***Attributes defined elsewhere:***
**%stdAttrs;, %testAttrs;, externalResourcesRequired, %animationEvents;, %animElementAttrs;, %animAttributeAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.**

The **from**, **by** and **to** attributes take a value expressed using the same syntax that is available for the given transformation type:

- For a **type="translate"**, each individual value is expressed as **<tx> [,<ty>]**.
- For a **type="scale"**, each individual value is expressed as **<sx> [,<sy>]**.
- For a **type="rotate"**, each individual value is expressed as **<rotate-angle> [<cx> <cy>]**.
- For a **type="skewX"** and **type="skewY"**, each individual value is expressed as **<skew-angle>**.

(See The **transform** attribute.)

The **values** attribute for the 'animateTransform' element consists of a semicolon-separated list of values, where each individual value is expressed as described above for **from**, **by** and **to**.

If **calcMode** has the value **paced**, then a total "distance" for each component of the transformation is calculated (e.g., for a translate operation, a total distance is calculated for both *tx* and *ty*) consisting of the sum of the absolute values of the differences between each pair of values, and the animation runs to produce a constant distance movement for each individual component.

When an animation is active, the effect of non-additive **'animateTransform'** (i.e., **additive="replace"**) is to replace the given attribute's value with the transformation defined by the **'animateTransform'**. The effect of additive (i.e., **additive="sum"**) is to post-multiply the transformation matrix corresponding to the transformation defined by this **'animateTransform'**. To illustrate:

```
<rect transform="skewX(30)"...>
  <animateTransform attributeName="transform" attributeType="XML"
```

```
                    type="rotate" from="0" to="90" dur="5s"
                    additive="replace" fill="freeze"/>
  <animateTransform attributeName="transform" attributeType="XML"
                    type="scale" from="1" to="2" dur="5s"
                    additive="replace" fill="freeze"/>
</rect>
```

In the code snippet above, because the both animations have **additive="replace"**, the first animation overrides the transformation on the rectangle itself and the second animation overrides the transformation from the first animation; therefore, at time 5 seconds, the visual result of the above two animations would be equivalent to the following static rectangle:

```
<rect transform="scale(2)" ... />
```

whereas in the following example:

```
<rect transform="skewX(30)"...>
  <animateTransform attributeName="transform" attributeType="XML"
                    type="rotate" from="0" to="90" dur="5s"
                    additive="sum" fill="freeze"/>
  <animateTransform attributeName="transform" attributeType="XML"
                    type="scale" from="1" to="2" dur="5s"
                    additive="sum" fill="freeze"/>
</rect>
```

In this code snippet, because the both animations have **additive="sum"**, the first animation post-multiplies its transformation to any transformations on the rectangle itself and the second animation post-multiplies its transformation to any transformation from the first animation; therefore, at time 5 seconds, the visual result of the above two animations would be equivalent to the following static rectangle:

```
<rect transform="skewX(30) rotate(90) scale(2)" ... />
```

For a list of attributes and properties that can be animated using the **'animateTransform'** element, see Elements, attributes and properties that can be animated.

## 19.2.15 Elements, attributes and properties that can be animated

The following lists all of the elements which can be animated by an **'animateMotion'** element:

- **'g'**
- **'defs'**
- **'use'**
- **'image'**
- **'switch'**
- **'path'**
- **'rect'**
- **'circle'**
- **'ellipse'**
- **'line'**
- **'polyline'**
- **'polygon'**
- **'text'**

- 'clipPath'
- 'mask'
- 'a'
- 'foreignObject'

Each attribute or property within this specification indicates whether or not it can be animated by SVG's animation elements. Animatable attributes and properties are designated as follows:

*Animatable: yes.*

whereas attributes and properties that cannot be animated are designated:

*Animatable: no.*

SVG has a defined set of basic data types for its various supported attributes and properties. For those attributes and properties that can be animated, the following table indicates which animation elements can be used to animate each of the basic data types. If a given attribute or property can take values of keywords (which are not additive) or numeric values (which are additive), then additive animations are possible if the subsequent animation uses a numeric value even if the base animation uses a keyword value; however, if the subsequent animation uses a keyword value, additive animation is not possible.

| Data type | Additive? | 'animate' | 'set' | 'animateColor' | 'animateTransform' | Notes |
|---|---|---|---|---|---|---|
| <angle> | yes | yes | yes | no | no | |
| <color> | yes | yes | yes | yes | no | Only RGB color values are additive. |
| <coordinate> | yes | yes | yes | no | no | |
| <frequency> | no | no | no | no | no | |
| <integer> | yes | yes | yes | no | no | |
| <length> | yes | yes | yes | no | no | |
| <list of xxx> | no | yes | yes | no | no | |
| <number> | yes | yes | yes | no | no | |
| <paint> | yes | yes | yes | yes | no | Only RGB color values are additive. |
| <percentage> | yes | yes | yes | no | no | |
| <time> | no | no | no | no | no | |
| <transform-list> | yes | no | no | no | yes | Additive means that a transformation is post-multiplied to the base set of transformations. |
| <uri> | no | yes | yes | no | no | |
| All other data types used in animatable attributes and properties | no | yes | yes | no | no | |

Any deviation from the above table or other special note about the animation capabilities of a particular

attribute or property is included in the section of the specification where the given attribute or property is defined.

## 19.3 Animation using the SVG DOM

Example dom01 shows a simple animation using the DOM.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="2cm" viewBox="0 0 400 200"
     xmlns="http://www.w3.org/2000/svg"
     onload="StartAnimation(evt)" >

  <script type="text/ecmascript"><![CDATA[
    var timevalue = 0;
    var timer_increment = 50;
    var max_time = 5000;
    var text_element;
    function StartAnimation(evt) {
      text_element = evt.target.ownerDocument.getElementById("TextElement");
      ShowAndGrowElement();
    }
    function ShowAndGrowElement() {
      timevalue = timevalue + timer_increment;
      if (timevalue > max_time)
        return;

      // Scale the text string gradually until it is 20 times larger
      scalefactor = (timevalue * 20.) / max_time;
      text_element.setAttribute("transform", "scale(" + scalefactor + ")");
      // Make the string more opaque
      opacityfactor = timevalue / max_time;
      text_element.setAttribute("opacity", opacityfactor);

      // Call ShowAndGrowElement again <timer_increment> milliseconds later.
      setTimeout("ShowAndGrowElement()", timer_increment)
    }
    window.ShowAndGrowElement = ShowAndGrowElement
  ]]></script>
  <rect x="1" y="1" width="398" height="198"
        fill="none" stroke="blue" stroke-width="2"/>

  <g transform="translate(50,150)" fill="red" font-size="7">
    <text id="TextElement">SVG</text>
  </g>
</svg>
```

*At zero seconds*


*At 2.5 seconds*


*At five seconds*

***Example dom01***

*View this example as SVG (SVG-enabled browsers only)*

The above SVG file contains a single graphics element, a text string that says "SVG". The animation loops

for 5 seconds. The text string starts out small and transparent and grows to be large and opaque. Here is an explanation of how this example works:

- The `onload="StartAnimation(evt)"` attribute indicates that, once the document has been fully loaded and processed, invoke ECMAScript function `StartAnimation`.
- The 'script' element defines the ECMAScript which makes the animation happen. The `StartAnimation()` function is only called once to give a value to global variable `text_element` and to make the initial call to `ShowAndGrowElement()`. `ShowAndGrowElement()` is called every 50 milliseconds and resets the `transform` and `style` attributes on the text element to new values each time it is called. At the end of `ShowAndGrowElement`, the function tells the ECMAScript engine to call itself again after 50 more milliseconds.
- The 'g' element shifts the coordinate system so that the origin is shifted toward the lower-left of the viewing area. It also defines the fill color and font-size to use when drawing the text string.
- The 'text' element contains the text string and is the element whose attributes get changed during the animation.

If scripts are modifying the same attributes or properties that are being animated by SVG's animation elements, the scripts modify the base value for the animation. If a base value is modified while an animation element is animating the corresponding attribute or property, the animations are required to adjust dynamically to the new base value.

If a script is modifying a property on the override style sheet at the same time that an animation element is animating that property, the result is implementation-dependent; thus, it is recommended that this be avoided.

## 19.4 DOM interfaces

The following two interfaces are from SMIL Animation. They are included here for easy reference:

### Interface ElementTimeControl

The **ElementTimeControl** interface, part of the **org.w3c.dom.smil** module and defined in SMIL Animation: Supported interfaces, defines common methods for elements which define animation behaviors compatible with SMIL Animation.

Calling `beginElement()` causes the animation to begin in the same way that an animation with event-based begin timing begins. The effective begin time is the current presentation time at the time of the DOM method call. Note that `beginElement()` is subject to the `restart` attribute in the same manner that event-based begin timing is. If an animation is specified to disallow restarting at a given point, `beginElement()` methods calls must fail. Refer also to the section Restarting animation.

Calling `beginElementAt(seconds)` has the same behavior as `beginElement()`, except that the effective begin time is offset from the current presentation time by an amount specified as a parameter. Passing a negative value for the offset causes the element to begin as for `beginElement()`, but has the effect that the element begins at the specified offset into its active duration. The `beginElementAt()` method must also respect the `restart` attribute. The restart semantics for a `beginElementAt()` method call are evaluated at the time of the method call, and not at the effective begin time specified by the offset parameter.

Calling `endElement()` causes an animation to end the active duration, just as `end` does. Depending upon the value of the `fill` attribute, the animation effect may no longer be applied, or it may be frozen at the current effect. Refer also to the section Freezing animations. If an animation is not currently active (i.e. if it has not yet begun or if it is frozen), the `endElement()` method will fail.

Calling `endElementAt()` causes an animation to end the active duration, just as `endElement()` does, but allows the caller to specify a positive offset, to cause the element to end at a point in the future. Other than delaying when the end actually happens, the semantics are identical to those for `endElement()`. If `endElementAt()` is called more than once while an element is active, the end time specified by the last method call will determine the end behavior.

**IDL Definition**

```
interface ElementTimeControl {

  boolean beginElement (  )
                  raises( DOMException );
  boolean beginElementAt ( in float offset )
                  raises( DOMException );
  boolean endElement (  )
                  raises( DOMException );
  boolean endElementAt ( in float offset )
                  raises( DOMException );
};
```

**Methods**

**beginElement**

Causes this element to begin the local timeline (subject to restart constraints).

**No Parameters**

**Return value**

boolean `true` if the method call was successful and the element was begun. `false` if the method call failed. Possible reasons for failure include:

- The element is already active and cannot be restarted when it is active. The `restart` attribute is set to `"whenNotActive"`.
- The element is active or has been active and cannot be restarted. The `restart` attribute is set to `"never"`.

**Exceptions**

DOMException SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `beginElement` calls.

**beginElementAt**

Causes this element to begin the local timeline (subject to restart constraints), at the passed offset from the current time when the method is called. If the offset is >= 0, the semantics are equivalent to an event-base begin with the specified offset. If the offset is < 0, the semantics are equivalent to beginElement(), but the element active duration is evaluated as though the element had begun at the passed (negative) offset from the current time when the method is called.

**Parameters**

in float offset The offset in seconds at which to begin the element.

**Return value**

boolean `true` if the method call was successful and the element was begun. `false` if the method call failed. Possible reasons for failure include:

- The element is already active and cannot be restarted when it is active. The `restart` attribute is set to `"whenNotActive"`.
- The element is active or has been active and cannot be restarted. The `restart` attribute is set to `"never"`.

**Exceptions**

DOMException SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `beginElementAt` calls.

**endElement**

Causes this element to end the local timeline.

**No Parameters**

**Return value**

boolean `true` if the method call was successful and the element was ended. `false` if method call failed. Possible reasons for failure include:

- The element is not active.

**Exceptions**

DOMException SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `endElement` calls.

**endElementAt**

Causes this element to end the local timeline at the specified offset from the current time when the method is called.

**Parameters**

in float offset The offset in seconds at which to end the element. Must be >= 0.

**Return value**

boolean `true` if the method call was successful and the element was ended. `false` if the method call failed. Possible reasons for failure include:

- The element is not active.

**Exceptions**

DOMException SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `endElementAt` calls.

The corresponding Java binding:

```
package org.w3c.dom.svg;

import org.w3c.dom.DOMException;

public interface ElementTimeControl {
  boolean beginElement (  )
                  throws DOMException;
  boolean beginElementAt ( float offset )
                  throws DOMException;
  boolean endElement (  )
                  throws DOMException;
  boolean endElementAt ( float offset )
                  throws DOMException;
}
```

## Interface TimeEvent

The **TimeEvent** interface, defined in SMIL Animation: Supported interfaces defined in SMIL Animation: Supported interfaces, provides specific contextual information associated with Time events.

The different types of events that can occur are:

**beginEvent**

This event is raised when the element local timeline begins to play. It will be raised each time the element begins the active duration (i.e. when it restarts, but not when it repeats). It may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was begun with the `beginElement()` or `beginElementAt()` methods. Note that if an element is restarted while it is currently playing, the element will raise an end event and another begin event, as the element restarts.

- Bubbles: No
- Cancelable: No
- Context Info: None

**endEvent**

This event is raised at the active end of the element. Note that this event is not raised at the simple end of each repeat. This event may be raised both in the course of normal (i.e. scheduled or interactive) timeline play, as well as in the case that the element was ended with the `endElement()` or `endElementAt()` methods. Note that if an element is restarted while it is currently playing, the element will raise an end event and another begin event, as the element restarts.

- Bubbles: No
- Cancelable: No
- Context Info: None

**repeatEvent**

This event is raised when an element local timeline repeats. It will be raised each time the element repeats, after the first iteration.
The event provides a numerical indication of which repeat iteration is beginning. The value is a 0-based integer, but the repeat event is not raised for the first iteration and so the observed values of the detail attribute will be >= 1.

- Bubbles: No
- Cancelable: No
- Context Info: detail (current iteration)

**IDL Definition**

```
interface TimeEvent : events::Event {
  readonly attribute views::AbstractView view;
  readonly attribute long                detail;

  void initTimeEvent ( in DOMString typeArg, in views::AbstractView viewArg, in long detailArg );
};
```

**Attributes**

**readonly views::AbstractView view**

The `view` attribute identifies the *AbstractView* [DOM2-VIEWS] from which the event was generated.

**readonly long detail**

Specifies some detail information about the Event, depending on the type of the event. For this event type, indicates the repeat number for the animation.

**Methods**

**initTimeEvent**

The initTimeEvent method is used to initialize the value of a *TimeEvent* created through the *DocumentEvent* interface. This method may only be called before the *TimeEvent* has been dispatched via the *dispatchEvent* method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

**Parameters**

| | |
|---|---|
| in DOMString typeArg | Specifies the event type. |
| in views::AbstractView viewArg | Specifies the Event's *AbstractView*. |
| in long detailArg | Specifies the *Event*'s detail. |

**No Return Value**

**No Exceptions**

The corresponding Java binding:

```
package org.w3c.dom.svg;

import org.w3c.dom.events.Event;
import org.w3c.dom.views.AbstractView;

public interface TimeEvent extends
               Event {
  public AbstractView getView( );
  public int          getDetail( );

  void initTimeEvent ( String typeArg, AbstractView viewArg, int detailArg );
}
```

The following interfaces are defined below: SVGAnimationElement, SVGAnimateElement, SVGSetElement, SVGAnimateMotionElement, SVGMPathElement, SVGAnimateColorElement, SVGAnimateTransformElement.

## Interface SVGAnimationElement

The **SVGAnimationElement** interface is the base interface for all of the animation element interfaces: SVGAnimateElement, SVGSetElement, SVGAnimateColorElement, SVGAnimateMotionElement and SVGAnimateTransformElement.

Unlike other SVG DOM interfaces, the SVG DOM does not specify convenience DOM properties corresponding to the various language attributes on SVG's animation elements. Specification of these convenience properties in a way that will be compatible with future versions of SMIL Animation is expected in a future version of SVG. The current method for accessing and modifying the attributes on the animation elements is to use the standard `getAttribute`, `setAttribute`, `getAttributeNS` and `setAttributeNS` defined in DOM2.

**IDL Definition**

```
interface SVGAnimationElement :
               SVGElement,
               SVGTests,
               SVGExternalResourcesRequired,
               smil::ElementTimeControl,
               events::EventTarget {
```

```
    readonly attribute SVGElement targetElement;

    float getStartTime (  );
    float getCurrentTime (  );
    float getSimpleDuration (  )
                    raises( DOMException );
  };
```

**Attributes**

**readonly SVGElement targetElement**

The element which is being animated.

**Methods**

**getStartTime**

Returns the start time in seconds for this animation.

**No Parameters**

**Return value**

float The start time in seconds for this animation relative to the start time of the time container.

**No Exceptions**

**getCurrentTime**

Returns the current time in seconds relative to time zero for the given time container.

**No Parameters**

**Return value**

float The current time in seconds relative to time zero for the given time container.

**No Exceptions**

**getSimpleDuration**

Returns the number of seconds for the simple duration for this animation. If the simple duration is undefined (e.g., the end time is indefinite), then an exception is raised.

**No Parameters**

**Return value**

float The number of seconds for the simple duration for this animation.

**Exceptions**

DOMException NOT_SUPPORTED_ERR: The simple duration is not determined on the given element.

## Interface SVGAnimateElement

The **SVGAnimateElement** interface corresponds to the **'animate'** element.

Object-oriented access to the attributes of the **'animate'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGAnimateElement : SVGAnimationElement {};
```

## Interface SVGSetElement

The **SVGSetElement** interface corresponds to the **'set'** element.

Object-oriented access to the attributes of the **'set'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGSetElement : SVGAnimationElement {};
```

## Interface SVGAnimateMotionElement

The **SVGAnimateMotionElement** interface corresponds to the **'animateMotion'** element.

Object-oriented access to the attributes of the **'animateMotion'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGAnimateMotionElement : SVGAnimationElement {};
```

## Interface SVGMPathElement

The **SVGMPathElement** interface corresponds to the **'mpath'** element.

**IDL Definition**

```
interface SVGMPathElement :
               SVGElement,
               SVGURIReference,
               SVGExternalResourcesRequired {};
```

## Interface SVGAnimateColorElement

The **SVGAnimateColorElement** interface corresponds to the **'animateColor'** element.

Object-oriented access to the attributes of the **'animateColor'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGAnimateColorElement : SVGAnimationElement {};
```

## Interface SVGAnimateTransformElement

The **SVGAnimateTransformElement** interface corresponds to the **'animateTransform'** element.

Object-oriented access to the attributes of the **'animateTransform'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGAnimateTransformElement : SVGAnimationElement {};
```

---

04 September 2001

# 20 Fonts

## Contents



## 20.1 Introduction

Reliable delivery of fonts is a requirement for SVG. Designers need to create SVG content with arbitrary fonts and know that the same graphical result will appear when the content is viewed by all end users, even when end users do not have the necessary fonts installed on their computers. This parallels the print world, where the designer uses a given font when authoring a drawing for print, and the graphical content appears exactly the same in the printed version as it appeared on the designer's authoring system.

SVG utilizes the WebFonts facility defined in the "Cascading Style Sheets (CSS) level 2" specification [CSS2] as a key mechanism for reliable delivery of font data to end users. In a common scenario, SVG authoring applications generate compressed, subsetted WebFonts for all text elements used by a given SVG document fragment. Typically, the WebFonts are saved in a location relative to the referencing document.

One disadvantage to the WebFont facility to date is that specifications such as [CSS2] do not require support of particular font formats. The result is that different implementations support different Web font formats, thereby making it difficult for Web site creators to post a single Web site using WebFonts

that work across all user agents.

To provide a common font format for SVG that is guaranteed to be supported by all conforming SVG viewers, SVG provides a facility to define fonts in SVG. This facility is called **SVG fonts**.

SVG fonts can improve the semantic richness of graphics that represent text. For example, many company logos consist of the company name drawn artistically. In some cases, accessibility may be enhanced by expressing the logo as a series of glyphs in an SVG font and then rendering the logo as a **'text'** element which references this font.

## 20.2 Overview of SVG fonts

An **SVG font** is a font defined using SVG's **'font'** element.

The purpose of SVG fonts is to allow for delivery of glyph outlines in display-only environments. SVG fonts that accompany Web pages must be supported only in browsing and viewing situations. Graphics editing applications or file translation tools must not attempt to convert SVG fonts into system fonts. The intent is that SVG files be interchangeable between two content creators, but not the SVG fonts that might accompany these SVG files. Instead, each content creator will need to license the given font before being able to successfully edit the SVG file. The **font-face-name** element indicates the name of licensed font to use for editing.

SVG fonts contain unhinted font outlines. Because of this, on many implementations there will be limitations regarding the quality and legibility of text in small font sizes. For increased quality and legibility in small font sizes, content creators may want to use an alternate font technology, such as fonts that ship with operating systems or an alternate WebFont format.

Because SVG fonts are expressed using SVG elements and attributes, in some cases the SVG font will take up more space than if the font were expressed in a different WebFont format which was especially designed for compact expression of font data. For the fastest delivery of Web pages, content creators may want to use an alternate font technology.

A key value of SVG fonts is guaranteed availability in SVG user agents. In some situations, it might be appropriate for an SVG font to be the first choice for rendering some text. In other situations, the SVG font might be an alternate, back-up font in case the first choice font (perhaps a hinted system font) is not available to a given user.

The characteristics and attributes of SVG fonts correspond closely to the font characteristics and parameters described in the "Fonts" chapter of the "Cascading Style Sheets (CSS) level 2" specification [CSS2]. In this model, various font metrics, such as advance values and baseline locations, and the glyph outlines themselves, are expressed in units that are relative to an abstract square whose height is the intended distance between lines of type in the same type size. This square is called the **em square** and it is the design grid on which the glyph outlines are defined. The value of the **units-per-em** attribute on the **'font'** element specifies how many units the em square is divided into. Common values for other font types are, for example, 250 (Intellifont), 1000 (Type 1) and 2048 (TrueType, TrueType GX and Open-Type). Unlike standard graphics in SVG, where the initial coordinate system has the y-axis pointing downward (see The initial coordinate system), the design grid for SVG fonts, along with the initial coordinate system for the glyphs, has the y-axis pointing

upward for consistency with accepted industry practice for many popular font formats.

SVG fonts and their associated glyphs do not specify bounding box information. Because the glyph outlines are expressed as SVG graphics elements, the implementation has the option to render the glyphs either using standard graphics calls or by using special-purpose font rendering technology, in which case any necessary maximum bounding box and overhang calculations can be performed from analysis of the graphics elements contained within the glyph outlines.

An SVG font can be either embedded within the same document that uses the font or saved as part of an external resource.

Here is an example of how you might embed an SVG font inside of an SVG document.

```
<?xml version="1.0" standalone="yes"?>
<svg width="400px" height="300px"
  xmlns = 'http://www.w3.org/2000/svg'>
  <defs>
    <font id="Font1" horiz-adv-x="1000">
      <font-face font-family="Super Sans" font-weight="bold" font-style="normal"
          units-per-em="1000" cap-height="600" x-height="400"
          ascent="700" descent="300"
          alphabetic="0" mathematical="350" ideographic="400" hanging="500">
        <font-face-src>
          <font-face-name name="Super Sans Bold"/>
        </font-face-src>
      </font-face>
      <missing-glyph><path d="M0,0h200v200h-200z"/></missing-glyph>
      <glyph unicode="!" horiz-adv-x="300"><!-- Outline of exclam. pt. glyph --></glyph>
      <glyph unicode="@"><!-- Outline of @ glyph --></glyph>
      <!-- more glyphs -->
    </font>
  </defs>
  <text x="100" y="100"
           style="font-family: 'Super Sans', Helvetica, sans-serif;
                  font-weight: bold; font-style: normal">Text
    using embedded font</text>
</svg>
```

Here is an example of how you might use the CSS @font-face facility to reference an SVG font which is saved in an external file. First referenced SVG font file:

```
<?xml version="1.0" standalone="yes"?>
<svg width="100%" height="100%"
 xmlns = 'http://www.w3.org/2000/svg'>
  <defs>
    <font id="Font2" horiz-adv-x="1000">
      <font-face font-family="Super Sans" font-weight="normal" font-style="italic"
          units-per-em="1000" cap-height="600" x-height="400"
          ascent="700" descent="300"
          alphabetic="0" mathematical="350" ideographic="400" hanging="500">
        <font-face-src>
          <font-face-name name="Super Sans Italic"/>
        </font-face-src>
      </font-face>
      <missing-glyph><path d="M0,0h200v200h-200z"/></missing-glyph>
      <glyph unicode="!" horiz-adv-x="300"><!-- Outline of exclam. pt. glyph --></glyph>
      <glyph unicode="@"><!-- Outline of @ glyph --></glyph>
      <!-- more glyphs -->
    </font>
  </defs>
```

```
</svg>
```

The SVG file which uses/references the above SVG font

```
<?xml version="1.0" standalone="yes"?>
<svg width="400px" height="300px"
 xmlns = 'http://www.w3.org/2000/svg'>  <defs>
    <style type="text/css">
      <![CDATA[
        @font-face {
          font-family: 'Super Sans';
          font-weight: normal;
          font-style: italic;
          src: url("myfont.svg#Font2") format(svg)
        }
      ]]>
   </style>
  </defs>
  <text x="100" y="100"
           style="font-family: 'Super Sans'; font-weight:normal;
                  font-style: italic">Text using embedded font</text>
</svg>
```

## 20.3 The 'font' element

The **'font'** element defines an SVG font.

```
<!ENTITY % fontExt "" >
<!ELEMENT font (%descTitleMetadata;,font-face,
                  missing-glyph,(glyph|hkern|vkern %fontExt;)*) >
<!ATTLIST font
  %stdAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  horiz-origin-x %Number; #IMPLIED
  horiz-origin-y %Number; #IMPLIED
  horiz-adv-x %Number; #REQUIRED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >
```

*Attribute definitions:*

**`horiz-origin-x` = "<number>"**
: The X-coordinate in the font coordinate system of the origin of a glyph to be used when drawing horizontally oriented text. (Note that the origin applies to all glyphs in the font.) If the attribute is not specified, the effect is as if a value of "0" were specified. *Animatable: no.*

**`horiz-origin-y` = "<number>"**
: The Y-coordinate in the font coordinate system of the origin of a glyph to be used when drawing horizontally oriented text. (Note that the origin applies to all glyphs in the font.) If the attribute is not specified, the effect is as if a value of "0" were specified.

*Animatable: no.*

**`horiz-adv-x` = "<number>"**
The default horizontal advance after rendering a glyph in horizontal orientation. Glyph widths are required to be non-negative, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts.
*Animatable: no.*

**`vert-origin-x` = "<number>"**
The default X-coordinate in the font coordinate system of the origin of a glyph to be used when drawing vertically oriented text.
If the attribute is not specified, the effect is as if the attribute were set to half of the effective value of attribute **horiz-adv-x**.
*Animatable: no.*

**`vert-origin-y` = "<number>"**
The default Y-coordinate in the font coordinate system of the origin of a glyph to be used when drawing vertically oriented text.
If the attribute is not specified, the effect is as if the attribute were set to the position specified by the font's **ascent** attribute.
*Animatable: no.*

**`vert-adv-y` = "<number>"**
The default vertical advance after rendering a glyph in vertical orientation.
If the attribute is not specified, the effect is as if a value equivalent of one *em* were specified (see **units-per-em**).
*Animatable: no.*

***Attributes defined elsewhere:***
**%stdAttrs;, externalResourcesRequired, class, style, %PresentationAttributes-All;.**

Each **'font'** element must have a **'font-face'** child element which describes various characteristics of the font.

## 20.4 The 'glyph' element

The **'glyph'** element defines the graphics for a given glyph. The coordinate system for the glyph is defined by the various attributes in the **'font'** element.

The graphics that make up the **'glyph'** can be either a single path data specification within the **d** attribute or arbitrary SVG as content within the **'glyph'**. These two alternatives are processed differently (see below).

```
<!ENTITY % glyphExt "" >
<!ELEMENT glyph (desc|title|metadata|defs|
                 path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|view|switch|a|altGlyphDef|
                 script|style|symbol|marker|clipPath|mask|
                 linearGradient|radialGradient|pattern|filter|cursor|font|
                 animate|set|animateMotion|animateColor|animateTransform|
                 color-profile|font-face
                 %glyphExt;)* >
<!ATTLIST glyph
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  unicode CDATA #IMPLIED
  glyph-name CDATA #IMPLIED
  d %PathData; #IMPLIED
  orientation CDATA #IMPLIED
  arabic-form CDATA #IMPLIED
  lang %LanguageCodes; #IMPLIED
  horiz-adv-x %Number; #IMPLIED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >
```

*Attribute definitions:*

**unicode = "<string>"**

One or more Unicode characters indicating the sequence of Unicode characters which corresponds to this glyph. If a character is provided, then this glyph corresponds to the given Unicode character. If multiple characters are provided, then this glyph corresponds to the given sequence of Unicode characters. One use of a sequence of characters is ligatures. For example, if **unicode="ffl"**, then the given glyph will be used to render the sequence of characters "f", "f", and "l".

It is often useful to refer to characters using XML character references expressed in hexadecimal notation or decimal notation. For example, **unicode="ffl"** could be expressed as XML character references in hexadecimal notation as **unicode="ffl"** or in decimal notation as **unicode="ffl"**.

The **unicode** attribute contributes to the process for deciding which glyph(s) are used to represent which character(s). See glyph selection rules. If the **unicode** attribute is not provided for a given **'glyph'**, then the only way to use this glyph is via an **'altGlyph'** reference.
*Animatable: no.*

**glyph-name = "<name> [, <name> ]* "**

A name for the glyph. It is recommended that glyph names be unique within a font. The glyph names can be used in situations where Unicode character numbers do not provide sufficient information to access the correct glyph, such as when there are multiple glyphs per Unicode character. The glyph names can be referenced in kerning definitions.
*Animatable: no.*

**d = "path data"**

The definition of the outline of a glyph, using the same syntax as for the **d** attribute on a **'path'**

element. See Path data.

See below for a discussion of this attribute.

*Animatable: no.*

**`orientation` = "h | v"**

Indicates that the given glyph is only to be used for a particular inline-progression-direction (i.e., horizontal or vertical). If the attribute is not specified, then the glyph can be used in all cases (i.e., both horizontal and vertical inline-progression-direction).

*Animatable: no.*

**`arabic-form` = "initial | medial | terminal | isolated"**

For Arabic glyphs, indicates which of the four possible forms this glyph represents.

*Animatable: no.*

**`lang` = "%LanguageCodes;"**

The attribute value is a comma-separated list of language names as defined in [RFC3066].

The glyph can be used if the **xml:lang** attribute exactly matches one of the languages given in the value of this parameter, or if the **xml:lang** attribute exactly equals a prefix of one of the languages given in the value of this parameter such that the first tag character following the prefix is "-".

*Animatable: no.*

**`horiz-adv-x` = "<number>"**

The horizontal advance after rendering the glyph in horizontal orientation. If the attribute is not specified, the effect is as if the attribute were set to the value of the font's **horiz-adv-x** attribute.

Glyph widths are required to be non-negative, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts.

*Animatable: no.*

**`vert-origin-x` = "<number>"**

The X-coordinate in the font coordinate system of the origin of the glyph to be used when drawing vertically oriented text.

If the attribute is not specified, the effect is as if the attribute were set to the value of the font's **vert-origin-x** attribute.

*Animatable: no.*

**`vert-origin-y` = "<number>"**

The Y-coordinate in the font coordinate system of the origin of a glyph to be used when drawing vertically oriented text.

If the attribute is not specified, the effect is as if the attribute were set to the value of the font's **vert-origin-y** attribute.

*Animatable: no.*

**`vert-adv-y` = "<number>"**

The vertical advance after rendering a glyph in vertical orientation.

If the attribute is not specified, the effect is as if the attribute were set to the value of the font's **vert-adv-y** attribute.

*Animatable: no.*

***Attributes defined elsewhere:***

**%stdAttrs;, style, class, %PresentationAttributes-All;.**

The graphics for the **'glyph'** can be specified using either the **d** attribute or arbitrary SVG as content within the **'glyph'**.

If the **d** attribute is specified, then the path data within this attribute is processed as follows:

- Any relative coordinates within the path data specification are converted into equivalent absolute coordinates
- Each of these absolute coordinates is transformed from the font coordinate system into the **'text'** element's current coordinate system such that the origin of the font coordinate system is properly positioned and rotated to align with the current text position and orientation for the glyph, and scaled so that the correct **'font-size'** is achieved.
- The resulting, transformed path specification is rendered as if it were a **'path'** element, using the styling properties that apply to the characters which correspond to the given glyph, and ignoring any styling properties specified on the **'font'** element or the **'glyph'** element.

If the **'glyph'** has child elements, then those child elements are rendered in a manner similar to how the **'use'** element renders a referenced symbol. The rendering effect is as if the contents of the referenced **'glyph'** element were deeply cloned into a separate non-exposed DOM tree. Because the cloned DOM tree is non-exposed, the SVG DOM does not show the cloned instance.

For user agents that support Styling with CSS, the conceptual deep cloning of the referenced **'glyph'** element into a non-exposed DOM tree also copies any property values resulting from the CSS cascade [CSS2-CASCADE] on the referenced **'glyph'** and its contents, and also applies any property values on the **'font'** element. CSS2 selectors can be applied to the original (i.e., referenced) elements because they are part of the formal document structure. CSS2 selectors cannot be applied to the (conceptually) cloned DOM tree because its contents are not part of the formal document structure.

Property inheritance, however, works as if the referenced **'glyph'** had been textually included as a deeply cloned child within the document tree. The referenced **'glyph'** inherits properties from the element that contains the characters that correspond to the **'glyph'**. The **'glyph'** does not inherit properties from the **'font'** element's original parents.

In the generated content, for each instance of a given **'glyph'**, a **'g'** is created which carries with it all property values resulting from the CSS cascade [CSS2-CASCADE] on the **'font'** element for the referenced **'glyph'**. Within this **'g'** is another **'g'** which carries with it all property values resulting from the CSS cascade [CSS2-CASCADE] on the **'glyph'** element. The original contents of the **'glyph'** element are deep-cloned within the inner **'g'** element.

If the **'glyph'** has both a **d** attribute and child elements, the **d** attribute is rendered first, and then the child elements.

In general, the **d** attribute renders in the same manner as system fonts. For example, a dashed pattern will usually look the same if applied to a system font or to an SVG font which defines its glyphs using the **d** attribute. Many implementations will be able to render glyphs defined with the **d** attribute quickly and will be able to use a font cache for further performance gains.

Defining a glyph by including child elements within the **'glyph'** gives greater flexibility but more complexity. Different fill and stroke techniques can be used on different parts of the glyphs. For example, the base of an "i" could be red, and the dot could be blue. This approach has an inherent complexity with units. Any properties specified on a text elements which represents a length, such as

the **'stroke-width'** property, might produce surprising results since the length value will be processed in the coordinate system of the glyph.

## 20.5 The 'missing-glyph' element

The **'missing-glyph'** element defines the graphics to use if there is an attempt to draw a glyph from a given font and the given glyph has not been defined. The attributes on the **'missing-glyph'** element have the same meaning as the corresponding attributes on the **'glyph'** element.

```
<!ENTITY % missing-glyphExt "" >
<!ELEMENT missing-glyph (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %missing-glyphExt;)* >
<!ATTLIST missing-glyph
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  d %PathData; #IMPLIED
  horiz-adv-x %Number; #IMPLIED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >
```

***Attributes defined elsewhere:***

**%stdAttrs;**, **class**, **style**, **%PresentationAttributes-All;**, **d**, **horiz-adv-x**, **vert-origin-x**, **vert-origin-y**, **vert-adv-y**.

## 20.6 Glyph selection rules

When determining the glyph(s) to draw a given character sequence, the **'font'** element is searched from its first **'glyph'** element to its last in logical order to see if the upcoming sequence of Unicode characters to be rendered matches the sequence of Unicode characters specified in the **unicode** attribute for the given **'glyph'** element. The first successful match is used. Thus, the "ffl" ligature needs to be defined in the font before the "f" glyph; otherwise, the "ffl" will never be selected.

Note that any occurrences of **'altGlyph'** take precedence over the above glyph selection rules within an SVG font.

## 20.7 The 'hkern' and 'vkern' elements

The **'hkern'** and **'vkern'** elements define kerning pairs for horizontally-oriented and vertically-oriented

pairs of glyphs, respectively.

Kern pairs identify pairs of glyphs within a single font whose inter-glyph spacing is adjusted when the pair of glyphs are rendered next to each other. In addition to the requirement that the pair of glyphs are from the same font, SVG font kerning happens only when the two glyphs correspond to characters which have the same values for properties **'font-family'**, **'font-size'**, **'font-style'**, **'font-weight'**, **'font-variant'**, **'font-stretch'**, **'font-size-adjust'** and **'font'**.

An example of a kerning pair are the letters "Va", where the typographic result might look better if the letters "V" and the "a" were rendered slightly closer together.

Right-to-left and bidirectional text in SVG is laid out in a two-step process, which is described in Relationship with bidirectionality. If SVG fonts are used, before kerning is applied, characters are re-ordered into left-to-right (or top-to-bottom, for vertical text) visual rendering order. Kerning from SVG fonts is then applied on pairs of glyphs which are rendered contiguously. The first glyph in the kerning pair is the left (or top) glyph in visual rendering order. The second glyph in the kerning pair is the right (or bottom) glyph in the pair.

For convenience to font designers and to minimize file sizes, a single **'hkern'** and **'vkern'** can define a single kerning adjustment value between one set of glyphs (e.g., a range of Unicode characters) and another set of glyphs (e.g., another range of Unicode characters).

The **'hkern'** element defines kerning pairs and adjustment values in the horizontal advance value when drawing pairs of glyphs which the two glyphs are contiguous and are both rendered horizontally (i.e., side-by-side). The spacing between characters is reduced by the kerning adjustment. (Negative kerning adjustments increase the spacing between characters.)

```
<!ELEMENT hkern EMPTY >
<!ATTLIST hkern
  %stdAttrs;
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k %Number; #REQUIRED >
```

*Attribute definitions:*

**u1 = "[<character> | <urange> ] [, [<character> | <urange>] ]* "**

A sequence (comma-separated) of Unicode characters (refer to the description of the **unicode** attribute to the **'glyph'** element for a description of how to express individual Unicode characters) and/or ranges of Unicode characters (see description of ranges of Unicode characters in [CSS2]) which identify a set of possible first glyphs in the kerning pair. If a given Unicode character within the set has multiple corresponding **'glyph'** elements (i.e., there are multiple **'glyph'** elements with the same **unicode** attribute value, but different **glyphName** values), then all such glyphs are included in the set. Comma is the separator character; thus, to kern a comma, specify the comma as part of a range of Unicode characters or as a glyph name using the **g1** attribute. The total set of possible first glyphs in

the kerning pair is the union of glyphs specified by the **u1** and **g1** attributes.
*Animatable: no.*

**g1 = "<name> [, <name> ]* "**
A sequence (comma-separated) of glyph names (i.e., values that match **glyphName** attributes on **'glyph'** elements) which identify a set of possible first glyphs in the kerning pair. All glyphs with the given glyph name are included in the set. The total set of possible first glyphs in the kerning pair is the union of glyphs specified by the **u1** and **g1** attributes.
*Animatable: no.*

**u2 = "[<number> | <urange>] [, [<number> | <urange>] ]* "**
Same as the **u1** attribute, except that **u2** specifies possible second glyphs in the kerning pair.
*Animatable: no.*

**g2 = "<name> [, <name> ]* "**
Same as the **g1** attribute, except that **g2** specifies possible second glyphs in the kerning pair.
*Animatable: no.*

**k = "<number>"**
The amount to decrease the spacing between the two glyphs in the kerning pair. The value is in the font coordinate system.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: no.*

***Attributes defined elsewhere:***
**%stdAttrs;.**

At least one each of **u1** or **g1** and at least one of **u2** or **g2** must be provided.

The **'vkern'** element defines kerning pairs and adjustment values in the vertical advance value when drawing pairs of glyphs together when stacked vertically. The spacing between characters is reduced by the kerning adjustment.

```
<!ELEMENT vkern EMPTY >
<!ATTLIST vkern
  %stdAttrs;
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k %Number; #REQUIRED >
```

***Attributes defined elsewhere:***
**%stdAttrs;, u1, g1, u2, g2, k.**

## 20.8 Describing a font

### 20.8.1 Overview of font descriptions

A font description provides the bridge between an author's font specification and the font data, which is the data needed to format text and to render the abstract glyphs to which the characters map - the actual scalable outlines or bitmaps. Fonts are referenced by properties, such as the **'font-family'** property.

Each specified font description is added to the font database and so that it can be used to select the relevant font data. The font description contains descriptors such as the location of the font data on the Web, and characterizations of that font data. The font descriptors are also needed to match the font properties to particular font data. The level of detail of a font description can vary from just the name of the font up to a list of glyph widths.

For more about font descriptions, refer to the font chapter in the CSS2 specification [CSS2 Fonts].

### 20.8.2 Alternative ways for providing a font description

Font descriptions can be specified in either of the following ways:

- a **'font-face'** element
- an @font-face rule within a CSS style sheet (only applicable for user agents which support using CSS to style the SVG content)

### 20.8.3 The 'font-face' element

The **'font-face'** element corresponds directly to the @font-face facility in CSS2. It can be used to describe the characteristics of any font, SVG font or otherwise.

When used to describe the characteristics of an SVG font contained within the same document, it is recommended that the **'font-face'** element be a child of the **'font'** element it is describing so that the **'font'** element can be self-contained and fully-described. In this case, any **'font-face-src'** elements within the **'font-face'** element are ignored as it is assumed that the **'font-face'** element is describing the characteristics of its parent **'font'** element.

```
<!ELEMENT font-face (%descTitleMetadata;,font-face-src?,definition-src?) >
<!ATTLIST font-face
  %stdAttrs;
  font-family CDATA #IMPLIED
  font-style CDATA #IMPLIED
  font-variant CDATA #IMPLIED
  font-weight CDATA #IMPLIED
  font-stretch CDATA #IMPLIED
```

```
font-size CDATA #IMPLIED
unicode-range CDATA #IMPLIED
units-per-em %Number; #IMPLIED
panose-1 CDATA #IMPLIED
stemv %Number; #IMPLIED
stemh %Number; #IMPLIED
slope %Number; #IMPLIED
cap-height %Number; #IMPLIED
x-height %Number; #IMPLIED
accent-height %Number; #IMPLIED
ascent %Number; #IMPLIED
descent %Number; #IMPLIED
widths CDATA #IMPLIED
bbox CDATA #IMPLIED
ideographic %Number; #IMPLIED
alphabetic %Number; #IMPLIED
mathematical %Number; #IMPLIED
hanging %Number; #IMPLIED
v-ideographic %Number; #IMPLIED
v-alphabetic %Number; #IMPLIED
v-mathematical %Number; #IMPLIED
v-hanging %Number; #IMPLIED
underline-position %Number; #IMPLIED
underline-thickness %Number; #IMPLIED
strikethrough-position %Number; #IMPLIED
strikethrough-thickness %Number; #IMPLIED
overline-position %Number; #IMPLIED
overline-thickness %Number; #IMPLIED >
```

*Attribute definitions:*

**`font-family` = "<string>"**
Same syntax and semantics as the **'font-family'** descriptor within an @font-face rule.
*Animatable: no.*

**`font-style` = "all | [ normal | italic | oblique] [, [normal | italic | oblique]]*"**
Same syntax and semantics as the **'font-style'** descriptor within an @font-face rule. The style of a font. Takes on the same values as the 'font-style' property, except that a comma-separated list is permitted.
If the attribute is not specified, the effect is as if a value of "all" were specified.
*Animatable: no.*

**`font-variant` = "[normal | small-caps] [,[normal | small-caps]]*"**
Same syntax and semantics as the **'font-variant'** descriptor within an @font-face rule.
Indication of whether this face is the small-caps variant of a font. Takes on the same values as the 'font-variant' property, except that a comma-separated list is permitted.
If the attribute is not specified, the effect is as if a value of "normal" were specified.
*Animatable: no.*

**`font-weight` = "all | [normal | bold |100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900] [, [normal | bold |100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]]*"**
Same syntax and semantics as the **'font-weight'** descriptor within an @font-face rule.
The weight of a face relative to others in the same font family. Takes on the same values as the 'font-weight' property with three exceptions:

- relative keywords (bolder, lighter) are not permitted
- a comma-separated list of values is permitted, for fonts that contain multiple weights

- an additional keyword, 'all', is permitted, which means that the font will match for all possible weights; either because it contains multiple weights, or because that face only has a single weight.

If the attribute is not specified, the effect is as if a value of "all" were specified.
*Animatable: no.*

**`font-stretch` = "all | [ normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded] [, [ normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded] ]*"**

Same syntax and semantics as the **'font-stretch'** descriptor within an @font-face rule.
Indication of the condensed or expanded nature of the face relative to others in the same font family. Takes on the same values as the 'font-stretch' property except that:

- relative keywords (wider, narrower) are not permitted
- a comma-separated list is permitted
- the keyword 'all' is permitted

If the attribute is not specified, the effect is as if a value of "normal" were specified.
*Animatable: no.*

**`font-size` = "<string>"**

Same syntax and semantics as the **'font-size'** descriptor within an @font-face rule.
*Animatable: no.*

**`unicode-range` = "<urange> [, <urange>]*"**

Same syntax and semantics as the **'unicode-range'** descriptor within an @font-face rule. The range of ISO 10646 characters [UNICODE] possibly covered by the glyphs in the font. Except for any additional information provided in this specification, the normative definition of the attribute is in [CSS2].
If the attribute is not specified, the effect is as if a value of "U+0-10FFFF" were specified.
*Animatable: no.*

**`units-per-em` = "<number>"**

Same syntax and semantics as the **'units-per-em'** descriptor within an @font-face rule. The number of coordinate units on the em square, the size of the design grid on which glyphs are laid out.
This value is almost always necessary as nearly every other attribute requires the definition of a design grid.
If the attribute is not specified, the effect is as if a value of "1000" were specified.
*Animatable: no.*

**`panose-1` = "[<integer>]{10}"**

Same syntax and semantics as the **'panose-1'** descriptor within an @font-face rule. The Panose-1 number, consisting of ten decimal integers, separated by whitespace. Except for any additional information provided in this specification, the normative definition of the attribute is in [CSS2].
If the attribute is not specified, the effect is as if a value of "0 0 0 0 0 0 0 0 0 0" were specified.
*Animatable: no.*

**`stemv` = "<number>"**

Same syntax and semantics as the **'stemv'** descriptor within an @font-face rule.
*Animatable: no.*

**`stemh` = "<number>"**

Same syntax and semantics as the **'stemh'** descriptor within an @font-face rule.
*Animatable: no.*

**`slope` = "<number>"**

Same syntax and semantics as the **'slope'** descriptor within an @font-face rule. The vertical stroke angle of the font. Except for any additional information provided in this specification,

the normative definition of the attribute is in [CSS2].
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: no.*

**`cap-height` = "<number>"**
Same syntax and semantics as the **'cap-height'** descriptor within an @font-face rule. The height of uppercase glyphs in the font within the font coordinate system.
*Animatable: no.*

**`x-height` = "<number>"**
Same syntax and semantics as the **'x-height'** descriptor within an @font-face rule. The height of lowercase glyphs in the font within the font coordinate system.
*Animatable: no.*

**`accent-height` = "<number>"**
The distance from the origin to the top of accent characters, measured by a distance within the font coordinate system.
If the attribute is not specified, the effect is as if the attribute were set to the value of the **ascent** attribute. If this attribute is used, the **units-per-em** attribute must also be specified.
*Animatable: no.*

**`ascent` = "<number>"**
Same syntax and semantics as the **'ascent'** descriptor within an @font-face rule. The maximum unaccented height of the font within the font coordinate system.
If the attribute is not specified, the effect is as if the attribute were set to the difference between the **units-per-em** value and the **vert-origin-y** value for the corresponding font.
*Animatable: no.*

**`descent` = "<number>"**
Same syntax and semantics as the **'descent'** descriptor within an @font-face rule. The maximum unaccented depth of the font within the font coordinate system.
If the attribute is not specified, the effect is as if the attribute were set to the **vert-origin-y** value for the corresponding font.
*Animatable: no.*

**`widths` = "<string>"**
Same syntax and semantics as the **'widths'** descriptor within an @font-face rule.
*Animatable: no.*

**`bbox` = "<string>"**
Same syntax and semantics as the **'bbox'** descriptor within an @font-face rule.
*Animatable: no.*

**`ideographic` = "<number>"**
For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve ideographic baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.
*Animatable: no.*

**`alphabetic` = "<number>"**
Same syntax and semantics as the **'baseline'** descriptor within an @font-face rule. For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve alphabetic baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.
*Animatable: no.*

**`mathematical` = "<number>"**
Same syntax and semantics as the **'mathline'** descriptor within an @font-face rule. For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve mathematical baseline alignment. The value is an offset in the font coordinate system. If this

attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`hanging` = "<number>"**

For horizontally oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve hanging baseline alignment. The value is an offset in the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`v-ideographic` = "<number>"**

For vertically oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve ideographic baseline alignment. The value is an offset in the font coordinate system relative to the glyph-specific **vert-origin-x** attribute. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`v-alphabetic` = "<number>"**

For vertically oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve alphabetic baseline alignment. The value is an offset in the font coordinate system relative to the glyph-specific **vert-origin-x** attribute. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`v-mathematical` = "<number>"**

For vertically oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve mathematical baseline alignment. The value is an offset in the font coordinate system relative to the glyph-specific **vert-origin-x** attribute. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`v-hanging` = "<number>"**

For vertically oriented glyph layouts, indicates the alignment coordinate for glyphs to achieve hanging baseline alignment. The value is an offset in the font coordinate system relative to the glyph-specific **vert-origin-x** attribute. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`underline-position` = "<number>"**

The ideal position of an underline within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`underline-thickness` = "<number>"**

The ideal thickness of an underline, expressed as a length within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`strikethrough-position` = "<number>"**

The ideal position of a strike-through within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`strikethrough-thickness` = "<number>"**

The ideal thickness of a strike-through, expressed as a length within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`overline-position` = "<number>"**

The ideal position of an overline within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

**`overline-thickness` = "<number>"**

The ideal thickness of an overline, expressed as a length within the font coordinate system. If this attribute is provided, the **units-per-em** attribute must also be specified.

*Animatable: no.*

***Attributes defined elsewhere:***

**%stdAttrs;.**

The following elements and attributes correspond to the **'src'** descriptor within an @font-face rule. (Refer to the descriptions of the [@font-face rule] and ['src' descriptor] in the CSS2 specification.)

When a **'font-face-uri'** is referencing an SVG font, then that reference must be to an SVG **'font'** element, therefore requiring the use of a fragment identifier (see [URI]). The referenced **'font'** element can be local (i.e., within the same document as the **'font-face-uri'** element) or remote (i.e., within a different document).

```
<!ELEMENT font-face-src (font-face-uri|font-face-name)+ >
<!ATTLIST font-face-src
  %stdAttrs; >

<!ELEMENT font-face-uri (font-face-format*) >
<!ATTLIST font-face-uri
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED >

<!ELEMENT font-face-format EMPTY >
<!ATTLIST font-face-format
  %stdAttrs;
  string CDATA #IMPLIED >

<!ELEMENT font-face-name EMPTY >
<!ATTLIST font-face-name
  %stdAttrs;
  name CDATA #IMPLIED >
```

The **'definition-src'** element corresponds to the *'definition-src'* descriptor in CSS2. (Refer to description of the ['definition-src' descriptor] in CSS2 specification.)

```
<!ELEMENT definition-src EMPTY >
<!ATTLIST definition-src
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED >
```

***Attributes defined elsewhere:***

**%stdAttrs;, %xlinkRefAttrs;, xlink:href.**

## 20.9 DOM interfaces

The following interfaces are defined below: SVGFontElement, SVGGlyphElement, SVGMissingGlyphElement, SVGHKernElement, SVGVKernElement, SVGFontFaceElement, SVGFontFaceSrcElement, SVGFontFaceUriElement, SVGFontFaceFormatElement, SVGFontFaceNameElement, SVGDefinitionSrcElement.

### Interface SVGFontElement

The **SVGFontElement** interface corresponds to the **'font'** element.

Object-oriented access to the attributes of the **'font'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGFontElement :
                SVGElement,
                SVGExternalResourcesRequired,
                SVGStylable {};
```

### Interface SVGGlyphElement

The **SVGGlyphElement** interface corresponds to the **'glyph'** element.

Object-oriented access to the attributes of the **'glyph'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGGlyphElement :
                SVGElement,
                SVGStylable {};
```

### Interface SVGMissingGlyphElement

The **SVGMissingGlyphElement** interface corresponds to the **'missing-glyph'** element.

Object-oriented access to the attributes of the **'missing-glyph'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGMissingGlyphElement :
                SVGElement,
                SVGStylable {};
```

## Interface SVGHKernElement

The **SVGHKernElement** interface corresponds to the **'hkern'** element.

Object-oriented access to the attributes of the **'hkern'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGHKernElement : SVGElement {};
```

## Interface SVGVKernElement

The **SVGVKernElement** interface corresponds to the **'vkern'** element.

Object-oriented access to the attributes of the **'vkern'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGVKernElement : SVGElement {};
```

## Interface SVGFontFaceElement

The **SVGFontFaceElement** interface corresponds to the **'font-face'** element.

Object-oriented access to the attributes of the **'font-face'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGFontFaceElement : SVGElement {};
```

## Interface SVGFontFaceSrcElement

The **SVGFontFaceSrcElement** interface corresponds to the **'font-face-src'** element.

**IDL Definition**

```
interface SVGFontFaceSrcElement : SVGElement {};
```

## Interface SVGFontFaceUriElement

The **SVGFontFaceUriElement** interface corresponds to the **'font-face-uri'** element.

Object-oriented access to the attributes of the **'font-face-uri'** element via the SVG DOM is not available.

**IDL Definition**

```

```

```
interface SVGFontFaceUriElement : SVGElement {};
```

## Interface SVGFontFaceFormatElement

The **SVGFontFaceFormatElement** interface corresponds to the **'font-face-format'** element.

Object-oriented access to the attributes of the **'font-face-format'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGFontFaceFormatElement : SVGElement {};
```

## Interface SVGFontFaceNameElement

The **SVGFontFaceNameElement** interface corresponds to the **'font-face-name'** element.

Object-oriented access to the attributes of the **'font-face-name'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGFontFaceNameElement : SVGElement {};
```

## Interface SVGDefinitionSrcElement

The **SVGDefinitionSrcElement** interface corresponds to the **'definition-src'** element.

Object-oriented access to the attributes of the **'definition-src'** element via the SVG DOM is not available.

**IDL Definition**

```
interface SVGDefinitionSrcElement : SVGElement {};
```

---

04 September 2001

# 21 Metadata

## Contents



## 21.1 Introduction

Metadata is structured data about data.

In the computing industry, there are ongoing standardization efforts towards metadata with the goal of promoting industry interoperability and efficiency. Content creators should track these developments and include appropriate metadata in their SVG content which conforms to these various metadata standards as they emerge.

The W3C has a Semantic Web Activity which has been established to serve a leadership role, in both the design of enabling specifications and the open, collaborative development of technologies that support the automation, integration and reuse of data across various applications. The Semantic Web Activity builds upon the earlier W3C Metadata Activity, including the definition of Resource Description Framework (RDF). The specifications for RDF can be found at:

- Resource Description Framework Model and Syntax Specification
- Resource Description Framework (RDF) Schema Specification

Another activity relevant to most applications of metadata is the Dublin Core, which is a set of generally applicable core metadata properties (e.g., Title, Creator/Author, Subject, Description, etc.).

Individual industries or individual content creators are free to define their own metadata schema but

are encouraged to follow existing metadata standards and use standard metadata schema wherever possible to promote interchange and interoperability. If a particular standard metadata schema does not meet your needs, then it is usually better to define an additional metadata schema in an existing framework such as RDF and to use custom metadata schema in combination with standard metadata schema, rather than totally ignore the standard schema.

## 21.2 The 'metadata' element

Metadata which is included with SVG content should be specified within **'metadata'** elements. The contents of the **'metadata'** should be elements from other XML namespaces, with these elements from these namespaces expressed in a manner conforming with the "Namespaces in XML" Recommendation [XML-NS].

Authors should provide a **'metadata'** child element to the outermost **'svg'** element within a stand-alone SVG document. The **'metadata'** child element to an **'svg'** element serves the purposes of identifying document-level metadata.

The DTD definitions of many of SVG's elements (particularly, container and text elements) place no restriction on the placement or number of the **'metadata'** sub-elements. This flexibility is only present so that there will be a consistent content model for container elements, because some container elements in SVG allow for mixed content, and because the mixed content rules for XML [XML-MIXED] do not permit the desired restrictions. Representations of future versions of the SVG language might use more expressive representations than DTDs which allow for more restrictive mixed content rules. It is strongly recommended that at most one **'metadata'** element appear as a child of any particular element, and that this element appear before any other child elements (except possibly **'desc'** or **'title'** elements) or character data content. If metadata-processing user agents need to choose among multiple **'metadata'** elements for processing (e.g., to decide which string to use for a tooltip), the user agent shall choose the first one.

```
<!ENTITY % metadataExt "" >
<!ELEMENT metadata (#PCDATA %metadataExt;)* >
<!ATTLIST metadata
  %stdAttrs; >
```

*Attribute definitions:*

***Attributes defined elsewhere:***
**%stdAttrs;**.

## 21.3 An example

Here is an example of how metadata can be included in an SVG document. The example uses the Dublin Core version 1.1 schema. (Other XML-compatible metadata languages, including ones not based on RDF, can be used also.)

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
    xmlns = 'http://www.w3.org/2000/svg'>
    <desc xmlns:myfoo="http://example.org/myfoo">
      <myfoo:title>This is a financial report</myfoo:title>
      <myfoo:descr>The global description uses markup from the
        <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
      <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
      <myfoo:contains>$three $graph-bar</myfoo:contains>
        <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <rdf:RDF
           xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dc = "http://purl.org/dc/elements/1.1/" >
        <rdf:Description about="http://example.org/myfoo"
             dc:title="MyFoo Financial Report"
             dc:description="$three $bar $thousands $dollars $from 1998 $through 2000"
             dc:publisher="Example Organization"
             dc:date="2000-04-11"
             dc:format="image/svg+xml"
             dc:language="en" >
          <dc:creator>
            <rdf:Bag>
              <rdf:li>Irving Bird</rdf:li>
              <rdf:li>Mary Lambert</rdf:li>
            </rdf:Bag>
          </dc:creator>
        </rdf:Description>
      </rdf:RDF>
    </metadata>
</svg>
```

## 21.4 DOM interfaces

The following interfaces are defined below: SVGMetadataElement.

### Interface SVGMetadataElement

The **SVGMetadataElement** interface corresponds to the **'metadata'** element.

**IDL Definition**

```
interface SVGMetadataElement : SVGElement {};
```

---

04 September 2001

# 22 Backwards Compatibility

A user agent (UA) might not have the ability to process and view SVG content. The following list outlines two of the backwards compatibility scenarios associated with SVG content:

- For XML grammars with the ability to embed SVG content, it is assumed that some sort of alternate representation capability such as the 'switch' element and some sort of feature-availability test facility (such as what is described in the SMIL 1.0 specification [SMIL1]) will be available.

  This 'switch' element and feature-availability test facility (or their equivalents) are the recommended way for XML authors to provide an alternate representation to SVG content, such as an image or a text string. The following example shows how to embed an SVG drawing within a SMIL 1.0 document such that an alternate image will display in the event the user agent doesn't support SVG. Note that the MIME type in the "type" attribute is an important means for the user agent to decide if it can decode the referenced media.

  In this example, the SVG content is included via a URL reference. With some parent XML grammars it will also be possible to include an SVG document fragment inline within the same file as its parent grammar.

```
<?xml version="1.0" standalone="yes"?>
<smil>
  <body>
    <!-- With SMIL 1.0, the first child element of 'switch'
         which the SMIL 1.0 user agent is able to process
         and which tests true will get processed and all other
         child elements will have no visual effect. In this case,
         if the SMIL 1.0 user agent can process "image/svg+xml",
         then the SVG will appear; otherwise, the alternate image
         (the second child element) will appear. -->
    <switch>
      <!-- Render the SVG if possible. -->
      <ref type="image/svg+xml" src="drawing.svg" />

      <!-- Else, render the alternate image. -->
      <img src="alternate_image.jpg" />
    </switch>
  </body>
</smil>
```

- For HTML 4.0, SVG drawings can be embedded using the 'object' element. An alternate representation such as an image can be included as the content of the 'object' element.

In this case, the SVG content usually will be included via a URL reference. The following example shows how to use the 'object' element to include an SVG drawing via a URL reference with an image serving as the alternate representation in the absence of an SVG user agent:

```
<html>
  <body>
    <object type="image/svg+xml" data="drawing.svg">
      <!-- The contents of the 'object' element (i.e., an alternate
         image) are drawn in the event the user agent cannot process
         the SVG drawing. -->
      <img src="alternate_image.jpg" alt="alternate description" />
    </object>
  </body>
</html>
```

---

04 September 2001

# 23 Extensibility

## Contents



## 23.1 Foreign namespaces and private data

SVG allows inclusion of elements from foreign namespaces anywhere with the SVG content. In general, the SVG user agent will include the unknown elements in the DOM but will otherwise ignore unknown elements. (The notable exception is described under Embedding Foreign Object Types.)

Additionally, SVG allows inclusion of attributes from foreign namespaces on any SVG element. The SVG user agent will include unknown attributes in the DOM but with otherwise ignore unknown attributes.

SVG's ability to include foreign namespaces can be used for the following purposes:

- Application-specific information so that authoring applications can include model-level data in the SVG content to serve their "roundtripping" purposes (i.e., the ability to write, then read a file without loss of higher-level information).
- Supplemental data for extensibility. For example, suppose you have an extrusion extension which takes any 2D graphics and extrudes it in three dimensions. When applying the extrusion extension, you probably will need to set some parameters. The parameters can be included in the SVG content by inserting elements from an extrusion extension namespace.

To illustrate, a business graphics authoring application might want to include some private data within an SVG document so that it could properly reassemble the chart (a pie chart in this case) upon reading it back in:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
     xmlns = 'http://www.w3.org/2000/svg'>
  <defs>
    <myapp:piechart xmlns:myapp="http://example.org/myapp"
                    title="Sales by Region">
      <myapp:pieslice label="Northern Region" value="1.23"/>
      <myapp:pieslice label="Eastern Region" value="2.53"/>
      <myapp:pieslice label="Southern Region" value="3.89"/>
      <myapp:pieslice label="Western Region" value="2.04"/>
      <!-- Other private data goes here -->
    </myapp:piechart>
  </defs>
  <desc>This chart includes private data in another namespace
  </desc>
  <!-- In here would be the actual SVG graphics elements which
       draw the pie chart -->
</svg>
```

## 23.2 Embedding foreign object types

One goal for SVG is to provide a mechanism by which other XML language processors can render into an area within an SVG drawing, with those renderings subject to the various transformations and compositing parameters that are currently active at a given point within the SVG content tree. One particular example of this is to provide a frame for XML content styled with CSS or XSL so that dynamically reflowing text (subject to SVG transformations and compositing) could be inserted into the middle of some SVG content. Another example is inserting a MathML expression into an SVG drawing.

The **'foreignObject'** element allows for inclusion of a foreign namespace which has its graphical content drawn by a different user agent. The included foreign graphical content is subject to SVG transformations and compositing.

The contents of **'foreignObject'** are assumed to be from a different namespace. Any SVG elements within a **'foreignObject'** will not be drawn, except in the situation where a properly defined SVG subdocument with a proper **xmlns** (see "Namespaces in XML" [XML-NS]) attribute specification is embedded recursively. One situation where this can occur is when an SVG document fragment is embedded within another non-SVG document fragment, which in turn is embedded within an SVG document fragment (e.g., an SVG document fragment contains an XHTML document fragment which in turn contains yet another SVG document fragment).

Usually, a **'foreignObject'** will be used in conjunction with the **'switch'** element and the **requiredExtensions** attribute to provide proper checking for user agent support and provide an alternate rendering in case user agent support is not available.

## 23.3 The 'foreignObject' element

```
<!ENTITY % foreignObjectExt "" >
<!ELEMENT foreignObject (#PCDATA %ceExt;%foreignObjectExt;)* >
<!ATTLIST foreignObject
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED
  %StructuredText; >
```

*Attribute definitions:*

**x = "<coordinate>"**

The x-axis coordinate of one corner of the rectangular region into which the graphics associated with the contents of the **'foreignObject'** will be rendered.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**y = "<coordinate>"**

The y-axis coordinate of one corner of the rectangular region into which the referenced document is placed.
If the attribute is not specified, the effect is as if a value of "0" were specified.
*Animatable: yes.*

**`width` = "<length>"**

The width of the rectangular region into which the referenced document is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

**`height` = "<length>"**

The height of the rectangular region into which the referenced document is placed.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
*Animatable: yes.*

***Attributes defined elsewhere:***

**%stdAttrs;, %langSpaceAttrs;, class, transform, %graphicsElementEvents;, %testAttrs;, externalResourcesRequired, style, %PresentationAttributes-All;.**

## 23.4 An example

Here is an example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns = 'http://www.w3.org/2000/svg'>
  <desc>This example uses the 'switch' element to provide a
        fallback graphical representation of an paragraph, if
        XMHTML is not supported.</desc>
  <!-- The 'switch' element will process the first child element
       whose testing attributes evaluate to true.-->
  <switch>

    <!-- Process the embedded XHTML if the requiredExtensions attribute
         evaluates to true (i.e., the user agent supports XHTML
         embedded within SVG). -->
    <foreignObject width="100" height="50"
                   requiredExtensions="http://example.com/SVGExtensions/EmbeddedXHTML">
      <!-- XHTML content goes here -->
      <html xmlns="http://www.w3.org/1999/xhtml">
      <body>
        <p>Here is a paragraph that requires word wrap</p>
      </body>
      </html>
    </foreignObject>

    <!-- Else, process the following alternate SVG.
         Note that there are no testing attributes on the 'text' element.
         If no testing attributes are provided, it is as if there
         were testing attributes and they evaluated to true.-->
    <text font-size="10" font-family="Verdana">
      <tspan x="10" y="10">Here is a paragraph that</tspan>
      <tspan x="10" y="20">requires word wrap.</tspan>
    </text>

  </switch>
</svg>
```

It is not required that SVG user agent support the ability to invoke other arbitrary user agents to handle embedded foreign object types; however, all conforming SVG user agents would need to support the **'switch'** element and must be able to render valid SVG elements when they appear as one of the alternatives within a **'switch'** element.

Ultimately, it is expected that commercial Web browsers will support the ability for SVG to embed content from other XML grammars which use CSS or XSL to format their content, with the resulting CSS- or XSL-formatted content subject to SVG transformations and compositing. At this time, such a capability is not a requirement.

## 23.5 Adding private elements and attributes to the DTD

The SVG DTD allows for extending the SVG language within the internal DTD subset. Within the internal DTD subset, you have the ability to add custom elements and attributes to most SVG elements.

The DTD defines an extension entity for most of SVG elements. For example, the **'view'** element is defined in the DTD as follows:

```
<!ENTITY % viewExt "" >
<!ELEMENT view (%descTitleMetadata;%viewExt;) >
<!ATTLIST view
  %stdAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  zoomAndPan (disable | magnify) 'magnify'
  viewTarget CDATA #IMPLIED >
```

The entity `viewExt` can be defined in the internal DTD subset to add custom sub-element or custom attributes to the **'view'** element within a given document. For example, the following extends the **'view'** element with an additional child element **'customNS:customElement'** and an additional attribute **customNS:customAttr**:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
[
<!ENTITY % viewExt "| customNS:customElement" >
<!ATTLIST view
  xmlns:customNS CDATA #FIXED "http://www.example.org/customNS"
  customNS:customAttr CDATA #IMPLIED >

<!ELEMENT customNS:customElement EMPTY>
<!ATTLIST customNS:customElement
  xmlns:customNS CDATA #FIXED "http://www.example.org/customNS"
  info CDATA #IMPLIED>
]>
<svg width="8cm" height="4cm"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Extend the 'view' element via the internal DTD subset</desc>

  <!-- Presumably, some great graphics would go here. -->

  <view viewBox="100 110 20 30" customNS:customAttr="123">
    <customNS:customElement info="abc"/>
  </view>
</svg>
```

## 23.6 DOM interfaces

The following interfaces are defined below: SVGForeignObjectElement.

### Interface SVGForeignObjectElement

The **SVGForeignObjectElement** interface corresponds to the **'foreignObject'** element.

### IDL Definition

```
interface SVGForeignObjectElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
};
```

### Attributes

**readonly SVGAnimatedLength x**
: Corresponds to attribute **x** on the given **'foreignObject'** element.

**readonly SVGAnimatedLength y**
: Corresponds to attribute **y** on the given **'foreignObject'** element.

**readonly SVGAnimatedLength width**
: Corresponds to attribute **width** on the given **'foreignObject'** element.

**readonly SVGAnimatedLength height**
: Corresponds to attribute **height** on the given **'foreignObject'** element.

---

04 September 2001

# Appendix A: DTD

## Contents



**This appendix is normative.**

```
<!-- ============================================================
  This is the DTD for SVG 1.0.

  The specification for SVG that corresponds to this DTD is available at:

    http://www.w3.org/TR/2001/REC-SVG-20010904/

  Copyright (c) 2000 W3C (MIT, INRIA, Keio), All Rights Reserved.

  For SVG 1.0:

    Namespace:
      http://www.w3.org/2000/svg

    Public identifier:
      PUBLIC "-//W3C//DTD SVG 1.0//EN"

    URI for the DTD:
      http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd
============================================================================ -->


<!-- ============================================================
     ENTITY DECLARATIONS: Data types
     ============================================================ -->

<!ENTITY % BaselineShiftValue "CDATA">
    <!-- 'baseline-shift' property/attribute value (e.g., 'baseline', 'sub', etc.) -->

<!ENTITY % Boolean "(false | true)">
    <!-- feature specification -->
```

```
<!ENTITY % ClassList "CDATA">
    <!-- list of classes -->

<!ENTITY % ClipValue "CDATA">
    <!-- 'clip' property/attribute value (e.g., 'auto', rect(...)) -->

<!ENTITY % ClipPathValue "CDATA">
    <!-- 'clip-path' property/attribute value (e.g., 'none', %URI;) -->

<!ENTITY % ClipFillRule "(nonzero | evenodd | inherit)">
    <!-- 'clip-rule' or fill-rule property/attribute value -->

<!ENTITY % ContentType "CDATA">
    <!-- media type, as per [RFC2045] -->

<!ENTITY % Coordinate "CDATA">
    <!-- a <coordinate> -->

<!ENTITY % Coordinates "CDATA">
    <!-- a list of <coordinate>s -->

<!ENTITY % Color "CDATA">
    <!-- a <color> value -->

<!ENTITY % CursorValue "CDATA">
    <!-- 'cursor' property/attribute value (e.g., 'crosshair', %URI;) -->

<!ENTITY % EnableBackgroundValue "CDATA">
    <!-- 'enable-background' property/attribute value (e.g., 'new', 'accumulate') -->

<!ENTITY % ExtensionList "CDATA">
    <!-- extension list specification -->

<!ENTITY % FeatureList "CDATA">
    <!-- feature list specification -->

<!ENTITY % FilterValue "CDATA">
    <!-- 'filter' property/attribute value (e.g., 'none', %URI;) -->

<!ENTITY % FontFamilyValue "CDATA">
    <!-- 'font-family' property/attribute value (i.e., list of fonts) -->

<!ENTITY % FontSizeValue "CDATA">
    <!-- 'font-size' property/attribute value -->

<!ENTITY % FontSizeAdjustValue "CDATA">
    <!-- 'font-size-adjust' property/attribute value -->

<!ENTITY % GlyphOrientationHorizontalValue "CDATA">
    <!-- 'glyph-orientation-horizontal' property/attribute value (e.g., <angle>) -->

<!ENTITY % GlyphOrientationVerticalValue "CDATA">
    <!-- 'glyph-orientation-vertical' property/attribute value (e.g., 'auto', <angle>) -->

<!ENTITY % Integer "CDATA">
    <!-- a <integer> -->

<!ENTITY % KerningValue "CDATA">
    <!-- 'kerning' property/attribute value (e.g., auto | <length>) -->

<!ENTITY % LanguageCode "NMTOKEN">
    <!-- a language code, as per [RFC3066] -->

<!ENTITY % LanguageCodes "CDATA">
    <!-- comma-separated list of language codes, as per [RFC3066] -->

<!ENTITY % Length "CDATA">
    <!-- a <length> -->

<!ENTITY % Lengths "CDATA">
    <!-- a list of <length>s -->

<!ENTITY % LinkTarget "NMTOKEN">
    <!-- link to this target -->

<!ENTITY % MarkerValue "CDATA">
    <!-- 'marker' property/attribute value (e.g., 'none', %URI;) -->

<!ENTITY % MaskValue "CDATA">
    <!-- 'mask' property/attribute value (e.g., 'none', %URI;) -->

<!ENTITY % MediaDesc "CDATA">
    <!-- comma-separated list of media descriptors. -->
```

```
<!ENTITY % Number "CDATA">
    <!-- a <number> -->

<!ENTITY % NumberOptionalNumber "CDATA">
    <!-- list of <number>s, but at least one and at most two -->

<!ENTITY % NumberOrPercentage "CDATA">
    <!-- a <number> or a  <percentage> -->

<!ENTITY % Numbers "CDATA">
    <!-- a list of <number>s -->

<!ENTITY % OpacityValue "CDATA">
    <!-- opacity value (e.g., <number>) -->

<!ENTITY % Paint "CDATA">
    <!-- a 'fill' or 'stroke' property/attribute value: <paint> -->

<!ENTITY % PathData "CDATA">
    <!-- a path data specification -->

<!ENTITY % Points "CDATA">
    <!-- a list of points -->

<!ENTITY % PreserveAspectRatioSpec "CDATA">
    <!-- 'preserveAspectRatio' attribute specification -->

<!ENTITY % Script "CDATA">
    <!-- script expression -->

<!ENTITY % SpacingValue "CDATA">
    <!-- 'letter-spacing' or 'word-spacing' property/attribute value (e.g., normal | <length>) -->

<!ENTITY % StrokeDashArrayValue "CDATA">
    <!-- 'stroke-dasharray' property/attribute value (e.g., 'none', list of <number>s) -->

<!ENTITY % StrokeDashOffsetValue "CDATA">
    <!-- 'stroke-dashoffset' property/attribute value (e.g., 'none', <legnth>) -->

<!ENTITY % StrokeMiterLimitValue "CDATA">
    <!-- 'stroke-miterlimit' property/attribute value (e.g., <number>) -->

<!ENTITY % StrokeWidthValue "CDATA">
    <!-- 'stroke-width' property/attribute value (e.g., <length>) -->

<!ENTITY % StructuredText
  "content CDATA #FIXED 'structured text'" >

<!ENTITY % StyleSheet "CDATA">
    <!-- style sheet data -->

<!ENTITY % SVGColor "CDATA">
    <!-- An SVG color value (RGB plus optional ICC) -->

<!ENTITY % Text "CDATA">
    <!-- arbitrary text string -->

<!ENTITY % TextDecorationValue "CDATA">
    <!-- 'text-decoration' property/attribute value (e.g., 'none', 'underline') -->

<!ENTITY % TransformList "CDATA">
    <!-- list of transforms -->

<!ENTITY % URI "CDATA">
    <!-- a Uniform Resource Identifier, see [URI] -->

<!ENTITY % ViewBoxSpec "CDATA">
    <!-- 'viewBox' attribute specification -->


<!-- ================================================================
     ENTITY DECLARATIONS: Collections of common attributes
     ================================================================ -->

<!-- All elements have an ID. -->
<!ENTITY % stdAttrs
 "id ID #IMPLIED
  xml:base %URI; #IMPLIED" >

<!-- Common attributes for elements that might contain character data content. -->
<!ENTITY % langSpaceAttrs
```

```
 "xml:lang %LanguageCode; #IMPLIED
  xml:space (default|preserve) #IMPLIED" >

<!-- Common attributes to check for system capabilities. -->
<!ENTITY % testAttrs
 "requiredFeatures %FeatureList; #IMPLIED
  requiredExtensions %ExtensionList; #IMPLIED
  systemLanguage %LanguageCodes; #IMPLIED" >

<!-- For most uses of URI referencing:
        standard XLink attributes other than xlink:href. -->
<!ENTITY % xlinkRefAttrs
 "xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'
  xlink:type (simple) #FIXED 'simple'
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (other) 'other'
  xlink:actuate (onLoad) #FIXED 'onLoad'" >

<!-- Standard XLink attributes for uses of URI referencing where xlink:show is 'embed' -->
<!ENTITY % xlinkRefAttrsEmbed
 "xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'
  xlink:type (simple) #FIXED 'simple'
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (embed) 'embed'
  xlink:actuate (onLoad) #FIXED 'onLoad'" >

<!ENTITY % graphicsElementEvents
  "onfocusin %Script; #IMPLIED
   onfocusout %Script; #IMPLIED
   onactivate %Script; #IMPLIED
   onclick %Script; #IMPLIED
   onmousedown %Script; #IMPLIED
   onmouseup %Script; #IMPLIED
   onmouseover %Script; #IMPLIED
   onmousemove %Script; #IMPLIED
   onmouseout %Script; #IMPLIED
   onload %Script; #IMPLIED" >

<!ENTITY % documentEvents
  "onunload %Script; #IMPLIED
   onabort %Script; #IMPLIED
   onerror %Script; #IMPLIED
   onresize %Script; #IMPLIED
   onscroll %Script; #IMPLIED
   onzoom %Script; #IMPLIED" >

<!ENTITY % animationEvents
  "onbegin %Script; #IMPLIED
   onend %Script; #IMPLIED
   onrepeat %Script; #IMPLIED" >

<!-- This entity allows for at most one of desc, title and metadata,
     supplied in any order -->
<!ENTITY % descTitleMetadata
          "(((desc,((title,metadata?)|(metadata,title?))?)|
          (title,((desc,metadata?)|(metadata,desc?))?)|
          (metadata,((desc,title?)|(title,desc?))?))?)" >


<!-- ================================================================
     ENTITY DECLARATIONS: Collections of presentation attributes
     ================================================================ -->

<!-- The following presentation attributes have to do with specifying color. -->
<!ENTITY % PresentationAttributes-Color
  "color %Color; #IMPLIED
   color-interpolation (auto | sRGB | linearRGB | inherit) #IMPLIED
   color-rendering (auto | optimizeSpeed | optimizeQuality | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to container elements. -->
<!ENTITY % PresentationAttributes-Containers
  "enable-background %EnableBackgroundValue; #IMPLIED " >

<!-- The following presentation attributes apply to 'feFlood' elements. -->
```

```
<!ENTITY % PresentationAttributes-feFlood
  "flood-color %SVGColor; #IMPLIED
   flood-opacity %OpacityValue; #IMPLIED " >

<!-- The following presentation attributes apply to filling and stroking operations. -->
<!ENTITY % PresentationAttributes-FillStroke
  "fill %Paint; #IMPLIED
   fill-opacity %OpacityValue; #IMPLIED
   fill-rule %ClipFillRule; #IMPLIED
   stroke %Paint; #IMPLIED
   stroke-dasharray %StrokeDashArrayValue; #IMPLIED
   stroke-dashoffset %StrokeDashOffsetValue; #IMPLIED
   stroke-linecap (butt | round | square | inherit) #IMPLIED
   stroke-linejoin (miter | round | bevel | inherit) #IMPLIED
   stroke-miterlimit %StrokeMiterLimitValue; #IMPLIED
   stroke-opacity %OpacityValue; #IMPLIED
   stroke-width %StrokeWidthValue; #IMPLIED " >

<!-- The following presentation attributes apply to filter primitives. -->
<!ENTITY % PresentationAttributes-FilterPrimitives
  "color-interpolation-filters (auto | sRGB | linearRGB | inherit) #IMPLIED " >

<!-- The following presentation attributes have to do with selecting a font to use. -->
<!ENTITY % PresentationAttributes-FontSpecification
  "font-family %FontFamilyValue; #IMPLIED
   font-size %FontSizeValue; #IMPLIED
   font-size-adjust %FontSizeAdjustValue; #IMPLIED
   font-stretch (normal | wider | narrower | ultra-condensed | extra-condensed |
                 condensed | semi-condensed | semi-expanded | expanded |
                 extra-expanded | ultra-expanded | inherit) #IMPLIED
   font-style (normal | italic | oblique | inherit) #IMPLIED
   font-variant (normal | small-caps | inherit) #IMPLIED
   font-weight (normal | bold | bolder | lighter | 100 | 200 | 300 |
                400 | 500 | 600 | 700 | 800 | 900 | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to gradient 'stop' elements. -->
<!ENTITY % PresentationAttributes-Gradients
  "stop-color %SVGColor; #IMPLIED
   stop-opacity %OpacityValue; #IMPLIED " >

<!-- The following presentation attributes apply to graphics elements. -->
<!ENTITY % PresentationAttributes-Graphics
  "clip-path %ClipPathValue; #IMPLIED
   clip-rule %ClipFillRule; #IMPLIED
   cursor %CursorValue; #IMPLIED
   display (inline | block | list-item | run-in | compact | marker |
            table | inline-table | table-row-group | table-header-group |
            table-footer-group | table-row | table-column-group | table-column |
            table-cell | table-caption | none | inherit) #IMPLIED
   filter %FilterValue; #IMPLIED
   image-rendering (auto | optimizeSpeed | optimizeQuality | inherit) #IMPLIED
   mask %MaskValue; #IMPLIED
   opacity %OpacityValue; #IMPLIED
   pointer-events (visiblePainted | visibleFill | visibleStroke | visible |
                   painted | fill | stroke | all | none | inherit) #IMPLIED
   shape-rendering (auto | optimizeSpeed | crispEdges | geometricPrecision | inherit) #IMPLIED
   text-rendering (auto | optimizeSpeed | optimizeLegibility | geometricPrecision | inherit) #IMPLIED
   visibility (visible | hidden | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to 'image' elements. -->
<!ENTITY % PresentationAttributes-Images
  "color-profile CDATA #IMPLIED " >

<!--The following presentation attributes apply to 'feDiffuseLighting' and 'feSpecularLighting' elements. -->
<!ENTITY % PresentationAttributes-LightingEffects
  "lighting-color %SVGColor; #IMPLIED " >

<!-- The following presentation attributes apply to marker operations. -->
<!ENTITY % PresentationAttributes-Markers
  "marker-start %MarkerValue; #IMPLIED
   marker-mid %MarkerValue; #IMPLIED
   marker-end %MarkerValue; #IMPLIED " >

<!-- The following presentation attributes apply to text content elements. -->
<!ENTITY % PresentationAttributes-TextContentElements
  "alignment-baseline (baseline | top | before-edge | text-top | text-before-edge |
                       middle | bottom | after-edge | text-bottom | text-after-edge |
                       ideographic | lower | hanging | mathematical | inherit) #IMPLIED
   baseline-shift %BaselineShiftValue; #IMPLIED
   direction (ltr | rtl | inherit) #IMPLIED
   dominant-baseline (auto | autosense-script | no-change | reset|
```

```
                    ideographic | lower | hanging | mathematical | inherit ) #IMPLIED
   glyph-orientation-horizontal %GlyphOrientationHorizontalValue; #IMPLIED
   glyph-orientation-vertical %GlyphOrientationVerticalValue; #IMPLIED
   kerning %KerningValue; #IMPLIED
   letter-spacing %SpacingValue; #IMPLIED
   text-anchor (start | middle | end | inherit) #IMPLIED
   text-decoration %TextDecorationValue; #IMPLIED
   unicode-bidi (normal | embed | bidi-override | inherit) #IMPLIED
   word-spacing %SpacingValue; #IMPLIED " >

<!-- The following presentation attributes apply to 'text' elements. -->
<!ENTITY % PresentationAttributes-TextElements
  "writing-mode (lr-tb | rl-tb | tb-rl | lr | rl | tb | inherit) #IMPLIED " >

<!-- The following presentation attributes apply to elements that establish viewports. -->
<!ENTITY % PresentationAttributes-Viewports
  "clip %ClipValue; #IMPLIED
   overflow (visible | hidden | scroll | auto | inherit) #IMPLIED " >

<!--The following represents the complete list of presentation attributes. -->
<!ENTITY % PresentationAttributes-All
  "%PresentationAttributes-Color;
   %PresentationAttributes-Containers;
   %PresentationAttributes-feFlood;
   %PresentationAttributes-FillStroke;
   %PresentationAttributes-FilterPrimitives;
   %PresentationAttributes-FontSpecification;
   %PresentationAttributes-Gradients;
   %PresentationAttributes-Graphics;
   %PresentationAttributes-Images;
   %PresentationAttributes-LightingEffects;
   %PresentationAttributes-Markers;
   %PresentationAttributes-TextContentElements;
   %PresentationAttributes-TextElements;
   %PresentationAttributes-Viewports;" >


<!-- ================================================================
     ENTITY DECLARATIONS: DTD extensions
     ================================================================ -->

<!-- Allow for extending the DTD with internal subset for
     container and graphics elements -->
<!ENTITY % ceExt "" >
<!ENTITY % geExt "" >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Document Structure
     ================================================================ -->

<!ENTITY % svgExt "" >
<!ELEMENT svg (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%svgExt;)* >
<!ATTLIST svg
  xmlns CDATA #FIXED "http://www.w3.org/2000/svg"
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  zoomAndPan (disable | magnify) 'magnify'
  %graphicsElementEvents;
  %documentEvents;
  version %Number; #FIXED "1.0"
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
```

```
  width %Length; #IMPLIED
  height %Length; #IMPLIED
  contentScriptType %ContentType; "text/ecmascript"
  contentStyleType %ContentType; "text/css" >

<!ENTITY % gExt "" >
<!ELEMENT g (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%gExt;)* >
<!ATTLIST g
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;  >

<!ENTITY % defsExt "" >
<!ELEMENT defs (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%defsExt;)* >
<!ATTLIST defs
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;  >

<!ENTITY % descExt "" >
<!ELEMENT desc (#PCDATA %descExt;)* >
<!ATTLIST desc
  %stdAttrs;
  %langSpaceAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %StructuredText; >

<!ENTITY % titleExt "" >
<!ELEMENT title (#PCDATA %titleExt;)* >
<!ATTLIST title
  %stdAttrs;
  %langSpaceAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %StructuredText; >

<!ENTITY % symbolExt "" >
<!ELEMENT symbol (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%symbolExt;)* >
```

```
<!ATTLIST symbol
  %stdAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  %graphicsElementEvents; >

<!ENTITY % useExt "" >
<!ELEMENT use (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                  %geExt;%useExt;)*) >
<!ATTLIST use
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >

<!ENTITY % imageExt "" >
<!ELEMENT image (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                  %geExt;%imageExt;)*) >
<!ATTLIST image
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Images;
  %PresentationAttributes-Viewports;
  transform %TransformList; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED >

<!ENTITY % switchExt "" >
<!ELEMENT switch (%descTitleMetadata;,
                 (path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|switch|a|foreignObject|
                  animate|set|animateMotion|animateColor|animateTransform
                  %ceExt;%switchExt;)*) >
<!ATTLIST switch
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
```

```
  %graphicsElementEvents; >


<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Styling
     ============================================================ -->

<!ELEMENT style (#PCDATA) >
<!ATTLIST style
  %stdAttrs;
  xml:space (preserve) #FIXED "preserve"
  type %ContentType; #REQUIRED
  media %MediaDesc; #IMPLIED
  title %Text; #IMPLIED >


<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Paths
     ============================================================ -->

<!ENTITY % pathExt "" >
<!ELEMENT path (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%pathExt;)*) >
<!ATTLIST path
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  d %PathData; #REQUIRED
  pathLength %Number; #IMPLIED >


<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Basic Shapes
     ============================================================ -->

<!ENTITY % rectExt "" >
<!ELEMENT rect (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%rectExt;)*) >
<!ATTLIST rect
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED
  rx %Length; #IMPLIED
  ry %Length; #IMPLIED >

<!ENTITY % circleExt "" >
<!ELEMENT circle (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%circleExt;)*) >
<!ATTLIST circle
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
```

```
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  r %Length; #REQUIRED >

<!ENTITY % ellipseExt "" >
<!ELEMENT ellipse (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%ellipseExt;)*) >
<!ATTLIST ellipse
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  rx %Length; #REQUIRED
  ry %Length; #REQUIRED >

<!ENTITY % lineExt "" >
<!ELEMENT line (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%lineExt;)*) >
<!ATTLIST line
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x1 %Coordinate; #IMPLIED
  y1 %Coordinate; #IMPLIED
  x2 %Coordinate; #IMPLIED
  y2 %Coordinate; #IMPLIED >

<!ENTITY % polylineExt "" >
<!ELEMENT polyline (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polylineExt;)*) >
<!ATTLIST polyline
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  points %Points; #REQUIRED >
```

```
<!ENTITY % polygonExt "" >
<!ELEMENT polygon (%descTitleMetadata;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polygonExt;)*) >
<!ATTLIST polygon
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-Markers;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  points %Points; #REQUIRED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Text
     ================================================================ -->

<!ENTITY % textExt "" >
<!ELEMENT text (#PCDATA|desc|title|metadata|
                tspan|tref|textPath|altGlyph|a|animate|set|
                animateMotion|animateColor|animateTransform
                %geExt;%textExt;)* >
<!ATTLIST text
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %PresentationAttributes-TextElements;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >

<!ENTITY % tspanExt "" >
<!ELEMENT tspan (#PCDATA|desc|title|metadata|tspan|tref|altGlyph|a|animate|set|animateColor
                %tspanExt;)* >
<!ATTLIST tspan
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
```

```
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >

<!ENTITY % trefExt "" >
<!ELEMENT tref (desc|title|metadata|animate|set|animateColor
                %trefExt;)* >
<!ATTLIST tref
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED >

<!ENTITY % textPathExt "" >
<!ELEMENT textPath (#PCDATA|desc|title|metadata|tspan|tref|altGlyph|a|animate|set|animateColor
                    %textPathExt;)* >
<!ATTLIST textPath
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %langSpaceAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  startOffset %Length; #IMPLIED
  textLength %Length; #IMPLIED
  lengthAdjust (spacing|spacingAndGlyphs) #IMPLIED
  method (align|stretch) #IMPLIED
  spacing (auto|exact) #IMPLIED >

<!ENTITY % altGlyphExt "" >
<!ELEMENT altGlyph (#PCDATA %altGlyphExt;)* >
<!ATTLIST altGlyph
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  glyphRef CDATA #IMPLIED
  format CDATA #IMPLIED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
```

```
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %graphicsElementEvents;
  x %Coordinates; #IMPLIED
  y %Coordinates; #IMPLIED
  dx %Lengths; #IMPLIED
  dy %Lengths; #IMPLIED
  rotate %Numbers; #IMPLIED >

<!ENTITY % altGlyphDefExt "" >
<!ELEMENT altGlyphDef ((glyphRef+|altGlyphItem+) %altGlyphDefExt;) >
<!ATTLIST altGlyphDef
  %stdAttrs; >

<!ENTITY % altGlyphItemExt "" >
<!ELEMENT altGlyphItem (glyphRef+ %altGlyphItemExt;) >
<!ATTLIST altGlyphItem
  %stdAttrs; >

<!ELEMENT glyphRef EMPTY >
<!ATTLIST glyphRef
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-FontSpecification;
  glyphRef CDATA #IMPLIED
  format CDATA #IMPLIED
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  dx %Number; #IMPLIED
  dy %Number; #IMPLIED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Painting: Filling, Stroking and Marker Symbols
     ================================================================ -->

<!ENTITY % markerExt "" >
<!ELEMENT marker (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%markerExt;)* >
<!ATTLIST marker
  %stdAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  refX %Coordinate; #IMPLIED
  refY %Coordinate; #IMPLIED
  markerUnits (strokeWidth | userSpaceOnUse) #IMPLIED
  markerWidth  %Length; #IMPLIED
  markerHeight %Length; #IMPLIED
  orient CDATA #IMPLIED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Color
     ================================================================ -->

<!ELEMENT color-profile (%descTitleMetadata;) >
<!ATTLIST color-profile
  %stdAttrs;
  %xlinkRefAttrs;
```

```
  xlink:href %URI; #IMPLIED
  local CDATA #IMPLIED
  name CDATA #REQUIRED
  rendering-intent (auto | perceptual | relative-colorimetric | saturation | absolute-colorimetric) "auto" >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Gradients and Patterns
     ================================================================ -->

<!ENTITY % linearGradientExt "" >
<!ELEMENT linearGradient (%descTitleMetadata;,(stop|animate|set|animateTransform
                  %linearGradientExt;)*) >
<!ATTLIST linearGradient
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  gradientUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  gradientTransform %TransformList; #IMPLIED
  x1 %Coordinate; #IMPLIED
  y1 %Coordinate; #IMPLIED
  x2 %Coordinate; #IMPLIED
  y2 %Coordinate; #IMPLIED
  spreadMethod (pad | reflect | repeat) #IMPLIED >


<!ENTITY % radialGradientExt "" >
<!ELEMENT radialGradient (%descTitleMetadata;,(stop|animate|set|animateTransform
                  %radialGradientExt;)*) >
<!ATTLIST radialGradient
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  gradientUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  gradientTransform %TransformList; #IMPLIED
  cx %Coordinate; #IMPLIED
  cy %Coordinate; #IMPLIED
  r %Length; #IMPLIED
  fx %Coordinate; #IMPLIED
  fy %Coordinate; #IMPLIED
  spreadMethod (pad | reflect | repeat) #IMPLIED >


<!ENTITY % stopExt "" >
<!ELEMENT stop (animate|set|animateColor
                  %stopExt;)* >
<!ATTLIST stop
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-Gradients;
  offset %NumberOrPercentage; #REQUIRED >

<!ENTITY % patternExt "" >
<!ELEMENT pattern (desc|title|metadata|defs|
                   path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|view|switch|a|altGlyphDef|
                   script|style|symbol|marker|clipPath|mask|
                   linearGradient|radialGradient|pattern|filter|cursor|font|
                   animate|set|animateMotion|animateColor|animateTransform|
                   color-profile|font-face
```

```
                  %ceExt;%patternExt;)* >
<!ATTLIST pattern
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  patternUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  patternContentUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  patternTransform %TransformList; #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Clipping, Masking and Compositing
     ================================================================ -->

<!ENTITY % clipPathExt "" >
<!ELEMENT clipPath (%descTitleMetadata;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%clipPathExt;)*) >
<!ATTLIST clipPath
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FillStroke;
  %PresentationAttributes-FontSpecification;
  %PresentationAttributes-Graphics;
  %PresentationAttributes-TextContentElements;
  %PresentationAttributes-TextElements;
  transform %TransformList; #IMPLIED
  clipPathUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED >

<!ENTITY % maskExt "" >
<!ELEMENT mask (desc|title|metadata|defs|
                  path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|view|switch|a|altGlyphDef|
                  script|style|symbol|marker|clipPath|mask|
                  linearGradient|radialGradient|pattern|filter|cursor|font|
                  animate|set|animateMotion|animateColor|animateTransform|
                  color-profile|font-face
                  %ceExt;%maskExt;)*  >
<!ATTLIST mask
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  maskUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  maskContentUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED >
```

```
<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Filter Effects
     ============================================================ -->

<!ENTITY % filterExt "" >
<!ELEMENT filter (%descTitleMetadata;,(feBlend|feFlood|
  feColorMatrix|feComponentTransfer|
  feComposite|feConvolveMatrix|feDiffuseLighting|feDisplacementMap|
  feGaussianBlur|feImage|feMerge|
  feMorphology|feOffset|feSpecularLighting|
  feTile|feTurbulence|
  animate|set
  %filterExt;)*) >
<!ATTLIST filter
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  filterUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  primitiveUnits (userSpaceOnUse | objectBoundingBox) #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #IMPLIED
  height %Length; #IMPLIED
  filterRes %NumberOptionalNumber; #IMPLIED >

<!ENTITY % filter_primitive_attributes
  "x %Coordinate; #IMPLIED
   y %Coordinate; #IMPLIED
   width %Length; #IMPLIED
   height %Length; #IMPLIED
   result CDATA #IMPLIED" >

<!ENTITY % filter_primitive_attributes_with_in
  "%filter_primitive_attributes;
   in CDATA #IMPLIED">

<!ELEMENT feDistantLight (animate|set)* >
<!ATTLIST feDistantLight
  %stdAttrs;
  azimuth %Number; #IMPLIED
  elevation %Number; #IMPLIED >

<!ELEMENT fePointLight (animate|set)* >
<!ATTLIST fePointLight
  %stdAttrs;
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  z %Number; #IMPLIED >

<!ELEMENT feSpotLight (animate|set)* >
<!ATTLIST feSpotLight
  %stdAttrs;
  x %Number; #IMPLIED
  y %Number; #IMPLIED
  z %Number; #IMPLIED
  pointsAtX %Number; #IMPLIED
  pointsAtY %Number; #IMPLIED
  pointsAtZ %Number; #IMPLIED
  specularExponent %Number; #IMPLIED
  limitingConeAngle %Number; #IMPLIED >

<!ELEMENT feBlend (animate|set)* >
<!ATTLIST feBlend
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
```

```
  in2 CDATA #REQUIRED
  mode (normal | multiply | screen | darken | lighten) "normal" >

<!ELEMENT feColorMatrix (animate|set)* >
<!ATTLIST feColorMatrix
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  type (matrix | saturate | hueRotate | luminanceToAlpha) "matrix"
  values CDATA #IMPLIED >

<!ELEMENT feComponentTransfer (feFuncR?,feFuncG?,feFuncB?,feFuncA?) >
<!ATTLIST feComponentTransfer
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >

<!ENTITY % component_transfer_function_attributes
  "type (identity | table | discrete | linear | gamma) #REQUIRED
   tableValues CDATA #IMPLIED
   slope %Number; #IMPLIED
   intercept %Number; #IMPLIED
   amplitude %Number; #IMPLIED
   exponent %Number; #IMPLIED
   offset %Number; #IMPLIED" >

<!ELEMENT feFuncR (animate|set)* >
<!ATTLIST feFuncR
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feFuncG (animate|set)* >
<!ATTLIST feFuncG
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feFuncB (animate|set)* >
<!ATTLIST feFuncB
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feFuncA (animate|set)* >
<!ATTLIST feFuncA
  %stdAttrs;
  %component_transfer_function_attributes; >

<!ELEMENT feComposite (animate|set)* >
<!ATTLIST feComposite
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  in2 CDATA #REQUIRED
  operator (over | in | out | atop | xor | arithmetic) "over"
  k1 %Number; #IMPLIED
  k2 %Number; #IMPLIED
  k3 %Number; #IMPLIED
  k4 %Number; #IMPLIED >

<!ELEMENT feConvolveMatrix (animate|set)* >
<!ATTLIST feConvolveMatrix
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  order %NumberOptionalNumber; #REQUIRED
  kernelMatrix CDATA #REQUIRED
  divisor %Number; #IMPLIED
  bias %Number; #IMPLIED
  targetX %Integer; #IMPLIED
  targetY %Integer; #IMPLIED
  edgeMode (duplicate|wrap|none) "duplicate"
  kernelUnitLength %NumberOptionalNumber; #IMPLIED
```

```
  preserveAlpha %Boolean; #IMPLIED >

<!ELEMENT feDiffuseLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feDiffuseLighting
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FilterPrimitives;
  %PresentationAttributes-LightingEffects;
  %filter_primitive_attributes_with_in;
  surfaceScale %Number; #IMPLIED
  diffuseConstant %Number; #IMPLIED
  kernelUnitLength %NumberOptionalNumber; #IMPLIED >

<!ELEMENT feDisplacementMap (animate|set)* >
<!ATTLIST feDisplacementMap
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  in2 CDATA #REQUIRED
  scale %Number; #IMPLIED
  xChannelSelector (R | G | B | A) "A"
  yChannelSelector (R | G | B | A) "A" >

<!ELEMENT feFlood (animate|set|animateColor)* >
<!ATTLIST feFlood
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-feFlood;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >

<!ELEMENT feGaussianBlur (animate|set)* >
<!ATTLIST feGaussianBlur
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  stdDeviation %NumberOptionalNumber; #IMPLIED >

<!ELEMENT feImage (animate|set|animateTransform)* >
<!ATTLIST feImage
  %stdAttrs;
  %xlinkRefAttrsEmbed;
  xlink:href %URI; #REQUIRED
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All; >

<!ELEMENT feMerge (feMergeNode)* >
<!ATTLIST feMerge
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes; >

<!ELEMENT feMergeNode (animate|set)* >
<!ATTLIST feMergeNode
  %stdAttrs;
  in CDATA #IMPLIED >

<!ELEMENT feMorphology (animate|set)* >
<!ATTLIST feMorphology
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  operator (erode | dilate) "erode"
  radius %NumberOptionalNumber; #IMPLIED >
```

```
<!ELEMENT feOffset (animate|set)* >
<!ATTLIST feOffset
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in;
  dx %Number; #IMPLIED
  dy %Number; #IMPLIED >

<!ELEMENT feSpecularLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feSpecularLighting
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-Color;
  %PresentationAttributes-FilterPrimitives;
  %PresentationAttributes-LightingEffects;
  %filter_primitive_attributes_with_in;
  surfaceScale %Number; #IMPLIED
  specularConstant %Number; #IMPLIED
  specularExponent %Number; #IMPLIED
  kernelUnitLength %NumberOptionalNumber; #IMPLIED >

<!ELEMENT feTile (animate|set)* >
<!ATTLIST feTile
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes_with_in; >

<!ELEMENT feTurbulence (animate|set)* >
<!ATTLIST feTurbulence
  %stdAttrs;
  %PresentationAttributes-FilterPrimitives;
  %filter_primitive_attributes;
  baseFrequency %NumberOptionalNumber; #IMPLIED
  numOctaves %Integer; #IMPLIED
  seed %Number; #IMPLIED
  stitchTiles (stitch | noStitch) "noStitch"
  type (fractalNoise | turbulence) "turbulence" >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Interactivity
     ================================================================ -->

<!ELEMENT cursor (%descTitleMetadata;) >
<!ATTLIST cursor
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Linking
     ================================================================ -->

<!ENTITY % aExt "" >
<!ELEMENT a       (#PCDATA|desc|title|metadata|defs|
                   path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|view|switch|a|altGlyphDef|
                   script|style|symbol|marker|clipPath|mask|
                   linearGradient|radialGradient|pattern|filter|cursor|font|
                   animate|set|animateMotion|animateColor|animateTransform|
                   color-profile|font-face
                   %ceExt;%aExt;)* >
<!ATTLIST a
  %stdAttrs;
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
  xlink:type (simple) #FIXED "simple"
```

```
  xlink:role %URI; #IMPLIED
  xlink:arcrole %URI; #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|replace) 'replace'
  xlink:actuate (onRequest) #FIXED 'onRequest'
  xlink:href %URI; #REQUIRED
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  target %LinkTarget; #IMPLIED >

<!ENTITY % viewExt "" >
<!ELEMENT view (%descTitleMetadata;%viewExt;) >
<!ATTLIST view
  %stdAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  viewBox %ViewBoxSpec; #IMPLIED
  preserveAspectRatio %PreserveAspectRatioSpec; 'xMidYMid meet'
  zoomAndPan (disable | magnify) 'magnify'
  viewTarget CDATA #IMPLIED >


<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Scripting
     ============================================================ -->

<!ELEMENT script (#PCDATA) >
<!ATTLIST script
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #IMPLIED
  externalResourcesRequired %Boolean; #IMPLIED
  type %ContentType; #REQUIRED >


<!-- ============================================================
     DECLARATIONS CORRESPONDING TO: Animation
     ============================================================ -->

<!ENTITY % animElementAttrs
 "%xlinkRefAttrs;
  xlink:href %URI; #IMPLIED" >

<!ENTITY % animAttributeAttrs
 "attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED" >

<!ENTITY % animTimingAttrs
 "begin CDATA #IMPLIED
  dur CDATA #IMPLIED
  end CDATA #IMPLIED
  min CDATA #IMPLIED
  max CDATA #IMPLIED
  restart (always | never | whenNotActive) 'always'
  repeatCount CDATA #IMPLIED
  repeatDur CDATA #IMPLIED
  fill (remove | freeze) 'remove'" >

<!ENTITY % animValueAttrs
 "calcMode (discrete | linear | paced | spline) 'linear'
  values CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED" >

<!ENTITY % animAdditionAttrs
```

```
 "additive       (replace | sum) 'replace'
  accumulate     (none | sum) 'none'" >

<!ENTITY % animateExt "" >
<!ELEMENT animate (%descTitleMetadata;%animateExt;) >
<!ATTLIST animate
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >

<!ENTITY % setExt "" >
<!ELEMENT set (%descTitleMetadata;%setExt;) >
<!ATTLIST set
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  to CDATA #IMPLIED >

<!ENTITY % animateMotionExt "" >
<!ELEMENT animateMotion (%descTitleMetadata;,mpath? %animateMotionExt;) >
<!ATTLIST animateMotion
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animTimingAttrs;
  calcMode (discrete | linear | paced | spline) 'paced'
  values CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  %animAdditionAttrs;
  path CDATA #IMPLIED
  keyPoints CDATA #IMPLIED
  rotate CDATA #IMPLIED
  origin CDATA #IMPLIED >

<!ENTITY % mpathExt "" >
<!ELEMENT mpath (%descTitleMetadata;%mpathExt;) >
<!ATTLIST mpath
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED
  externalResourcesRequired %Boolean; #IMPLIED >

<!ENTITY % animateColorExt "" >
<!ELEMENT animateColor (%descTitleMetadata;%animateColorExt;) >
<!ATTLIST animateColor
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >
```

```
<!ENTITY % animateTransformExt "" >
<!ELEMENT animateTransform (%descTitleMetadata;%animateTransformExt;) >
<!ATTLIST animateTransform
  %stdAttrs;
  %testAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  %animationEvents;
  %animElementAttrs;
  %animAttributeAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  type (translate | scale | rotate | skewX | skewY) "translate" >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Fonts
     ================================================================ -->

<!ENTITY % fontExt "" >
<!ELEMENT font (%descTitleMetadata;,font-face,
                missing-glyph,(glyph|hkern|vkern %fontExt;)*) >
<!ATTLIST font
  %stdAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  horiz-origin-x %Number; #IMPLIED
  horiz-origin-y %Number; #IMPLIED
  horiz-adv-x %Number; #REQUIRED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >

<!ENTITY % glyphExt "" >
<!ELEMENT glyph (desc|title|metadata|defs|
                 path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|view|switch|a|altGlyphDef|
                 script|style|symbol|marker|clipPath|mask|
                 linearGradient|radialGradient|pattern|filter|cursor|font|
                 animate|set|animateMotion|animateColor|animateTransform|
                 color-profile|font-face
                 %glyphExt;)* >
<!ATTLIST glyph
  %stdAttrs;
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  unicode CDATA #IMPLIED
  glyph-name CDATA #IMPLIED
  d %PathData; #IMPLIED
  orientation CDATA #IMPLIED
  arabic-form CDATA #IMPLIED
  lang %LanguageCodes; #IMPLIED
  horiz-adv-x %Number; #IMPLIED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >

<!ENTITY % missing-glyphExt "" >
<!ELEMENT missing-glyph (desc|title|metadata|defs|
                 path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|view|switch|a|altGlyphDef|
                 script|style|symbol|marker|clipPath|mask|
                 linearGradient|radialGradient|pattern|filter|cursor|font|
                 animate|set|animateMotion|animateColor|animateTransform|
                 color-profile|font-face
                 %missing-glyphExt;)* >
<!ATTLIST missing-glyph
  %stdAttrs;
```

```
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  d %PathData; #IMPLIED
  horiz-adv-x %Number; #IMPLIED
  vert-origin-x %Number; #IMPLIED
  vert-origin-y %Number; #IMPLIED
  vert-adv-y %Number; #IMPLIED >

<!ELEMENT hkern EMPTY >
<!ATTLIST hkern
  %stdAttrs;
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k %Number; #REQUIRED >

<!ELEMENT vkern EMPTY >
<!ATTLIST vkern
  %stdAttrs;
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k %Number; #REQUIRED >

<!ELEMENT font-face (%descTitleMetadata;,font-face-src?,definition-src?) >
<!ATTLIST font-face
  %stdAttrs;
  font-family CDATA #IMPLIED
  font-style CDATA #IMPLIED
  font-variant CDATA #IMPLIED
  font-weight CDATA #IMPLIED
  font-stretch CDATA #IMPLIED
  font-size CDATA #IMPLIED
  unicode-range CDATA #IMPLIED
  units-per-em %Number; #IMPLIED
  panose-1 CDATA #IMPLIED
  stemv %Number; #IMPLIED
  stemh %Number; #IMPLIED
  slope %Number; #IMPLIED
  cap-height %Number; #IMPLIED
  x-height %Number; #IMPLIED
  accent-height %Number; #IMPLIED
  ascent %Number; #IMPLIED
  descent %Number; #IMPLIED
  widths CDATA #IMPLIED
  bbox CDATA #IMPLIED
  ideographic %Number; #IMPLIED
  alphabetic %Number; #IMPLIED
  mathematical %Number; #IMPLIED
  hanging %Number; #IMPLIED
  v-ideographic %Number; #IMPLIED
  v-alphabetic %Number; #IMPLIED
  v-mathematical %Number; #IMPLIED
  v-hanging %Number; #IMPLIED
  underline-position %Number; #IMPLIED
  underline-thickness %Number; #IMPLIED
  strikethrough-position %Number; #IMPLIED
  strikethrough-thickness %Number; #IMPLIED
  overline-position %Number; #IMPLIED
  overline-thickness %Number; #IMPLIED >

<!ELEMENT font-face-src (font-face-uri|font-face-name)+ >
<!ATTLIST font-face-src
  %stdAttrs; >

<!ELEMENT font-face-uri (font-face-format*) >
<!ATTLIST font-face-uri
```

```
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED >

<!ELEMENT font-face-format EMPTY >
<!ATTLIST font-face-format
  %stdAttrs;
  string CDATA #IMPLIED >

<!ELEMENT font-face-name EMPTY >
<!ATTLIST font-face-name
  %stdAttrs;
  name CDATA #IMPLIED >

<!ELEMENT definition-src EMPTY >
<!ATTLIST definition-src
  %stdAttrs;
  %xlinkRefAttrs;
  xlink:href %URI; #REQUIRED >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Metadata
     ================================================================ -->

<!ENTITY % metadataExt "" >
<!ELEMENT metadata (#PCDATA %metadataExt;)* >
<!ATTLIST metadata
  %stdAttrs; >


<!-- ================================================================
     DECLARATIONS CORRESPONDING TO: Extensibility
     ================================================================ -->

<!ENTITY % foreignObjectExt "" >
<!ELEMENT foreignObject (#PCDATA %ceExt;%foreignObjectExt;)* >
<!ATTLIST foreignObject
  %stdAttrs;
  %testAttrs;
  %langSpaceAttrs;
  externalResourcesRequired %Boolean; #IMPLIED
  class %ClassList; #IMPLIED
  style %StyleSheet; #IMPLIED
  %PresentationAttributes-All;
  transform %TransformList; #IMPLIED
  %graphicsElementEvents;
  x %Coordinate; #IMPLIED
  y %Coordinate; #IMPLIED
  width %Length; #REQUIRED
  height %Length; #REQUIRED
  %StructuredText; >
```

---

04 September 2001

# Appendix B: SVG Document Object Model (DOM)

## Contents



**This appendix is normative.**

## B.1 SVG DOM Overview

This appendix provides an introduction to the SVG DOM and discusses the relationship of the SVG DOM with the Document Object Model (DOM) Level 2 Specification [DOM2]. The specific SVG DOM interfaces that correspond to particular sections of the SVG specification are defined at the end of corresponding chapters in this specification, as follows:

- Basic DOM interfaces
- Styling interfaces
- Document Structure interfaces
- Coordinate Systems, Transformations and Units interfaces
- Paths interfaces
- Basic Shapes interfaces
- Text interfaces
- Painting: Filling, Stroking and Marker Symbols interfaces
- Color interfaces

- Gradients and Patterns interfaces
- Clipping, Masking and Compositing interfaces
- Filter Effects interfaces
- Interactivity interfaces
- Linking interfaces
- Scripting interfaces
- Animation interfaces
- Fonts interfaces
- Metadata interfaces
- Extensibility interfaces

The SVG DOM is builds upon and is compatible with the Document Object Model (DOM) Level 2 Specification [DOM2]. In particular:

- The SVG DOM requires complete support for the DOM2 core [DOM2-CORE]
- Wherever appropriate, the SVG DOM is modeled after and maintains consistency with the *Document Object Model HTML* [DOM1-HTML].
- The SVG DOM requires complete support for the DOM2 views [DOM2-VIEWS]
- The SVG DOM requires support for relevant aspects of the DOM2 event model [DOM2-EVENTS]. (For the specific [DOM2-EVENTS] features that are required, see Relationship with DOM2 event model.)
- The traversal [DOM2-TRAV] and range [DOM2-RANGE] features from DOM2 are optional features within the SVG DOM.
- For implementations that support CSS, the SVG DOM requires complete support for the DOM2 style sheets [DOM2-SHEETS] and relevant aspects of the *Document Object Model CSS* [DOM2-CSS]. (For the specific [DOM2-CSS] features that are supported, see Relationship with DOM2 CSS object model.)

A DOM application can use the hasFeature method of the **DOMImplementation** interface to verify that the interfaces listed in this section are supported. The list of available interfaces is provided in section Feature strings for the **hasFeature** method call.

## B.2 Naming Conventions

The SVG DOM follows similar naming conventions to the Document Object Model HTML [DOM1-HTML].

All names are defined as one or more English words concatenated together to form a single string. Property or method names start with the initial keyword in lowercase, and each subsequent word starts with a capital letter. For example, a property that returns document meta information such as the date the file was created might be named "fileDateCreated". In the ECMAScript binding, properties are exposed as properties of a given object. In Java, properties are exposed with get and set methods.

For attributes with the CDATA data type, the case of the return value is that given in the source

document.

## B.3 Interface SVGException

**Exception *SVGException***

This exception is raised when a specific SVG operation is impossible to perform.

**IDL Definition**

```
exception SVGException {
  unsigned short   code;
};

// SVGExceptionCode
const unsigned short SVG_WRONG_TYPE_ERR           = 0;
const unsigned short SVG_INVALID_VALUE_ERR        = 1;
const unsigned short SVG_MATRIX_NOT_INVERTABLE    = 2;
```

## B.4 Feature strings for the hasFeature method call

The feature strings that are available for the **hasFeature** method call that is part of the SVG DOM's support for the **DOMImplementation** interface defined in [DOM2-CORE] are the same features strings available for the **requiredFeatures** attribute that is available for many SVG elements.

For all features that correspond to the SVG language and are documented in this specification (see section the **requiredFeatures** attribute for a list of features in the SVG language), the **version** number for the **hasFeature** method call is "1.0". For features that correspond to other languages, refer to the relevant other specifications to determine the appropriate version number for the given feature.

## B.5 Relationship with DOM2 events

The SVG DOM supports the following interfaces and event types from [DOM2-EVENTS]:

- The SVG DOM supports all of the interfaces defined in [DOM2-EVENTS].
- The SVG DOM supports the following UI event types [DOM2-UIEVENTS]:
  - DOMFocusIn
  - DOMFocusOut
  - DOMActivate
- The SVG DOM supports the following mouse event types [DOM2-MOUSEEVENTS]:
  - click
  - mousedown
  - mouseup
  - mouseover
  - mousemove
  - mouseout

  *clientX* and *clientY* parameters for mouse events represent viewport coordinates for the corresponding **'svg'** element. *relatedNode* is the corresponding outermost **'svg'** element.
- The SVG DOM supports the following mutation event types [DOM2-MUTEVENTS]:
  - DOMSubtreeModified
  - DOMNodeInserted
  - DOMNodeRemoved
  - DOMNodeRemovedFromDocument
  - DOMNodeInsertedIntoDocument
  - DOMAttrModified
  - DOMCharacterDataModified
- The SVG DOM defines the following SVG-specific custom event interfaces. These event interfaces are mandatory for SVG user agents:
  - SVGLoad
  - SVGUnload
  - SVGAbort
  - SVGError
  - SVGResize
  - SVGScroll (triggered by either scroll or pan user actions)
- The SVG DOM defines an additional custom event interface:
  - SVGZoom (definition can be found in the description of **SVGZoomEvent**)
- The following event types are triggered due to state changes in animations. (The definitions for these events can be found in the description of Interface **TimeEvent** in the SMIL Animation specification.)
  - beginEvemt
  - endEvent
  - repeatEvent

Each SVG element which has at least one event attribute assigned to it in the SVG DTD supports the DOM2 event registration interfaces [DOM2-EVREG] and can be registered as an event listener for the corresponding DOM2 event using the event registration interfaces. Thus, for example, if the SVG DTD indicates that a given element supports the "onclick" event attribute, then an event listener for the "click" event can be registered with the given element as the event target.

SVG's animation elements also support the DOM2 event registration interfaces [DOM2-EVREG]. Event listeners for animation events (i.e., start, end or repeat) can be registered on any of the animation elements.

Event listeners which are established by DOM2 Event registration interfaces [DOM2-EVREG] receive events before any event listeners that correspond to event attributes (see Event attributes) or animations.

In Java, one way that event listeners can be established is to define a class which implements the EventListener interface, such as:

```
class MyAction1 implements EventListener {
  public void handleEvent(Event evt) {
    // process the event
  }
}
// ... later ...
MyAction1 mc1 = new MyAction1();
myElement.addEventListener("DOMActivate", mc1, false);
```

In ECMAScript, one way to establish an event listener is to define a function and pass the name of that function to the addEventListener method:

```
function myAction1(evt) {
  // process the event
}
// ... later ...
myElement.addEventListener("DOMActivate", myAction1, false)
```

In ECMAScript, the character data content of an Event attribute become the definition of the ECMAScript function which gets invoked in response to the event. As with all registered ECMAScript event listener functions, this function receives an Event object as a parameter, and the name of the Event object is **evt**. For example, it is possible to say:

```
<rect onactivate="MyActivateHandler(evt)" .../>
```

which will pass the Event object **evt** into function `MyActivateHandler`.

## B.6 Relationship with DOM2 CSS object model (CSS OM)

### B.6.1 Introduction

The section describes the facilities from the Document Object Model CSS [DOM2-CSS], the CSS OM, that are part of the SVG DOM.

### B.6.2 User agents that do not support styling with CSS

User agents that do not support styling with CSS are only required to support the following interfaces from [DOM2-CSS], along with any interfaces necessary to implement the interfaces, such as **CSSPrimitiveValue** and **CSSValueList**. These interfaces are used in conjunction with the getPresentationAttribute method call on interface **SVGStylable**. This method must be supported on all implementations of the SVG DOM:

- Interface RGBColor
- Interface CSSValue

### B.6.3 User agents that support styling with CSS

User agents that support Styling with CSS, the SVG DOM, and aural styling [CSS2-AURAL] must support all of the interfaces defined in [DOM2-CSS] which apply to aural properties.

For visual media [CSS2-VISUAL], the SVG DOM supports all of the required interfaces defined in [DOM2-CSS]. All of the interfaces that are optional for [DOM2-CSS] are also optional for the SVG DOM.

### B.6.4 Extended interfaces

Whether or not a user agent supports styling with CSS, a user agent still must support interface CSSValue, as this is the type that is returned from the getPresentationAttribute method call on interface **SVGStylable**.

[DOM2-CSS] defines a set of extended interfaces [DOM2-CSS-EI] for use in conjunction with interface CSSValue. The table below specifies the type of CSSValue [DOM2-CSSVALUE] used to represent each SVG property that applies to visual media [CSS2-VISUAL]. The expectation is that the CSSValue returned from the getPropertyCSSValue method on the **CSSStyleDeclaration** interface or the getPresentationAttribute method on the **SVGStylable** interface can be cast down, using binding-specific casting methods, to the specific derived interface.

For properties that are represented by a custom interface (the valueType of the CSSValue is CSS_CUSTOM), the name of the derived interface is specified in the table. For these properties, the table below indicates which extended interfaces are mandatory and which are not.

For properties that consist of lists of values (the valueType of the CSSValue is CSS_VALUE_LIST), the derived interface is CSSValueList. For all other properties (the valueType of the CSSValue is CSS_PRIMITIVE_VALUE), the derived interface is CSSPrimitiveValue.

For shorthand properties, a CSSValue always will have a value of null. Shorthand property values can only be accessed and modified as strings.

The SVG DOM defines the following SVG-specific custom property interfaces, all of which are mandatory for SVG user agents:

- **SVGColor**
- **SVGICCColor**
- **SVGPaint**

| Property Name | Representation | (Extended interfaces only) Mandatory? |
|---|---|---|
| 'alignment-baseline' | ident | |
| 'baseline-shift' | ident, length, percentage | |
| 'clip' | rect, ident | |
| 'clip-path' | uri, ident | |
| 'clip-rule' | ident | |
| 'color' | rgbcolor, ident | |
| 'color-interpolation' | ident | |
| 'color-profile' | list of strings, uri's and idents | |
| 'color-rendering' | ident | |
| 'cursor' | uri, ident | no |
| 'direction' | ident | |
| 'display' | ident | |
| 'dominant-baseline' | ident | |
| 'enable-background' | list of idents and numbers | |
| 'fill' | **SVGPaint** | yes |
| 'fill-opacity' | number | |
| 'fill-rule' | ident | |
| 'filter' | uri, ident | |
| 'flood-color' | **SVGColor** | yes |
| 'flood-opacity' | number | |
| 'font' | null | |

| | | |
|---|---|---|
| 'font-family' | list of strings and idents | |
| 'font-size' | ident, length, percentage | |
| 'font-size-adjust' | number, ident | |
| 'font-stretch' | ident | |
| 'font-style' | ident | |
| 'font-variant' | ident | |
| 'font-weight' | ident | |
| 'glyph-orientation-horizontal' | ident | |
| 'glyph-orientation-vertical' | ident | |
| 'image-rendering' | ident | |
| 'kerning' | ident, length | |
| 'letter-spacing' | ident, length | |
| 'lighting-color' | **SVGColor** | yes |
| 'marker' | null | |
| 'marker-end' | uri, ident | |
| 'marker-mid' | uri, ident | |
| 'marker-start' | uri, ident | |
| 'mask' | uri, ident | |
| 'opacity' | number | |
| 'overflow' | ident | |
| 'pointer-events' | ident | |
| 'shape-rendering' | ident | |
| 'stop-color' | **SVGColor** | yes |
| 'stop-opacity' | number | |
| 'stroke' | **SVGPaint** | yes |
| 'stroke-dasharray' | ident or list of lengths | |
| 'stroke-dashoffset' | length | |
| 'stroke-linecap' | ident | |
| 'stroke-linejoin' | ident | |
| 'stroke-miterlimit' | length | |
| 'stroke-opacity' | number | |
| 'stroke-width' | length | |

| | | |
|---|---|---|
| 'text-anchor' | ident | |
| 'text-decoration' | list of ident | |
| 'text-rendering' | ident | |
| 'unicode-bidi' | ident | |
| 'visibility' | ident | |
| 'word-spacing' | length, ident | |
| 'writing-mode' | ident | |

## B.7 Invalid values

If a script sets a DOM attribute to an invalid value (e.g., a negative number for an attribute that requires a non-negative number or an out-of-range value for an enumeration), unless this specification indicates otherwise, no exception shall be raised on setting, but the given document fragment shall become technically **in error** as described in Error processing.

---

# Appendix C: IDL Definitions

This appendix contains the complete OMG IDL for the SVG Document Object Model definitions. The IDL is also available at: http://www.w3.org/TR/2001/REC-SVG-20010904/idl.zip.

The SVG IDL defines the model for the SVG DOM. Note that the SVG IDL is defined such that some interfaces have more than one base class. The different standard language bindings for the SVG DOM are responsible for defining how to map all aspects of the SVG DOM into the given language, including how the language should implement interfaces with more than one base class.

```
// File: svg.idl
#ifndef _SVG_IDL_
#define _SVG_IDL_


// For access to DOM2 core
#include "dom.idl"

// For access to DOM2 events
#include "events.idl"

// For access to those parts from DOM2 CSS OM used by SVG DOM.
#include "css.idl"

// For access to those parts from DOM2 Views OM used by SVG DOM.
#include "views.idl"

// For access to the SMIL OM used by SVG DOM.
#include "smil.idl"

#pragma prefix "dom.w3c.org"
#pragma javaPackage "org.w3c.dom"
module svg
{
  typedef dom::DOMString DOMString;
  typedef dom::DOMException DOMException;
  typedef dom::Element Element;
  typedef dom::Document Document;
  typedef dom::NodeList NodeList;

  // Predeclarations
  interface SVGElement;
  interface SVGLangSpace;
  interface SVGExternalResourcesRequired;
  interface SVGTests;
  interface SVGFitToViewBox;
  interface SVGZoomAndPan;
  interface SVGViewSpec;
  interface SVGURIReference;
  interface SVGPoint;
  interface SVGMatrix;
  interface SVGPreserveAspectRatio;
  interface SVGAnimatedPreserveAspectRatio;
  interface SVGTransformList;
  interface SVGAnimatedTransformList;
  interface SVGTransform;
  interface SVGICCColor;
  interface SVGColor;
  interface SVGPaint;
  interface SVGTransformable;
  interface SVGDocument;
  interface SVGSVGElement;
  interface SVGElementInstance;
  interface SVGElementInstanceList;


  exception SVGException {
    unsigned short   code;
  };

  // SVGExceptionCode
  const unsigned short SVG_WRONG_TYPE_ERR           = 0;
  const unsigned short SVG_INVALID_VALUE_ERR        = 1;
  const unsigned short SVG_MATRIX_NOT_INVERTABLE    = 2;

  interface SVGElement : Element {
            attribute DOMString id;
                        // raises DOMException on setting
            attribute DOMString xmlbase;
                        // raises DOMException on setting
    readonly attribute SVGSVGElement ownerSVGElement;
    readonly attribute SVGElement viewportElement;
  };

  interface SVGAnimatedBoolean {

            attribute boolean baseVal;
                        // raises DOMException on setting
    readonly attribute boolean animVal;
  };

  interface SVGAnimatedString {

            attribute DOMString baseVal;
                        // raises DOMException on setting
    readonly attribute DOMString animVal;
  };

  interface SVGStringList {

    readonly attribute unsigned long numberOfItems;

    void   clear (  )
```

```
                    raises( DOMException );
  DOMString initialize ( in DOMString newItem )
                    raises( DOMException, SVGException );
  DOMString getItem ( in unsigned long index )
                    raises( DOMException );
  DOMString insertItemBefore ( in DOMString newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  DOMString replaceItem ( in DOMString newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  DOMString removeItem ( in unsigned long index )
                    raises( DOMException );
  DOMString appendItem ( in DOMString newItem )
                    raises( DOMException, SVGException );
};

interface SVGAnimatedEnumeration {

           attribute unsigned short baseVal;
                      // raises DOMException on setting
  readonly attribute unsigned short animVal;
};

interface SVGAnimatedInteger {

           attribute long baseVal;
                      // raises DOMException on setting
  readonly attribute long animVal;
};

interface SVGNumber {

           attribute float value;
                      // raises DOMException on setting
};

interface SVGAnimatedNumber {

           attribute float baseVal;
                      // raises DOMException on setting
  readonly attribute float animVal;
};

interface SVGNumberList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                    raises( DOMException );
  SVGNumber initialize ( in SVGNumber newItem )
                    raises( DOMException, SVGException );
  SVGNumber getItem ( in unsigned long index )
                    raises( DOMException );
  SVGNumber insertItemBefore ( in SVGNumber newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  SVGNumber replaceItem ( in SVGNumber newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  SVGNumber removeItem ( in unsigned long index )
                    raises( DOMException );
  SVGNumber appendItem ( in SVGNumber newItem )
                    raises( DOMException, SVGException );
};

interface SVGAnimatedNumberList {

  readonly attribute SVGNumberList baseVal;
  readonly attribute SVGNumberList animVal;
};

interface SVGLength {

  // Length Unit Types
  const unsigned short SVG_LENGTHTYPE_UNKNOWN    = 0;
  const unsigned short SVG_LENGTHTYPE_NUMBER     = 1;
  const unsigned short SVG_LENGTHTYPE_PERCENTAGE = 2;
  const unsigned short SVG_LENGTHTYPE_EMS        = 3;
  const unsigned short SVG_LENGTHTYPE_EXS        = 4;
  const unsigned short SVG_LENGTHTYPE_PX         = 5;
  const unsigned short SVG_LENGTHTYPE_CM         = 6;
  const unsigned short SVG_LENGTHTYPE_MM         = 7;
  const unsigned short SVG_LENGTHTYPE_IN         = 8;
  const unsigned short SVG_LENGTHTYPE_PT         = 9;
  const unsigned short SVG_LENGTHTYPE_PC         = 10;

  readonly attribute unsigned short unitType;
           attribute float          value;
                      // raises DOMException on setting
           attribute float          valueInSpecifiedUnits;
                      // raises DOMException on setting
           attribute DOMString      valueAsString;
                      // raises DOMException on setting

  void newValueSpecifiedUnits ( in unsigned short unitType, in float valueInSpecifiedUnits );
  void convertToSpecifiedUnits ( in unsigned short unitType );
};

interface SVGAnimatedLength {

  readonly attribute SVGLength baseVal;
  readonly attribute SVGLength animVal;
};

interface SVGLengthList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                    raises( DOMException );
  SVGLength initialize ( in SVGLength newItem )
                    raises( DOMException, SVGException );
  SVGLength getItem ( in unsigned long index )
                    raises( DOMException );
  SVGLength insertItemBefore ( in SVGLength newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  SVGLength replaceItem ( in SVGLength newItem, in unsigned long index )
                    raises( DOMException, SVGException );
  SVGLength removeItem ( in unsigned long index )
                    raises( DOMException );
  SVGLength appendItem ( in SVGLength newItem )
                    raises( DOMException, SVGException );
};

interface SVGAnimatedLengthList {
```

```
  readonly attribute SVGLengthList baseVal;
  readonly attribute SVGLengthList animVal;
};

interface SVGAngle {

  // Angle Unit Types
  const unsigned short SVG_ANGLETYPE_UNKNOWN     = 0;
  const unsigned short SVG_ANGLETYPE_UNSPECIFIED = 1;
  const unsigned short SVG_ANGLETYPE_DEG         = 2;
  const unsigned short SVG_ANGLETYPE_RAD         = 3;
  const unsigned short SVG_ANGLETYPE_GRAD        = 4;

  readonly attribute unsigned short unitType;
           attribute float          value;
                     // raises DOMException on setting
           attribute float          valueInSpecifiedUnits;
                     // raises DOMException on setting
           attribute DOMString      valueAsString;
                     // raises DOMException on setting

  void newValueSpecifiedUnits ( in unsigned short unitType, in float valueInSpecifiedUnits );
  void convertToSpecifiedUnits ( in unsigned short unitType );
};

interface SVGAnimatedAngle {

  readonly attribute SVGAngle baseVal;
  readonly attribute SVGAngle animVal;
};

interface SVGColor : css::CSSValue {
  // Color Types
  const unsigned short SVG_COLORTYPE_UNKNOWN           = 0;
  const unsigned short SVG_COLORTYPE_RGBCOLOR          = 1;
  const unsigned short SVG_COLORTYPE_RGBCOLOR_ICCCOLOR = 2;
  const unsigned short SVG_COLORTYPE_CURRENTCOLOR      = 3;

  readonly attribute unsigned short colorType;
  readonly attribute css::RGBColor  rgbColor;
  readonly attribute SVGICCColor    iccColor;

  void        setRGBColor ( in DOMString rgbColor )
                  raises( SVGException );
  void        setRGBColorICCColor ( in DOMString rgbColor, in DOMString iccColor )
                  raises( SVGException );
  void        setColor ( in unsigned short colorType, in DOMString rgbColor, in DOMString iccColor )
                  raises( SVGException );
};

interface SVGICCColor {

           attribute DOMString      colorProfile;
                     // raises DOMException on setting
  readonly attribute SVGNumberList colors;
};

interface SVGRect {

           attribute float x;
                     // raises DOMException on setting
           attribute float y;
                     // raises DOMException on setting
           attribute float width;
                     // raises DOMException on setting
           attribute float height;
                     // raises DOMException on setting
};

interface SVGAnimatedRect {

  readonly attribute SVGRect baseVal;
  readonly attribute SVGRect animVal;
};

interface SVGUnitTypes {

  // Unit Types
  const unsigned short SVG_UNIT_TYPE_UNKNOWN           = 0;
  const unsigned short SVG_UNIT_TYPE_USERSPACEONUSE    = 1;
  const unsigned short SVG_UNIT_TYPE_OBJECTBOUNDINGBOX = 2;
};

interface SVGStylable {

  readonly attribute SVGAnimatedString className;
  readonly attribute css::CSSStyleDeclaration style;

  css::CSSValue getPresentationAttribute ( in DOMString name );
};

interface SVGLocatable {

  readonly attribute SVGElement             nearestViewportElement;
  readonly attribute SVGElement             farthestViewportElement;

  SVGRect   getBBox (  );
  SVGMatrix getCTM (  );
  SVGMatrix getScreenCTM (  );
  SVGMatrix getTransformToElement ( in SVGElement element )
                raises( SVGException );
};

interface SVGTransformable : SVGLocatable {
  readonly attribute SVGAnimatedTransformList transform;
};

interface SVGTests {

  readonly attribute SVGStringList requiredFeatures;
  readonly attribute SVGStringList requiredExtensions;
  readonly attribute SVGStringList systemLanguage;

  boolean hasExtension ( in DOMString extension );
};

interface SVGLangSpace {

           attribute DOMString xmllang;
                     // raises DOMException on setting
           attribute DOMString xmlspace;
                     // raises DOMException on setting
};
```

```
interface SVGExternalResourcesRequired {

  readonly attribute SVGAnimatedBoolean externalResourcesRequired;
};

interface SVGFitToViewBox {

  readonly attribute SVGAnimatedRect               viewBox;
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};

interface SVGZoomAndPan {

  // Zoom and Pan Types
  const unsigned short SVG_ZOOMANDPAN_UNKNOWN   = 0;
  const unsigned short SVG_ZOOMANDPAN_DISABLE = 1;
  const unsigned short SVG_ZOOMANDPAN_MAGNIFY = 2;

           attribute unsigned short zoomAndPan;
                     // raises DOMException on setting
};

interface SVGViewSpec :
              SVGZoomAndPan,
              SVGFitToViewBox {

  readonly attribute SVGTransformList transform;
  readonly attribute SVGElement       viewTarget;
  readonly attribute DOMString        viewBoxString;
  readonly attribute DOMString        preserveAspectRatioString;
  readonly attribute DOMString        transformString;
  readonly attribute DOMString        viewTargetString;
};

interface SVGURIReference {

  readonly attribute SVGAnimatedString href;
};

interface SVGCSSRule : css::CSSRule {
  // Additional CSS RuleType to support ICC color specifications
  const unsigned short COLOR_PROFILE_RULE = 7;
};

interface SVGRenderingIntent {

  // Rendering Intent Types
  const unsigned short RENDERING_INTENT_UNKNOWN               = 0;
  const unsigned short RENDERING_INTENT_AUTO                  = 1;
  const unsigned short RENDERING_INTENT_PERCEPTUAL            = 2;
  const unsigned short RENDERING_INTENT_RELATIVE_COLORIMETRIC = 3;
  const unsigned short RENDERING_INTENT_SATURATION            = 4;
  const unsigned short RENDERING_INTENT_ABSOLUTE_COLORIMETRIC = 5;
};

interface SVGDocument :
              Document,
              events::DocumentEvent {

  readonly attribute DOMString    title;
  readonly attribute DOMString     referrer;
  readonly attribute DOMString      domain;
  readonly attribute DOMString      URL;
  readonly attribute SVGSVGElement rootElement;
};

interface SVGSVGElement :
              SVGElement,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGLocatable,
              SVGFitToViewBox,
              SVGZoomAndPan,
              events::EventTarget,
              events::DocumentEvent,
              css::ViewCSS,
              css::DocumentCSS {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
           attribute DOMString         contentScriptType;
                     // raises DOMException on setting
           attribute DOMString         contentStyleType;
                     // raises DOMException on setting
  readonly attribute SVGRect           viewport;
  readonly attribute float pixelUnitToMillimeterX;
  readonly attribute float pixelUnitToMillimeterY;
  readonly attribute float screenPixelToMillimeterX;
  readonly attribute float screenPixelToMillimeterY;
           attribute boolean useCurrentView;
                     // raises DOMException on setting
  readonly attribute SVGViewSpec currentView;
           attribute float currentScale;
                     // raises DOMException on setting
  readonly attribute SVGPoint currentTranslate;

  unsigned long suspendRedraw ( in unsigned long max_wait_milliseconds );
  void          unsuspendRedraw ( in unsigned long suspend_handle_id )
                  raises( DOMException );
  void          unsuspendRedrawAll (  );
  void          forceRedraw (  );
  void          pauseAnimations (  );
  void          unpauseAnimations (  );
  boolean       animationsPaused (  );
  float         getCurrentTime (  );
  void          setCurrentTime ( in float seconds );
  NodeList      getIntersectionList ( in SVGRect rect, in SVGElement referenceElement );
  NodeList      getEnclosureList ( in SVGRect rect, in SVGElement referenceElement );
  boolean       checkIntersection ( in SVGElement element, in SVGRect rect );
  boolean       checkEnclosure ( in SVGElement element, in SVGRect rect );
  void          deselectAll (  );
  SVGNumber              createSVGNumber (  );
  SVGLength              createSVGLength (  );
  SVGAngle               createSVGAngle (  );
  SVGPoint               createSVGPoint (  );
  SVGMatrix              createSVGMatrix (  );
  SVGRect                createSVGRect (  );
```

```
  SVGTransform           createSVGTransform (  );
  SVGTransform     createSVGTransformFromMatrix ( in SVGMatrix matrix );
  Element         getElementById ( in DOMString elementId );
};

interface SVGGElement :
              SVGElement,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              events::EventTarget {};

interface SVGDefsElement :
              SVGElement,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              events::EventTarget {};

interface SVGDescElement :
              SVGElement,
              SVGLangSpace,
              SVGStylable {};

interface SVGTitleElement :
              SVGElement,
              SVGLangSpace,
              SVGStylable {};

interface SVGSymbolElement :
              SVGElement,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGFitToViewBox,
              events::EventTarget {};

interface SVGUseElement :
              SVGElement,
              SVGURIReference,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              events::EventTarget {

  readonly attribute SVGAnimatedLength   x;
  readonly attribute SVGAnimatedLength   y;
  readonly attribute SVGAnimatedLength   width;
  readonly attribute SVGAnimatedLength   height;
  readonly attribute SVGElementInstance instanceRoot;
  readonly attribute SVGElementInstance animatedInstanceRoot;
};

interface SVGElementInstance : events::EventTarget {
  readonly attribute SVGElement correspondingElement;
  readonly attribute SVGUseElement correspondingUseElement;
  readonly attribute SVGElementInstance parentNode;
  readonly attribute SVGElementInstanceList childNodes;
  readonly attribute SVGElementInstance firstChild;
  readonly attribute SVGElementInstance lastChild;
  readonly attribute SVGElementInstance previousSibling;
  readonly attribute SVGElementInstance nextSibling;
};

interface SVGElementInstanceList {

  readonly attribute unsigned long length;

  SVGElementInstance item ( in unsigned long index );
};

interface SVGImageElement :
              SVGElement,
              SVGURIReference,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              events::EventTarget {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};

interface SVGSwitchElement :
              SVGElement,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              events::EventTarget {};

interface GetSVGDocument {

  SVGDocument getSVGDocument (  )
                  raises( DOMException );
};

interface SVGStyleElement : SVGElement {
           attribute DOMString xmlspace;
                       // raises DOMException on setting
           attribute DOMString type;
                       // raises DOMException on setting
           attribute DOMString media;
                       // raises DOMException on setting
           attribute DOMString title;
                       // raises DOMException on setting
};

interface SVGPoint {
```

```
            attribute float x;
                        // raises DOMException on setting
            attribute float y;
                        // raises DOMException on setting

  SVGPoint matrixTransform ( in SVGMatrix matrix );
};

interface SVGPointList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                raises( DOMException );
  SVGPoint initialize ( in SVGPoint newItem )
                raises( DOMException, SVGException );
  SVGPoint getItem ( in unsigned long index )
                raises( DOMException );
  SVGPoint insertItemBefore ( in SVGPoint newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGPoint replaceItem ( in SVGPoint newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGPoint removeItem ( in unsigned long index )
                raises( DOMException );
  SVGPoint appendItem ( in SVGPoint newItem )
                raises( DOMException, SVGException );
};

interface SVGMatrix {

            attribute float a;
                        // raises DOMException on setting
            attribute float b;
                        // raises DOMException on setting
            attribute float c;
                        // raises DOMException on setting
            attribute float d;
                        // raises DOMException on setting
            attribute float e;
                        // raises DOMException on setting
            attribute float f;
                        // raises DOMException on setting

  SVGMatrix multiply ( in SVGMatrix secondMatrix );
  SVGMatrix inverse (  )
                raises( SVGException );
  SVGMatrix translate ( in float x, in float y );
  SVGMatrix scale ( in float scaleFactor );
  SVGMatrix scaleNonUniform ( in float scaleFactorX, in float scaleFactorY );
  SVGMatrix rotate ( in float angle );
  SVGMatrix rotateFromVector ( in float x, in float y )
                raises( SVGException );
  SVGMatrix flipX (  );
  SVGMatrix flipY (  );
  SVGMatrix skewX ( in float angle );
  SVGMatrix skewY ( in float angle );
};

interface SVGTransform {

  // Transform Types
  const unsigned short SVG_TRANSFORM_UNKNOWN   = 0;
  const unsigned short SVG_TRANSFORM_MATRIX    = 1;
  const unsigned short SVG_TRANSFORM_TRANSLATE = 2;
  const unsigned short SVG_TRANSFORM_SCALE     = 3;
  const unsigned short SVG_TRANSFORM_ROTATE    = 4;
  const unsigned short SVG_TRANSFORM_SKEWX     = 5;
  const unsigned short SVG_TRANSFORM_SKEWY     = 6;

  readonly attribute unsigned short type;
  readonly attribute SVGMatrix matrix;
  readonly attribute float angle;

  void setMatrix ( in SVGMatrix matrix );
  void setTranslate ( in float tx, in float ty );
  void setScale ( in float sx, in float sy );
  void setRotate ( in float angle, in float cx, in float cy );
  void setSkewX ( in float angle );
  void setSkewY ( in float angle );
};

interface SVGTransformList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                raises( DOMException );
  SVGTransform initialize ( in SVGTransform newItem )
                raises( DOMException, SVGException );
  SVGTransform getItem ( in unsigned long index )
                raises( DOMException );
  SVGTransform insertItemBefore ( in SVGTransform newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGTransform replaceItem ( in SVGTransform newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGTransform removeItem ( in unsigned long index )
                raises( DOMException );
  SVGTransform appendItem ( in SVGTransform newItem )
                raises( DOMException, SVGException );
  SVGTransform createSVGTransformFromMatrix ( in SVGMatrix matrix );
  SVGTransform consolidate (  );
};

interface SVGAnimatedTransformList {

  readonly attribute SVGTransformList baseVal;
  readonly attribute SVGTransformList animVal;
};

interface SVGPreserveAspectRatio {

  // Alignment Types
  const unsigned short SVG_PRESERVEASPECTRATIO_UNKNOWN   = 0;
  const unsigned short SVG_PRESERVEASPECTRATIO_NONE     = 1;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMIN = 2;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMIN = 3;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMIN = 4;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMID = 5;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMID = 6;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMID = 7;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMINYMAX = 8;
  const unsigned short SVG_PRESERVEASPECTRATIO_XMIDYMAX = 9;
```

```
  const unsigned short SVG_PRESERVEASPECTRATIO_XMAXYMAX = 10;
  // Meet-or-slice Types
  const unsigned short SVG_MEETORSLICE_UNKNOWN   = 0;
  const unsigned short SVG_MEETORSLICE_MEET  = 1;
  const unsigned short SVG_MEETORSLICE_SLICE = 2;

           attribute unsigned short align;
                       // raises DOMException on setting
           attribute unsigned short meetOrSlice;
                       // raises DOMException on setting
};

interface SVGAnimatedPreserveAspectRatio {

  readonly attribute SVGPreserveAspectRatio baseVal;
  readonly attribute SVGPreserveAspectRatio animVal;
};

interface SVGPathSeg {

  // Path Segment Types
  const unsigned short PATHSEG_UNKNOWN                    = 0;
  const unsigned short PATHSEG_CLOSEPATH                  = 1;
  const unsigned short PATHSEG_MOVETO_ABS                 = 2;
  const unsigned short PATHSEG_MOVETO_REL                 = 3;
  const unsigned short PATHSEG_LINETO_ABS                 = 4;
  const unsigned short PATHSEG_LINETO_REL                 = 5;
  const unsigned short PATHSEG_CURVETO_CUBIC_ABS          = 6;
  const unsigned short PATHSEG_CURVETO_CUBIC_REL          = 7;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_ABS      = 8;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_REL      = 9;
  const unsigned short PATHSEG_ARC_ABS                    = 10;
  const unsigned short PATHSEG_ARC_REL                    = 11;
  const unsigned short PATHSEG_LINETO_HORIZONTAL_ABS      = 12;
  const unsigned short PATHSEG_LINETO_HORIZONTAL_REL      = 13;
  const unsigned short PATHSEG_LINETO_VERTICAL_ABS        = 14;
  const unsigned short PATHSEG_LINETO_VERTICAL_REL        = 15;
  const unsigned short PATHSEG_CURVETO_CUBIC_SMOOTH_ABS   = 16;
  const unsigned short PATHSEG_CURVETO_CUBIC_SMOOTH_REL   = 17;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS = 18;
  const unsigned short PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL = 19;

  readonly attribute unsigned short pathSegType;
  readonly attribute DOMString      pathSegTypeAsLetter;
};

interface SVGPathSegClosePath : SVGPathSeg {};

interface SVGPathSegMovetoAbs : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
};

interface SVGPathSegMovetoRel : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
};

interface SVGPathSegLinetoAbs : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
};

interface SVGPathSegLinetoRel : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
};

interface SVGPathSegCurvetoCubicAbs : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
           attribute float   x1;
                       // raises DOMException on setting
           attribute float   y1;
                       // raises DOMException on setting
           attribute float   x2;
                       // raises DOMException on setting
           attribute float   y2;
                       // raises DOMException on setting
};

interface SVGPathSegCurvetoCubicRel : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
           attribute float   x1;
                       // raises DOMException on setting
           attribute float   y1;
                       // raises DOMException on setting
           attribute float   x2;
                       // raises DOMException on setting
           attribute float   y2;
                       // raises DOMException on setting
};

interface SVGPathSegCurvetoQuadraticAbs : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
                       // raises DOMException on setting
           attribute float   x1;
                       // raises DOMException on setting
           attribute float   y1;
                       // raises DOMException on setting
};

interface SVGPathSegCurvetoQuadraticRel : SVGPathSeg {
           attribute float   x;
                       // raises DOMException on setting
           attribute float   y;
```

```
                        // raises DOMException on setting
           attribute float   x1;
                        // raises DOMException on setting
           attribute float   y1;
                        // raises DOMException on setting
};

interface SVGPathSegArcAbs : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
           attribute float   r1;
                        // raises DOMException on setting
           attribute float   r2;
                        // raises DOMException on setting
           attribute float   angle;
                        // raises DOMException on setting
           attribute boolean largeArcFlag;
                        // raises DOMException on setting
           attribute boolean sweepFlag;
                        // raises DOMException on setting
};

interface SVGPathSegArcRel : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
           attribute float   r1;
                        // raises DOMException on setting
           attribute float   r2;
                        // raises DOMException on setting
           attribute float   angle;
                        // raises DOMException on setting
           attribute boolean largeArcFlag;
                        // raises DOMException on setting
           attribute boolean sweepFlag;
                        // raises DOMException on setting
};

interface SVGPathSegLinetoHorizontalAbs : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
};

interface SVGPathSegLinetoHorizontalRel : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
};

interface SVGPathSegLinetoVerticalAbs : SVGPathSeg {
           attribute float   y;
                        // raises DOMException on setting
};

interface SVGPathSegLinetoVerticalRel : SVGPathSeg {
           attribute float   y;
                        // raises DOMException on setting
};

interface SVGPathSegCurvetoCubicSmoothAbs : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
           attribute float   x2;
                        // raises DOMException on setting
           attribute float   y2;
                        // raises DOMException on setting
};

interface SVGPathSegCurvetoCubicSmoothRel : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
           attribute float   x2;
                        // raises DOMException on setting
           attribute float   y2;
                        // raises DOMException on setting
};

interface SVGPathSegCurvetoQuadraticSmoothAbs : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
};

interface SVGPathSegCurvetoQuadraticSmoothRel : SVGPathSeg {
           attribute float   x;
                        // raises DOMException on setting
           attribute float   y;
                        // raises DOMException on setting
};

interface SVGPathSegList {

  readonly attribute unsigned long numberOfItems;

  void   clear (  )
                raises( DOMException );
  SVGPathSeg initialize ( in SVGPathSeg newItem )
                raises( DOMException, SVGException );
  SVGPathSeg getItem ( in unsigned long index )
                raises( DOMException );
  SVGPathSeg insertItemBefore ( in SVGPathSeg newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGPathSeg replaceItem ( in SVGPathSeg newItem, in unsigned long index )
                raises( DOMException, SVGException );
  SVGPathSeg removeItem ( in unsigned long index )
                raises( DOMException );
  SVGPathSeg appendItem ( in SVGPathSeg newItem )
                raises( DOMException, SVGException );
};

interface SVGAnimatedPathData {

  readonly attribute SVGPathSegList   pathSegList;
  readonly attribute SVGPathSegList   normalizedPathSegList;
```

```
  readonly attribute SVGPathSegList   animatedPathSegList;
  readonly attribute SVGPathSegList   animatedNormalizedPathSegList;
};

interface SVGPathElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget,
             SVGAnimatedPathData {

  readonly attribute SVGAnimatedNumber pathLength;

  float         getTotalLength (  );
  SVGPoint      getPointAtLength ( in float distance );
  unsigned long getPathSegAtLength ( in float distance );
  SVGPathSegClosePath    createSVGPathSegClosePath (  );
  SVGPathSegMovetoAbs    createSVGPathSegMovetoAbs ( in float x, in float y );
  SVGPathSegMovetoRel    createSVGPathSegMovetoRel ( in float x, in float y );
  SVGPathSegLinetoAbs    createSVGPathSegLinetoAbs ( in float x, in float y );
  SVGPathSegLinetoRel    createSVGPathSegLinetoRel ( in float x, in float y );
  SVGPathSegCurvetoCubicAbs    createSVGPathSegCurvetoCubicAbs ( in float x, in float y, in float x1, in float y1, in float x2, in float y2 );
  SVGPathSegCurvetoCubicRel    createSVGPathSegCurvetoCubicRel ( in float x, in float y, in float x1, in float y1, in float x2, in float y2 );
  SVGPathSegCurvetoQuadraticAbs    createSVGPathSegCurvetoQuadraticAbs ( in float x, in float y, in float x1, in float y1 );
  SVGPathSegCurvetoQuadraticRel    createSVGPathSegCurvetoQuadraticRel ( in float x, in float y, in float x1, in float y1 );
  SVGPathSegArcAbs    createSVGPathSegArcAbs ( in float x, in float y, in float r1, in float r2, in float angle, in boolean largeArcFlag, in boolean sweepFlag );
  SVGPathSegArcRel    createSVGPathSegArcRel ( in float x, in float y, in float r1, in float r2, in float angle, in boolean largeArcFlag, in boolean sweepFlag );
  SVGPathSegLinetoHorizontalAbs    createSVGPathSegLinetoHorizontalAbs ( in float x );
  SVGPathSegLinetoHorizontalRel    createSVGPathSegLinetoHorizontalRel ( in float x );
  SVGPathSegLinetoVerticalAbs    createSVGPathSegLinetoVerticalAbs ( in float y );
  SVGPathSegLinetoVerticalRel    createSVGPathSegLinetoVerticalRel ( in float y );
  SVGPathSegCurvetoCubicSmoothAbs    createSVGPathSegCurvetoCubicSmoothAbs ( in float x, in float y, in float x2, in float y2 );
  SVGPathSegCurvetoCubicSmoothRel    createSVGPathSegCurvetoCubicSmoothRel ( in float x, in float y, in float x2, in float y2 );
  SVGPathSegCurvetoQuadraticSmoothAbs    createSVGPathSegCurvetoQuadraticSmoothAbs ( in float x, in float y );
  SVGPathSegCurvetoQuadraticSmoothRel    createSVGPathSegCurvetoQuadraticSmoothRel ( in float x, in float y );
};

interface SVGRectElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};

interface SVGCircleElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget {

  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
};

interface SVGEllipseElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget {

  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};

interface SVGLineElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget {

  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};

interface SVGAnimatedPoints {

  readonly attribute SVGPointList   points;
  readonly attribute SVGPointList   animatedPoints;
};

interface SVGPolylineElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget,
             SVGAnimatedPoints {};

interface SVGPolygonElement :
             SVGElement,
             SVGTests,
             SVGLangSpace,
```

```
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget,
                SVGAnimatedPoints {};

interface SVGTextContentElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                events::EventTarget {

  // lengthAdjust Types
  const unsigned short LENGTHADJUST_UNKNOWN   = 0;
  const unsigned short LENGTHADJUST_SPACING     = 1;
  const unsigned short LENGTHADJUST_SPACINGANDGLYPHS     = 2;

  readonly attribute SVGAnimatedLength      textLength;
  readonly attribute SVGAnimatedEnumeration lengthAdjust;

  long     getNumberOfChars (  );
  float    getComputedTextLength (  );
  float    getSubStringLength ( in unsigned long charnum, in unsigned long nchars )
                  raises( DOMException );
  SVGPoint getStartPositionOfChar ( in unsigned long charnum )
                  raises( DOMException );
  SVGPoint getEndPositionOfChar ( in unsigned long charnum )
                  raises( DOMException );
  SVGRect  getExtentOfChar ( in unsigned long charnum )
                  raises( DOMException );
  float    getRotationOfChar ( in unsigned long charnum )
                  raises( DOMException );
  long     getCharNumAtPosition ( in SVGPoint point );
  void     selectSubString ( in unsigned long charnum, in unsigned long nchars )
                  raises( DOMException );
};

interface SVGTextPositioningElement : SVGTextContentElement {
  readonly attribute SVGAnimatedLengthList x;
  readonly attribute SVGAnimatedLengthList y;
  readonly attribute SVGAnimatedLengthList dx;
  readonly attribute SVGAnimatedLengthList dy;
  readonly attribute SVGAnimatedNumberList rotate;
};

interface SVGTextElement :
                SVGTextPositioningElement,
                SVGTransformable {};

interface SVGTSpanElement : SVGTextPositioningElement {};

interface SVGTRefElement :
                SVGTextPositioningElement,
                SVGURIReference {};

interface SVGTextPathElement :
                SVGTextContentElement,
                SVGURIReference {

  // textPath Method Types
  const unsigned short TEXTPATH_METHODTYPE_UNKNOWN   = 0;
  const unsigned short TEXTPATH_METHODTYPE_ALIGN     = 1;
  const unsigned short TEXTPATH_METHODTYPE_STRETCH     = 2;
  // textPath Spacing Types
  const unsigned short TEXTPATH_SPACINGTYPE_UNKNOWN   = 0;
  const unsigned short TEXTPATH_SPACINGTYPE_AUTO     = 1;
  const unsigned short TEXTPATH_SPACINGTYPE_EXACT     = 2;

  readonly attribute SVGAnimatedLength              startOffset;
  readonly attribute SVGAnimatedEnumeration method;
  readonly attribute SVGAnimatedEnumeration spacing;
};

interface SVGAltGlyphElement :
                SVGTextPositioningElement,
                SVGURIReference {

            attribute DOMString glyphRef;
                        // raises DOMException on setting
            attribute DOMString format;
                        // raises DOMException on setting
};

interface SVGAltGlyphDefElement : SVGElement {};

interface SVGAltGlyphItemElement : SVGElement {};

interface SVGGlyphRefElement :
                SVGElement,
                SVGURIReference,
                SVGStylable {

            attribute DOMString glyphRef;
                        // raises DOMException on setting
            attribute DOMString format;
                        // raises DOMException on setting
            attribute float    x;
                        // raises DOMException on setting
            attribute float    y;
                        // raises DOMException on setting
            attribute float    dx;
                        // raises DOMException on setting
            attribute float    dy;
                        // raises DOMException on setting
};

interface SVGPaint : SVGColor {
  // Paint Types
  const unsigned short SVG_PAINTTYPE_UNKNOWN               = 0;
  const unsigned short SVG_PAINTTYPE_RGBCOLOR              = 1;
  const unsigned short SVG_PAINTTYPE_RGBCOLOR_ICCCOLOR     = 2;
  const unsigned short SVG_PAINTTYPE_NONE                  = 101;
  const unsigned short SVG_PAINTTYPE_CURRENTCOLOR          = 102;
  const unsigned short SVG_PAINTTYPE_URI_NONE              = 103;
  const unsigned short SVG_PAINTTYPE_URI_CURRENTCOLOR      = 104;
  const unsigned short SVG_PAINTTYPE_URI_RGBCOLOR          = 105;
  const unsigned short SVG_PAINTTYPE_URI_RGBCOLOR_ICCCOLOR = 106;
  const unsigned short SVG_PAINTTYPE_URI                   = 107;

  readonly attribute unsigned short paintType;
```

```
  readonly attribute DOMString      uri;

  void setUri ( in DOMString uri );
  void setPaint ( in unsigned short paintType, in DOMString uri, in DOMString rgbColor, in DOMString iccColor )
                  raises( SVGException );
};

interface SVGMarkerElement :
              SVGElement,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGFitToViewBox {

  // Marker Unit Types
  const unsigned short SVG_MARKERUNITS_UNKNOWN        = 0;
  const unsigned short SVG_MARKERUNITS_USERSPACEONUSE = 1;
  const unsigned short SVG_MARKERUNITS_STROKEWIDTH    = 2;
  // Marker Orientation Types
  const unsigned short SVG_MARKER_ORIENT_UNKNOWN      = 0;
  const unsigned short SVG_MARKER_ORIENT_AUTO         = 1;
  const unsigned short SVG_MARKER_ORIENT_ANGLE        = 2;

  readonly attribute SVGAnimatedLength      refX;
  readonly attribute SVGAnimatedLength      refY;
  readonly attribute SVGAnimatedEnumeration markerUnits;
  readonly attribute SVGAnimatedLength      markerWidth;
  readonly attribute SVGAnimatedLength      markerHeight;
  readonly attribute SVGAnimatedEnumeration orientType;
  readonly attribute SVGAnimatedAngle      orientAngle;

  void setOrientToAuto (  );
  void setOrientToAngle ( in SVGAngle angle );
};

interface SVGColorProfileElement :
              SVGElement,
              SVGURIReference,
              SVGRenderingIntent {

           attribute DOMString      local;
                       // raises DOMException on setting
           attribute DOMString      name;
                       // raises DOMException on setting
           attribute unsigned short renderingIntent;
                       // raises DOMException on setting
};

interface SVGColorProfileRule :
              SVGCSSRule,
              SVGRenderingIntent {

           attribute DOMString      src;
                       // raises DOMException on setting
           attribute DOMString      name;
                       // raises DOMException on setting
           attribute unsigned short renderingIntent;
                       // raises DOMException on setting
};

interface SVGGradientElement :
              SVGElement,
              SVGURIReference,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGUnitTypes {

  // Spread Method Types
  const unsigned short SVG_SPREADMETHOD_UNKNOWN = 0;
  const unsigned short SVG_SPREADMETHOD_PAD     = 1;
  const unsigned short SVG_SPREADMETHOD_REFLECT = 2;
  const unsigned short SVG_SPREADMETHOD_REPEAT  = 3;

  readonly attribute SVGAnimatedEnumeration   gradientUnits;
  readonly attribute SVGAnimatedTransformList gradientTransform;
  readonly attribute SVGAnimatedEnumeration   spreadMethod;
};

interface SVGLinearGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};

interface SVGRadialGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
  readonly attribute SVGAnimatedLength fx;
  readonly attribute SVGAnimatedLength fy;
};

interface SVGStopElement :
              SVGElement,
              SVGStylable {

  readonly attribute SVGAnimatedNumber offset;
};

interface SVGPatternElement :
              SVGElement,
              SVGURIReference,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGFitToViewBox,
              SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration   patternUnits;
  readonly attribute SVGAnimatedEnumeration   patternContentUnits;
  readonly attribute SVGAnimatedTransformList patternTransform;
  readonly attribute SVGAnimatedLength        x;
  readonly attribute SVGAnimatedLength        y;
  readonly attribute SVGAnimatedLength        width;
  readonly attribute SVGAnimatedLength        height;
};

interface SVGClipPathElement :
              SVGElement,
              SVGTests,
```

```
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGTransformable,
              SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration clipPathUnits;
};

interface SVGMaskElement :
              SVGElement,
              SVGTests,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration maskUnits;
  readonly attribute SVGAnimatedEnumeration maskContentUnits;
  readonly attribute SVGAnimatedLength      x;
  readonly attribute SVGAnimatedLength      y;
  readonly attribute SVGAnimatedLength      width;
  readonly attribute SVGAnimatedLength      height;
};

interface SVGFilterElement :
              SVGElement,
              SVGURIReference,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGStylable,
              SVGUnitTypes {

  readonly attribute SVGAnimatedEnumeration filterUnits;
  readonly attribute SVGAnimatedEnumeration primitiveUnits;
  readonly attribute SVGAnimatedLength      x;
  readonly attribute SVGAnimatedLength      y;
  readonly attribute SVGAnimatedLength      width;
  readonly attribute SVGAnimatedLength      height;
  readonly attribute SVGAnimatedInteger    filterResX;
  readonly attribute SVGAnimatedInteger    filterResY;

  void setFilterRes ( in unsigned long filterResX, in unsigned long filterResY );
};

interface SVGFilterPrimitiveStandardAttributes : SVGStylable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};

interface SVGFEBlendElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  // Blend Mode Types
  const unsigned short SVG_FEBLEND_MODE_UNKNOWN  = 0;
  const unsigned short SVG_FEBLEND_MODE_NORMAL   = 1;
  const unsigned short SVG_FEBLEND_MODE_MULTIPLY = 2;
  const unsigned short SVG_FEBLEND_MODE_SCREEN   = 3;
  const unsigned short SVG_FEBLEND_MODE_DARKEN   = 4;
  const unsigned short SVG_FEBLEND_MODE_LIGHTEN  = 5;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration mode;
};

interface SVGFEColorMatrixElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  // Color Matrix Types
  const unsigned short SVG_FECOLORMATRIX_TYPE_UNKNOWN          = 0;
  const unsigned short SVG_FECOLORMATRIX_TYPE_MATRIX           = 1;
  const unsigned short SVG_FECOLORMATRIX_TYPE_SATURATE         = 2;
  const unsigned short SVG_FECOLORMATRIX_TYPE_HUEROTATE        = 3;
  const unsigned short SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA = 4;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  values;
};

interface SVGFEComponentTransferElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
};

interface SVGComponentTransferFunctionElement : SVGElement {
  // Component Transfer Types
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN  = 0;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_IDENTITY = 1;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_TABLE    = 2;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE    = 3;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_LINEAR   = 4;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_GAMMA    = 5;

  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  tableValues;
  readonly attribute SVGAnimatedNumber      slope;
  readonly attribute SVGAnimatedNumber      intercept;
  readonly attribute SVGAnimatedNumber      amplitude;
  readonly attribute SVGAnimatedNumber      exponent;
  readonly attribute SVGAnimatedNumber      offset;
};

interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {};

interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {};

interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {};

interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {};

interface SVGFECompositeElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {
```

```
  // Composite Operators
  const unsigned short SVG_FECOMPOSITE_OPERATOR_UNKNOWN    = 0;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OVER       = 1;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_IN         = 2;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OUT        = 3;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ATOP       = 4;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_XOR        = 5;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ARITHMETIC = 6;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedNumber      k1;
  readonly attribute SVGAnimatedNumber      k2;
  readonly attribute SVGAnimatedNumber      k3;
  readonly attribute SVGAnimatedNumber      k4;
};

interface SVGFEConvolveMatrixElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  // Edge Mode Values
  const unsigned short SVG_EDGEMODE_UNKNOWN   = 0;
  const unsigned short SVG_EDGEMODE_DUPLICATE = 1;
  const unsigned short SVG_EDGEMODE_WRAP      = 2;
  const unsigned short SVG_EDGEMODE_NONE      = 3;

  readonly attribute SVGAnimatedInteger     orderX;
  readonly attribute SVGAnimatedInteger     orderY;
  readonly attribute SVGAnimatedNumberList  kernelMatrix;
  readonly attribute SVGAnimatedNumber      divisor;
  readonly attribute SVGAnimatedNumber      bias;
  readonly attribute SVGAnimatedInteger     targetX;
  readonly attribute SVGAnimatedInteger     targetY;
  readonly attribute SVGAnimatedEnumeration edgeMode;
  readonly attribute SVGAnimatedLength      kernelUnitLengthX;
  readonly attribute SVGAnimatedLength      kernelUnitLengthY;
  readonly attribute SVGAnimatedBoolean     preserveAlpha;
};

interface SVGFEDiffuseLightingElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
};

interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};

interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};

interface SVGFESpotLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
  readonly attribute SVGAnimatedNumber pointsAtX;
  readonly attribute SVGAnimatedNumber pointsAtY;
  readonly attribute SVGAnimatedNumber pointsAtZ;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber limitingConeAngle;
};

interface SVGFEDisplacementMapElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  // Channel Selectors
  const unsigned short SVG_CHANNEL_UNKNOWN = 0;
  const unsigned short SVG_CHANNEL_R       = 1;
  const unsigned short SVG_CHANNEL_G       = 2;
  const unsigned short SVG_CHANNEL_B       = 3;
  const unsigned short SVG_CHANNEL_A       = 4;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedNumber      scale;
  readonly attribute SVGAnimatedEnumeration xChannelSelector;
  readonly attribute SVGAnimatedEnumeration yChannelSelector;
};

interface SVGFEFloodElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString      in1;
};

interface SVGFEGaussianBlurElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;

  void setStdDeviation ( in float stdDeviationX, in float stdDeviationY );
};

interface SVGFEImageElement :
              SVGElement,
              SVGURIReference,
              SVGLangSpace,
              SVGExternalResourcesRequired,
              SVGFilterPrimitiveStandardAttributes {};

interface SVGFEMergeElement :
              SVGElement,
              SVGFilterPrimitiveStandardAttributes {};

interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
```

```
};

interface SVGFEMorphologyElement :
             SVGElement,
             SVGFilterPrimitiveStandardAttributes {

  // Morphology Operators
  const unsigned short SVG_MORPHOLOGY_OPERATOR_UNKNOWN = 0;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_ERODE   = 1;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_DILATE  = 2;

  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedLength      radiusX;
  readonly attribute SVGAnimatedLength      radiusY;
};

interface SVGFEOffsetElement :
             SVGElement,
             SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};

interface SVGFESpecularLightingElement :
             SVGElement,
             SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
};

interface SVGFETileElement :
             SVGElement,
             SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
};

interface SVGFETurbulenceElement :
             SVGElement,
             SVGFilterPrimitiveStandardAttributes {

  // Turbulence Types
  const unsigned short SVG_TURBULENCE_TYPE_UNKNOWN      = 0;
  const unsigned short SVG_TURBULENCE_TYPE_FRACTALNOISE = 1;
  const unsigned short SVG_TURBULENCE_TYPE_TURBULENCE   = 2;
  // Stitch Options
  const unsigned short SVG_STITCHTYPE_UNKNOWN  = 0;
  const unsigned short SVG_STITCHTYPE_STITCH   = 1;
  const unsigned short SVG_STITCHTYPE_NOSTITCH = 2;

  readonly attribute SVGAnimatedNumber      baseFrequencyX;
  readonly attribute SVGAnimatedNumber      baseFrequencyY;
  readonly attribute SVGAnimatedInteger     numOctaves;
  readonly attribute SVGAnimatedNumber      seed;
  readonly attribute SVGAnimatedEnumeration stitchTiles;
  readonly attribute SVGAnimatedEnumeration type;
};

interface SVGCursorElement :
             SVGElement,
             SVGURIReference,
             SVGTests,
             SVGExternalResourcesRequired {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
};

interface SVGAElement :
             SVGElement,
             SVGURIReference,
             SVGTests,
             SVGLangSpace,
             SVGExternalResourcesRequired,
             SVGStylable,
             SVGTransformable,
             events::EventTarget {

  readonly attribute SVGAnimatedString target;
};

interface SVGViewElement :
             SVGElement,
             SVGExternalResourcesRequired,
             SVGFitToViewBox,
             SVGZoomAndPan {

  readonly attribute SVGStringList viewTarget;
};

interface SVGScriptElement :
             SVGElement,
             SVGURIReference,
             SVGExternalResourcesRequired {

           attribute DOMString type;
                     // raises DOMException on setting
};

interface SVGEvent : events::Event {};

interface SVGZoomEvent : events::UIEvent {
  readonly attribute SVGRect zoomRectScreen;
  readonly attribute float previousScale;
  readonly attribute SVGPoint previousTranslate;
  readonly attribute float newScale;
  readonly attribute SVGPoint newTranslate;
};

interface SVGAnimationElement :
             SVGElement,
             SVGTests,
             SVGExternalResourcesRequired,
             smil::ElementTimeControl,
             events::EventTarget {

  readonly attribute SVGElement targetElement;
```

```
    float getStartTime (  );
    float getCurrentTime (  );
    float getSimpleDuration (  )
                    raises( DOMException );
  };

  interface SVGAnimateElement : SVGAnimationElement {};

  interface SVGSetElement : SVGAnimationElement {};

  interface SVGAnimateMotionElement : SVGAnimationElement {};

  interface SVGMPathElement :
                SVGElement,
                SVGURIReference,
                SVGExternalResourcesRequired {};

  interface SVGAnimateColorElement : SVGAnimationElement {};

  interface SVGAnimateTransformElement : SVGAnimationElement {};

  interface SVGFontElement :
                SVGElement,
                SVGExternalResourcesRequired,
                SVGStylable {};

  interface SVGGlyphElement :
                SVGElement,
                SVGStylable {};

  interface SVGMissingGlyphElement :
                SVGElement,
                SVGStylable {};

  interface SVGHKernElement : SVGElement {};

  interface SVGVKernElement : SVGElement {};

  interface SVGFontFaceElement : SVGElement {};

  interface SVGFontFaceSrcElement : SVGElement {};

  interface SVGFontFaceUriElement : SVGElement {};

  interface SVGFontFaceFormatElement : SVGElement {};

  interface SVGFontFaceNameElement : SVGElement {};

  interface SVGDefinitionSrcElement : SVGElement {};

  interface SVGMetadataElement : SVGElement {};

  interface SVGForeignObjectElement :
                SVGElement,
                SVGTests,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGTransformable,
                events::EventTarget {

    readonly attribute SVGAnimatedLength x;
    readonly attribute SVGAnimatedLength y;
    readonly attribute SVGAnimatedLength width;
    readonly attribute SVGAnimatedLength height;
  };

};

#endif // _SVG_IDL_
```

---

---

04 September 2001

# Appendix D: Java Language Binding

The Java binding for the SVG Document Object Model definitions is available at:

http://www.w3.org/TR/2001/REC-SVG-20010904/java-binding.zip

---

---

04 September 2001

# Appendix E: ECMAScript Language Binding

The ECMAScript binding for the SVG Document Object Model definitions is available at:

http://www.w3.org/TR/2001/REC-SVG-20010904/ecmascript-binding.html

---

04 September 2001

# Appendix F: Implementation Requirements

## Contents



**This appendix is normative.**

## F.1 Introduction

The following are notes about implementation requirements corresponding to various features in the SVG language.

## F.2 Error processing

There are various scenarios where an SVG document fragment is technically **in error**:

- When the content does not conform to the XML 1.0 specification [XML10], such as the use of incorrect XML syntax
- When an element or attribute is encountered in the document which is not part of the SVG DTD and which is not properly identified as being part of another namespace (see "Namespaces in XML" [XML-NS])
- When an element has an attribute or property value which is not permissible according to this specification

- Other situations that are described as being *in error* in this specification

A document can go in and out of error over time. For example, document changes from the SVG DOM or from animation can cause a document to become *in error* and a further change can cause the document to become correct again.

The following error processing shall occur when a document is in error:

- The document shall be rendered up to, but not including, the first element which has an error. Exceptions:
  - If a **'path'** element is the first element which has an error and the only errors are in the path data specification, then render the **'path'** up to the point of the path data error. For related information, see **'path'** element implementation notes.
  - If a **'polyline'** or **'polygon'** element is the first element which has an error and the only errors are within the points attribute, then render the **'polyline'** or **'polygon'** up to the segment with the error.

  This approach will provide a visual clue to the user or developer about where the error might be in the document.
- If the document has animations, the animations shall stop at the point at which an error is encountered and the visual presentation of the document shall reflect the animated status of the document at the point the error was encountered.
- A highly perceivable indication of error shall occur. For visual rendering situations, an example of an indication of error would be to render a translucent colored pattern such as a checkerboard on top of the area where the SVG content is rendered.
- If the user agent has access to an error reporting capability such as status bar, it is recommended that the user agent provide whatever additional detail it can to enable the user or developer to quickly find the source of the error. For example, the user agent might provide an error message along with a line number and character number at which the error was encountered.

Because of situations where a block of scripting changes might cause a given SVG document fragment to go into and out of error, error processing shall occur only at times when document presentation (e.g., rendering to the display device) is updated. In particular, error processing shall be disabled whenever redraw has been suspended via DOM calls to suspendRedraw().

## F.3 Version control

The SVG user agent must verify the reference to the PUBLIC identifier in the `<!DOCTYPE>` statement or the namespace reference in the `xmlns` attribute on the **'svg'** element to ensure that the given document (or document fragment) identifies a version of the SVG language which the SVG user agent supports. If the version information is missing or the version information indicates a version of the SVG language which the SVG user agent does not support, then the SVG user agent is not required to render that document or fragment. In particular, it is not required that an SVG user agent attempt to render future versions of the SVG language. If the user environment provides such an option, the user agent should alert or otherwise notify the user that the version of the file is not supported and suggest an alternate processing option (e.g., installing an updated version of the user agent) if such an option exists.

An SVG user agent which supports the SVG Recommendation should alert or otherwise notify the user whenever it encounters an SVG document (or document fragment) whose `<!DOCTYPE>` statement or corresponding `xmlns` attribute corresponds to a working draft version of the SVG specification. All content based on working drafts of this specification should be updated to the SVG Recommendation.

## F.4 Clamping values which are restricted to a particular range

Some numeric attribute and property values have restricted ranges, such as color component values. When out-of-range values are provided, the user agent shall defer any error checking until after presentation time, as composited actions might produce intermediate values which are out-of-range but final values which are within range.

Color values are not in error if they are out-of-range, even if final computations produce an out-of-range color value at presentation time. It is recommended that user agents clamp color values to the nearest color value (possibly determined by simple clipping) which the system can process as late as possible (e.g., presentation time), although it is acceptable for user agents to clamp color values as early as parse time. Thus, implementation dependencies might preclude consistent behavior across different systems when out-of-range color values are used.

Opacity values out-of-range are not in error and should be clamped to the range 0 to 1 at the time which opacity values have to be processed (e.g., at presentation time or when it is necessary to perform intermediate filter effect calculations).

## F.5 'path' element implementation notes

A conforming SVG user agent must implement path rendering as follows:

- Error handling:
  - The general rule for error handling in path data is that the SVG user agent shall render a 'path' element up to (but not including) the path command containing the first error in the path data specification. This will provide a visual clue to the user or developer about where the error might be in the path data specification. This rule will greatly discourage generation of invalid SVG path data.
  - If a path data command contains an incorrect set of parameters, then the given path data command is rendered up to and including the last correctly defined path segment, even if that path segment is a sub-component of a compound path data command, such as a "lineto" with several pairs of coordinates. For example, for the path data string "M 10,10 L 20,20,30", there is an odd number of parameters for the "L" command, which requires an even number of parameters. The user agent is required to draw the line from (10,10) to (20,20) and then perform error reporting since "L 20 20" is the last correctly defined segment of the pat data specification.
  - Wherever possible, all SVG user agents shall report all errors to the user.
- Markers, directionality and zero-length path segments:
  - If markers are specified, then a marker is drawn on every applicable vertex, even if the given vertex is the end point of a zero-length path segment and even if "moveto" commands follow each other.
  - Certain line-capping and line-joining situations and markers require that a path segment have directionality at its start and end points. Zero-length path segments have no directionality. In these cases, the following algorithm is used to establish directionality: to determine the directionality of the start point of a zero-length path segment, go backwards in the path data specification within the current subpath until you find a segment which has directionality at its end point (e.g., a path segment with non-zero length) and use its ending direction; otherwise, temporarily consider the start point to lack directionality. Similarly, to determine the directionality of the end point of a zero-length path segment, go forwards in the path data specification within the current subpath until you find a segment which has directionality at its start point (e.g., a path segment with non-zero length) and use its starting direction; otherwise, temporarily consider the end point to lack directionality. If the

start point has directionality but the end point doesn't, then the end point uses the start point's directionality. If the end point has directionality but the start point doesn't, then the start point uses the end point's directionality. Otherwise, set the directionality for the path segment's start and end points to align with the positive x-axis in user space.
    - If **'stroke-linecap'** is set to **butt** and the given path segment has zero length, do not draw the linecap for that segment; however, do draw the linecap for zero-length path segments when **'stroke-linecap'** is set to either **round** or **square**. (This allows round and square dots to be drawn on the canvas.)
- The S/s commands indicate that the first control point of the given cubic Bézier segment is calculated by reflecting the previous path segments second control point relative to the current point. The exact math is as follows. If the current point is (curx, cury) and the second control point of the previous path segment is (oldx2, oldy2), then the reflected point (i.e., (newx1, newy1), the first control point of the current path segment) is:

```
(newx1, newy1) = (curx - (oldx2 - curx), cury - (oldy2 - cury))
               = (2*curx - oldx2, 2*cury - oldy2)
```

- A non-positive radius value is an error.
- Unrecognized contents within a path data stream (i.e., contents that are not part of the path data grammar) is an error.

## F.6 Elliptical arc implementation notes

### F.6.1 Elliptical arc syntax

An elliptical arc is a particular path command.  As such, it is described by the following parameters in order:

$(x_1, y_1)$ are the absolute coordinates of the current point on the path, obtained from the last two parameters of the previous path command.

$r_X$ and $r_Y$ are the radii of the ellipse (also known as its semi-major and semi-minor axes).

 is the angle from the x-axis of the current coordinate system to the x-axis of the ellipse.

$f_A$ is the large arc flag, and is $0$ if an arc spanning less than or equal to $180$ degrees is chosen, or $1$ if an arc spanning greater than $180$ degrees is chosen.

$f_S$ is the sweep flag, and is $0$ if the line joining center to arc sweeps through decreasing angles, or $1$ if it sweeps through increasing angles.

$(x_2, y_2)$ are the absolute coordinates of the final point of the arc.

This parameterization of elliptical arcs will be referred to as *endpoint parameterization*.  One of the advantages of endpoint parameterization is that it permits a consistent path syntax in which all path

commands end in the coordinates of the new "current point". The following notes give rules and formulae to help implementers deal with endpoint parameterization.

## F.6.2 Out-of-range parameters

Arbitrary numerical values are permitted for all elliptical arc parameters, but where these values are invalid or out-of-range, an implementation must make sense of them as follows:

If the endpoints $(x_1, y_1)$ and $(x_2, y_2)$ are identical, then this is equivalent to omitting the elliptical arc segment entirely.

If $r_X = 0$ or $r_Y = 0$ then this arc is treated as a straight line segment (a "lineto") joining the endpoints.

If $r_X$ or $r_Y$ have negative signs, these are dropped; the absolute value is used instead.

If $r_X$ , $r_Y$ and $\varphi$ are such that there is no solution (basically, the ellipse is not big enough to reach from $(x_1, y_1)$ to $(x_2, y_2)$) then the ellipse is scaled up uniformly until there is exactly one solution (until the ellipse is just big enough).

$\varphi$ is taken mod $360$ degrees.

Any nonzero value for either of the flags $f_A$or $f_S$ is taken to mean the value $1$.

This forgiving yet consistent treatment of out-of-range values ensures that:

- The inevitable approximations arising from computer arithmetic cannot cause a valid set of values written by one SVG implementation to be treated as invalid when read by another SVG implementation. This would otherwise be a problem for common boundary cases such as a semicircular arc.
- Continuous animations that cause parameters to pass through invalid values are not a problem. The motion remains continuous.

## F.6.3 Parameterization alternatives

An arbitrary point $(x, y)$ on the elliptical arc can be described by the 2-dimensional matrix equation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos\theta \\ r_Y \sin\theta \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \quad \text{(F.6.3.1)}$$

$(c_X, c_Y)$ are the coordinates of the center of the ellipse.

$r_X$ and $r_Y$ are the radii of the ellipse (also known as its semi-major and semi-minor axes).

$\varphi$ is the angle from the x-axis of the current coordinate system to the x-axis of the ellipse.

$\theta$ ranges from:

$\theta_1$ which is the start angle of the elliptical arc prior to the stretch and rotate operations.

$\theta_2$ which is the end angle of the elliptical arc prior to the stretch and rotate operations.

$\Delta\theta$ which is the difference between these two angles.

If one thinks of an ellipse as a circle that has been stretched and then rotated, then

$\theta_1$, $\theta_1$ and $\Delta\theta$

are the start angle, end angle and sweep angle, respectively of the arc prior to the stretch and rotate operations. This leads to an alternate parameterization which is common among graphics APIs, which will be referred to as *center parameterization*. In the next sections, formulas are given for mapping in both directions between center parameterization and endpoint parameterization.

## F.6.4 Conversion from center to endpoint parameterization

Given:

$$c_X \quad c_Y \quad r_X \quad r_Y \quad \varphi \quad \theta_1 \quad \Delta\theta$$

the task is to find:

$$x_1 \quad y_1 \quad x_2 \quad y_2 \quad f_A \quad f_S$$

Here are the formulas:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos\theta_1 \\ r_Y \sin\theta_1 \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \quad \text{(F.6.4.1)}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos(\theta_1 + \Delta\theta) \\ r_Y \sin(\theta_1 + \Delta\theta) \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \quad \text{(F.6.4.2)}$$

$$f_A = \begin{cases} 1 & \text{if } |\Delta\theta| > 180° \\ 0 & \text{if } |\Delta\theta| \le 180° \end{cases} \quad \text{(F.6.4.3)}$$

$$f_S = \begin{cases} 1 & \text{if } \Delta\theta > 0° \\ 0 & \text{if } \Delta\theta < 0° \end{cases} \quad \text{(F.6.4.4)}$$

## F.6.5 Conversion from endpoint to center parameterization

Given:

$$x_1 \quad y_1 \quad x_2 \quad y_2 \quad f_A \quad f_S$$

the task is to find:

$$c_X \quad c_Y \quad r_X \quad r_Y \quad \varphi \quad \theta_1 \quad \Delta\theta$$

The equations simplify after a translation which places the origin at the midpoint of the line joining $(x_1, y_1)$ to $(x_2, y_2)$, followed by a rotation to line up the coordinate axes with the axes of the ellipse. All transformed coordinates will be written with primes. They are computed as intermediate values on the way toward finding the required center parameterization variables. This procedure consists of the following steps:

*Step 1: Compute* $(x_1', y_1')$ *according to the formula*

$$\begin{pmatrix} x_1' \\ y_1' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} \frac{x_1 - x_2}{2} \\ \frac{y_1 - y_2}{2} \end{pmatrix} \quad \text{(F.6.5.1)}$$

*Step 2: Compute* $(c_X', c_Y')$ *according to the formula*

$$\begin{pmatrix} c_X' \\ c_Y' \end{pmatrix} = \pm \sqrt{\frac{r_X^2 r_Y^2 - r_X^2 y_1'^2 - r_Y^2 x_1'^2}{r_X^2 y_1'^2 + r_Y^2 x_1'^2}} \begin{pmatrix} \frac{r_X y_1'}{r_Y} \\ -\frac{r_Y x_1'}{r_X} \end{pmatrix} \qquad \text{(F.6.5.2)}$$

where the + sign is chosen if $f_A \neq f_S$

and the - sign is chosen if $f_A = f_S$

*Step 3: Compute* ($c_X$, $c_Y$) *from* ($c_X'$, $c_Y'$)

$$\begin{pmatrix} c_X \\ c_Y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} c_X' \\ c_Y' \end{pmatrix} + \begin{pmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{pmatrix} \qquad \text{(F.6.5.3)}$$

*Step 4: Compute* $\theta_1$ and $\Delta\theta$

In general, the angle between two vectors ($u_X$, $u_Y$) and ($v_X$, $v_Y$) can be computed as

$$\angle(\vec{u}, \vec{v}) = \pm \arccos \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \qquad \text{(F.6.5.4)}$$

where the ± sign appearing here is the sign of $u_X v_Y - u_Y v_X$

This angle function can be used to express $\theta_1$ and $\Delta\theta$ as follows:

$$\theta_1 = \angle\left( \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{x_1' - c_X'}{r_X} \\ \frac{y_1' - c_Y'}{r_Y} \end{pmatrix} \right) \qquad \text{(F.6.5.5)}$$

$$\Delta\theta \equiv \angle\left(\left(\frac{x_1{}' - c_X{}'}{r_X}, \frac{y_1{}' - c_Y{}'}{r_Y}\right), \left(\frac{-x_1{}' - c_X{}'}{r_X}, \frac{-y_1{}' - c_Y{}'}{r_Y}\right)\right) \bmod 360° \qquad \text{(F.6.5.6)}$$

where $\theta_1$ is fixed in the range -360° < $\Delta\theta$ < 360° such that:

if $f_S = 0$, then $\Delta\theta < 0$,

else if $f_S = 1$, then $\Delta\theta > 0$.

In other words, if $\boldsymbol{f}_S = 0$ and the right side of (F.6.5.6) is > 0, then subtract 360°, whereas if $\boldsymbol{f}_S = 1$ and the right side of (F.6.5.6) is < 0, then add 360°. In all other cases leave it as is.

## F.6.6 Correction of out-of-range radii

This section formalizes the adjustments to out-of-range $r_X$ and $r_Y$ mentioned in F.6.2. Algorithmically these adjustments consist of the following steps:

*Step 1: Ensure radii are non-zero*

If $r_X = 0$ or $r_Y = 0$, then treat this as a straight line from $(x_1, y_1)$ to $(x_2, y_2)$ and stop. Otherwise,

*Step 2: Ensure radii are positive*

Take the absolute value of $r_X$ and $r_Y$:

$$r_X \rightarrow |r_X| \qquad r_Y \rightarrow |r_Y| \qquad \text{(F.6.6.1)}$$

*Step 3: Ensure radii are large enough*

Using the primed coordinate values of equation (F.6.5.1), compute

$$\Lambda = \frac{x_1{}'^2}{r_X{}^2} + \frac{y_1{}'^2}{r_Y{}^2} \qquad \text{(F.6.6.2)}$$

If the result of the above equation is less than or equal to 1, then no further change need be made to $r_X$ and $r_Y$. If the result of the above equation is greater than 1, then make the replacements

$$r_X \rightarrow \sqrt{\Lambda}\, r_X \qquad r_Y \rightarrow \sqrt{\Lambda}\, r_Y \qquad \text{(F.6.6.3)}$$

*Step 4: Proceed with computations*

Proceed with the remaining elliptical arc computations, such as those in section F.6.5. Note: As a consequence of the radii corrections in this section, equation (F.6.5.2) for the center of the ellipse always has at least one solution (i.e. the radicand is never negative). In the case that the radii are scaled up using equation (F.6.6.3), the radicand of (F.6.5.2) is zero and there is exactly one solution for the center of the ellipse.

## F.7 Text selection implementation notes

The following implementation notes describe the algorithm for deciding which characters are selected during a text selection operation.

As the text selection operation occurs (e.g., while the user clicks and drags the mouse to identify the selection), the user agent determines a *start selection position* and an *end selection position*, each of which represents a position in the text string between two characters. After determining start selection position and end selection position, the user agent selects the appropriate characters, where the resulting text selection consists of either:

- no selection or
- a *start character*, an *end character* (possibly the same character), and all of the characters within the same **'text'** element whose position in the DOM is logically between the start character and end character.

On systems with pointer devices, to determine the *start selection position*, the SVG user agent determines which boundary between characters corresponding to rendered glyphs is the best target (e.g., closest) based on the current pointer location at the time of the event that initiates the selection operation (e.g., the mouse down event). The user agent then tracks the completion of the selection operation (e.g., the mouse drag, followed ultimately by the mouse up). At the end of the selection operation, the user agent determines which boundary between characters is the best target (e.g., closest) for the *end selection position*.

If no character reordering has occurred due to bidirectionality, then the selection consists of all characters between the *start selection position* and *end selection position*. For example, if a **'text'** element contains the string "abcdef" and the start selection position and end selection positions are 0 and 3 respectively (assuming the left side of the "a" is position zero), then the selection will consist of "abc".

When the user agent is implementing selection of bidirectional text, and when the selection starts (or ends) between characters which are not contiguous in logical order, then there might be multiple potential combinations of characters that can be considered part of the selection. The algorithms to choose among the combinations of potential selection options shall choose the selection option which most closely matches the text string's visual rendering order.

When multiple characters map inseparably to a given set of one or more glyphs, the user agent can either disallow the selection to start in the middle of the glyph set or can attempt to allocate portions of the area taken up by the glyph set to the characters that correspond to the glyph.

For systems which support pointer devices such as a mouse, the user agent is required to provide a mechanism for selecting text even when the given text has associated event handlers or links, which might block text selection due to event processing precedence rules (see Pointer events). One implementation option: For platforms which support a pointer device such as a mouse, the user agent may provide for a small additional region around character cells which initiates text selection operations but does not initiate event handlers or links.

## F.8 Printing implementation notes

For user agents which support both zooming on display devices and printing, it is recommended that the default printing option produce printed output that reflects the display device's current view of the current SVG document fragment (assuming there is no media-specific styling), taking into account any zooming and panning done by the user, the current state of animation, and any document changes due to DOM and scripting . Thus, if the user zooms into a particular area of a map on the display device and then requests a hardcopy, the hardcopy should show the same view of the map as appears on the display device. If a user pauses an animation and prints, the hardcopy should show the same graphics as the currently paused picture on the display device. If scripting has added or removed elements from the document, then the hardcopy should reflect the same changes that would be reflected on the display.

When an SVG document is rendered on a static-only device such as a printer which does not support SVG's animation and scripting and facilities, then the user agent shall ignore any animation and scripting elements in the document and render the remaining graphics elements according to the rules in this specification.

---

04 September 2001

# Appendix G: Conformance Criteria

## Contents



**This appendix is normative.**

## G.1 Introduction

Different sets of SVG conformance criteria exist for:

- Conforming SVG Document Fragments
- Conforming SVG Stand-Alone Files
- Conforming SVG Included Documents
- Conforming SVG Generators
- Conforming SVG Interpreters
- Conforming SVG Viewers

## G.2 Conforming SVG Document Fragments

An SVG document fragment is a *Conforming SVG Document Fragment* if it adheres to the specification described in this document (Scalable Vector Graphics (SVG) Specification) including SVG's DTD (see Document Type Definition) and also:

- (relative to XML) is well-formed.
- if all non-SVG namespace elements and attributes and all **xmlns** attributes which refer to non-SVG namespace elements are removed from the given document, and if an

appropriate XML declaration (i.e., <?xml...?>) is included at the top of the document, and if an appropriate document type declaration (i.e., <!DOCTYPE svg ... >) which points to the SVG DTD is included immediately thereafter, the result is a valid XML document.

- conforms to the following W3C Recommendations:
  - the XML 1.0 specification (Extensible Markup Language (XML) 1.0).
  - (if any namespaces other than SVG are used in the document) Namespaces in XML.
  - any use of CSS shall conform to Cascading Style Sheets, level 2 CSS2 Specification.
  - any references to external style sheets shall conform to Associating stylesheets with XML documents.

The SVG language or these conformance criteria provide no designated size limits on any aspect of SVG content. There are no maximum values on the number of elements, the amount of character data, or the number of characters in attribute values.

## G.3 Conforming SVG Stand-Alone Files

A file is a *Conforming SVG Stand-Alone File* if:

- it is an XML document.
- its root element is an **'svg'** element.
- it conforms to the criteria for Conforming SVG Document Fragment.

## G.4 Conforming SVG Included Document Fragments

SVG document fragments can be included within parent XML documents using the XML namespace facilities described in Namespaces in XML.

An SVG document fragment that is included within a parent XML document is a *Conforming Included SVG Document Fragment* if the SVG document fragment, when taken out of the parent XML document, conforms to the SVG Document Type Definitions (DTD).

In particular, note that individual elements from the SVG namespace *cannot* be used by themselves. Thus, the SVG part of the following document is *not* conforming:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE SomeParentXMLGrammar PUBLIC "-//SomeParent"
  "http://SomeParentXMLGrammar.dtd">
<ParentXML>
  <!-- Elements from ParentXML go here -->

  <!-- The following is not conforming -->
  <z:rect xmlns:z="http://www.w3.org/2000/svg"
            x="0" y="0" width="10" height="10" />
```

```
  <!-- More elements from ParentXML go here -->
</ParentXML>
```

Instead, for the SVG part to become a Conforming Included SVG Document Fragment, the file could be modified as follows (the example below shows the use of Stylable SVG):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE SomeParentXMLGrammar PUBLIC "-//SomeParent"
  "http://SomeParentXMLGrammar.dtd">
<ParentXML>
  <!-- Elements from ParentXML go here -->

  <!-- The following is conforming -->
  <z:svg xmlns:z="http://www.w3.org/2000/svg"
         width="100px" height="100px" >
    <z:rect x="0" y="0" width="10" height="10" />
  </z:svg>

  <!-- More elements from ParentXML go here -->
</ParentXML>
```

## G.5 Conforming SVG Generators

A *Conforming SVG Generator* is a program which:

- always creates at least one of Conforming SVG Document Fragments, Conforming SVG Stand-Alone Files or Conforming SVG Included Documents.
- does not create non-conforming SVG document fragments of any of the above types.

Additionally, an authoring tool which is a Conforming SVG Generator conforms to all of the Priority 1 accessibility guidelines from the document "Authoring Tool Accessibility Guidelines 1.0" [ATAG] that are relevant to generators of SVG content. (Priorities 2 and 3 are encouraged but not required for conformance.)

SVG generators are encouraged to follow W3C developments in the area of internationalization. Of particular interest is the *W3C Character Model* and the concept of *Webwide Early Uniform Normalization*, which promises to enhance the interchangability of Unicode character data across users and applications. Future versions of the SVG specification are likely to require support of the *W3C Character Model* in Conforming SVG Generators.

## G.6 Conforming SVG Interpreters

An SVG interpreter is a program which can parse and process SVG document fragments. Examples of SVG interpreters are server-side transcoding tools (e.g., a tool which converts SVG content into modified SVG content) or analysis tools (e.g., a tool which extracts the text content from SVG content). An SVG viewer also satisfies the requirements of an SVG interpreter in that it can parse and process SVG document fragments, where processing consists of rendering the SVG content to the target medium.

In a *Conforming SVG Interpreter*, the XML parser must be able to parse and process all XML constructs defined within [XML10] and [XML-NS].

There are two sub-categories of *Conforming SVG Interpreters*:

- *Conforming Static SVG Interpreters* must be able to parse and process the static language features of SVG that correspond to the feature string "org.w3c.svg.static" (see Feature strings).
- In addition to the requirements for the static category, *Conforming Dynamic SVG Interpreters* must be able to parse and process the language features of SVG that correspond to the feature string "org.w3c.svg.dynamic" (see Feature strings) and which support all of the required features in the SVG DOM described in this specification.

A Conforming SVG Interpreter must parse any SVG document correctly. It is not required to interpret the semantics of all features correctly.

Note: A transcoder from SVG into another graphics representation, such as an SVG-to-raster transcoder, represents a viewer, and thus viewer conformance criteria apply. (See Conforming SVG Viewers.)

## G.7 Conforming SVG Viewers

An SVG viewer is a program which can parse and process an SVG document fragment and render the contents of the document onto some sort of output medium such as a display or printer; thus, an *SVG Viewer* is also an *SVG Interpreter*.

There are two sub-categories of *Conforming SVG Viewers*:

- *Conforming Static SVG Viewers* support the static language features of SVG that correspond to the feature string "org.w3c.svg.static" (see Feature strings). This category often corresponds to platforms and environments which only render static documents, such as printers.
- *Conforming Dynamic SVG Viewers* support the language features of SVG that correspond to the feature string "org.w3c.svg.dynamic" (see Feature strings). This category often applies to platforms and environments such as common Web browsers which support user interaction and dynamic document content (i.e., documents whose content can change over time). (User interaction includes support for hyperlinking, events [e.g., mouse clicks], text selection, zooming and panning [see Interactivity]. Dynamic document content can be achieved via declarative animation or by scripts modifying the SVG DOM.)

Specific criteria that apply to both *Conforming Static SVG Viewers* and *Conforming Dynamic SVG Viewers*:

- The program must also be a Conforming SVG Interpreter,

- For interactive user environments, facilities must exist for zooming and panning of stand-alone SVG documents or SVG document fragments embedded within parent XML documents.
- In environments that have appropriate user interaction facilities, the viewer must support the ability to activate hyperlinks.
- If printing devices are supported, SVG content must be printable at printer resolutions with the same graphics features available as required for display (e.g., the specified colors must be rendered on color printers).
- On systems where this information is available, the parent environment must provide the viewer with information about physical device resolution. In situations where this information is impossible to determine, the parent environment shall pass a reasonable value for device resolution which tends to approximate most common target devices.
- The viewer must support JPEG [JPEG] and PNG [PNG10] image formats.
- Resampling of image data must be consistent with the specification of property **'image-rendering'**.
- The viewer must support alpha channel blending of the image of the SVG content onto the target canvas.
- SVG implementations which support the HTTP protocol must correctly support gzip-encoded SVG data streams according to the HTTP 1.1 specification [RFC2616]; thus, the client must specify "Accept-Encoding: gzip" [HTTP-ACCEPT-ENCODING] on its request-header field and then decompress any gzip-encoded data streams that are downloaded from the server. If the implementation supports progressive rendering, the implementation should also support progressive rendering of compressed data streams.
- The viewer must support base64 encoded content using the "data:" protocol [RFC2397] wherever URI referencing of whole documents (such as raster images, SVG documents, fonts and color profiles) is permitted within SVG content. (Note: fragments of SVG content which do not constitute an entire SVG document are not available using the "data:" protocol.)
- The viewer must support the following W3C Recommendations with regard to SVG content:
    - complete support for the XML 1.0 specification [XML10].
    - complete support for inclusion of non-SVG namespaces within SVG content as defined in "Namespaces in XML" [XML-NS]. (Note that data from non-SVG namespaces are included in the DOM but are otherwise ignored.)
- All visual rendering must be accurate to within one device pixel to the mathematically correct result.
- On systems which support accurate sRGB [SRGB] color, all sRGB color computations and all resulting color values must be accurate to within one sRGB color component value, where sRGB color component values range from 0 to 255.

Although anti-aliasing support is not a strict requirement for a Conforming SVG Viewer, it is highly recommended for display devices. Lack of anti-aliasing support will generally result in poor results on display devices.

Specific criteria that apply to only *Conforming Dynamic SVG Viewers*:

- In Web browser environments, the viewer must have the ability to search and select text strings within SVG content.
- If display devices are supported, the viewer must have the ability to select and copy text

from SVG content to the system clipboard.
- The viewer must have complete support for an ECMAScript binding of the SVG Document Object Model.

The Web Accessibility Initiative [WAI] is defining "User Agent Accessibility Guidelines 1.0" [UAAG]. Viewers are encouraged to conform to the Priority 1 accessibility guidelines defined in this document, and preferably also Priorities 2 and 3. Once the guidelines are completed, a future version of this specification is likely to require conformance to the Priority 1 guidelines in Conforming SVG Viewers.

A higher order concept is that of a *Conforming High-Quality SVG Viewer*, with sub-categories *Conforming High-Quality Static SVG Viewer* and *Conforming High-Quality Dynamic SVG Viewer*.

Both a *Conforming High-Quality Static SVG Viewer* and a *Conforming High-Quality Dynamic SVG Viewer* must support the following additional features:

- Professional-quality results with good processing and rendering performance and smooth, flicker-free animations.
- On low-resolution devices such as display devices at 150dpi or less, support for smooth edges on lines, curves and text. (Smoothing is often accomplished using anti-aliasing techniques.)
- Color management via ICC profile support (i.e., the ability to support colors defined using ICC profiles).
- Resampling of image data must conform to the requirements for Conforming High-Quality SVG Viewers as specified in the description of property **'image-rendering'**.
- At least double-precision floating point computation on coordinate system transformation numerical calculations.

A *Conforming High-Quality Dynamic SVG Viewer* must support the following additional features:

- Progressive rendering and animation effects (i.e., the start of the document will start appearing and animations will start running in parallel with downloading the rest of the document).
- Restricted screen updates (i.e., only required areas of the display are updated in response to redraw events).
- Background downloading of images and fonts retrieved from a Web server, with updating of the display once the downloads are complete.

A *Conforming SVG Viewer* must be able to apply styling properties to SVG content using presentation attributes.

If the user agent includes a CSS2 capability, a *Conforming SVG Viewer* must support CSS styling of SVG content and must support all features from CSS2 (Cascading Style Sheets, level 2 CSS2 Specification) that are described in this specification as applying to SVG (see properties shared with CSS and XSL, Styling with CSS and Facilities from CSS and XSL used by SVG). The supported features from CSS2 must be implemented in accordance with the conformance definitions from the CSS2 specification.

If the user agent includes an HTML or XHTML viewing capability or can apply CSS/XSL styling properties to XML documents, then a *Conforming SVG Viewer* must support resources of MIME type "image/svg+xml" wherever raster image external resources can be used, such as in the HTML or XHTML **'img'** element and in CSS/XSL properties that can refer to raster image resources (e.g., **'background-image'**).

---

04 September 2001

# Appendix H: Accessibility Support

## Contents



*This appendix is informative, not normative.*

## H.1 WAI Accessibility Guidelines

This appendix explains how accessibility guidelines published by W3C's Web Accessibility Initiative (WAI) apply to SVG.

1. The "Web Content Accessibility Guidelines 1.0" [WCAG] explains how authors can create Web content that is accessible to people with disabilities.
2. The "Authoring Tool Accessibility Guidelines 1.0" [ATAG] explains how developers can design accessible authoring tools such as SVG authoring tools. To conform to the SVG specification, an SVG authoring tool must conform to ATAG (priority 1). SVG support for element grouping and reuse is relevant to designing accessible SVG authoring tools.
3. The "User Agent Accessibility Guidelines 1.0" [UAAG] explains how developers can design accessible user agents such as SVG-enabled browsers. To conform to the SVG specification, an SVG user agent should conform to UAAG. SVG support for scaling, style sheets, the DOM, and metadata are all relevant to designing accessible SVG user agents.

The W3C Note "Accessibility Features of SVG" [SVG-ACCESS] explains in detail how the requirements of the three guidelines apply to SVG.

## H.2 SVG Content Accessibility Guidelines

This section explains briefly how authors can create accessible SVG documents; it summarizes

"Accessibility Features of SVG" [SVG-ACCESS].

**Provide text equivalents for graphics.**

- When the text content of a graphic (e.g., in a **'text'** element) explains its function, no text equivalent is required. Use the **'title'** child element to explain the function **'text'** elements whose meaning is not clear from their text content.
- When a graphic does not include explanatory text content, it requires a text equivalent. If the equivalent is complex, use the **'desc'** element, otherwise use the **'title'** child element.
- If a graphic is built from meaningful parts, build the description from meaningful parts.

**Do not rely on color alone.**

- Do not use color alone to convey information.
- Ensure adequate color contrast. Use style sheets so that users who require certain color combinations may apply them through user style sheets.

**Use markup and style sheets and do so properly.**

- Represent text as character data, not as images or curves. Style text with fonts. Authors may describe their own fonts in SVG.
- Separate structure from presentation.
- Use the **'g'** element and rich descriptions to structure SVG documents. Reuse named objects.
- Publish highly-structured documents, not just graphical representations. Documents that are rich in structure may be rendered graphically, as speech, or as braille. For example, express mathematical relationships in MathML [MATHML] and use SVG for explanatory graphics.
- Author documents that validate to the SVG grammar.
- Use style sheets to specify graphical and aural presentation.
- Use relative units in style sheets.

**Clarify natural language usage.**

- Use **xml:lang** to identify the natural language of content and changes in natural language.

**Ensure that dynamic content is accessible.**

- Ensure that text equivalents for dynamic content are updated when the dynamic content changes.
- Ensure that SVG documents are usable when scripts or other programmatic objects are turned off or not supported.

---

04 September 2001

# Appendix I: Internationalization Support

## Contents



*This appendix is informative, not normative.*

## I.1 Introduction

This appendix provides a brief summary of SVG's support for internationalization. The appendix is hyperlinked to the sections of the specification which elaborate on particular topics.

## I.2 Internationalization and SVG

SVG is an application of XML [XML10] and thus supports Unicode [UNICODE], which defines a standard universal character set.

Additionally, SVG provides a mechanism for precise control of the glyphs used to draw text strings, which is described in Alternate glyphs. This facility provides:

- the ability to specify the rendering of particular glyphs which might not be accessible when defining character data using Unicode
- the ability to override the user agent's character-to-glyph algorithms
- the ability to follow the guidelines for normalizing character data for the purposes of enhanced interoperability (see [CHARMOD]), while still having precise control over the glyphs that are drawn.

SVG supports:

- Horizontal, left-to-right text found in Roman scripts (see the **'writing-mode'** property)
- Vertical and vertical-ideographic text (see the **'writing-mode'** property)

- Bidirectional text (for languages such as Arabic and Hebrew - see the **'direction'** and **'unicode-bidi'** properties)

SVG fonts support contextual glyph selection for Arabic and Han text.

Multi-language SVG documents are possible by utilizing the **systemLanguage** attribute to have different text strings appear based on the client machine's language setting.

## I.3 SVG Internationalization Guidelines

SVG generators should follow W3C guidelines for normalizing character data [CHARMOD]. When precise control over glyph selection is required, use the facilities for Alternate glyphs to override the user agent's character-to-glyph mapping algorithms.

---

04 September 2001

# Appendix J: Minimizing SVG File Sizes

*This appendix is informative, not normative.*

Considerable effort has been made to make SVG file sizes as small as possible while still retaining the benefits of XML and achieving compatibility and leverage with other W3C specifications.

Here are some of the features in SVG that promote small file sizes:

- SVG's path data definition was defined to produce a compact data stream for vector graphics data: all commands are one character in length; relative coordinates are available; separator characters do not have to be supplied when tokens can be identified implicitly; smooth curve formulations are available (cubic Béziers, quadratic Béziers and elliptical arcs) to prevent the need to tesselate into polylines; and shortcut formulations exist for common forms of cubic Bézier segments, quadratic Bézier segments, and horizontal and vertical straight line segments so that the minimum number of coordinates need to be specified.
- Text can be specified using XML character data -- no need to convert to outlines.
- SVG contains a facility for defining symbols once and referencing them multiple times using different visual attributes and different sizing, positioning, clipping and client-side filter effects
- User agents that support styling with CSS can use CSS selectors and property inheritance to define commonly used sets of attributes once as named styles.
- Filter effects allow for compelling visual results and effects typically found only in image-authoring tools using small amounts of vector and/or raster data

Additionally, HTTP/1.1 allows for compressed data to be passed from server to client, which can result in significant file size reduction. Here are some sample compression results using gzip compression on SVG documents:

| Uncompressed SVG | With gzip compression | Compression ratio |
|---|---|---|
| 12,912 | 2,463 | 81% |
| 12,164 | 2,553 | 79% |
| 11,613 | 2,617 | 77% |
| 18,689 | 4,077 | 78% |
| 13,024 | 2,041 | 84% |

A related issue is progressive rendering. Some SVG viewers will support:

- the ability to display the first parts of an SVG document fragments as the remainder of the document is downloaded from the server; thus, the user will see part of the SVG drawing right away and interact with it, even if the SVG file size is large.
- delayed downloading of images and fonts. Just like some HTML browsers, some SVG viewers will download images and WebFonts last, substituting a temporary image and system fonts, respectively, until the given image and/or font is available.

Here are techniques for minimizing SVG file sizes and minimizing the time before the user is able to start interacting with the SVG document fragments:

- Construct the SVG file such that any links which the user might want to click on are included at the beginning of the SVG file
- Use default values whenever possible rather than defining all attributes and properties explicitly.
- Take advantage of the path data data compaction facilities: use relative coordinates; use *h* and *v* for horizontal and vertical lines; use *s* or *t* for cubic and quadratic Bézier segments whenever possible; eliminate extraneous white space and separators.
- Utilize symbols if the same graphic appears multiple times in the document
- For user agents that support styling with CSS, utilize CSS property inheritance and selectors to consolidate commonly used properties into named styles or to assign the properties to a parent **'g'** element.
- Utilize filter effects to help construct graphics via client-side graphics operations.

---

04 September 2001

# Appendix K: References

## Contents



## K.1 Normative references


**[ATAG]**
"Authoring Tool Accessibility Guidelines 1.0", J. Treviranus, J. Richards, I. Jacobs, C. McCathieNevile, editors, 3 February 2000.
Available at http://www.w3.org/TR/ATAG10/

**[COLORIMETRY]**
"Colorimetry, Second Edition", CIE Publication 15.2-1986, ISBN 3-900-734-00-3.
Available at http://www.cie.co.at/cie/publ/abst/15-2-86.html.

**[CSS2]**
"Cascading Style Sheets, level 2", B. Bos, H. W. Lie, C. Lilley, I. Jacobs, 12 May 1998.
Available at http://www.w3.org/TR/REC-CSS2/.

**[DOM1]**
"Document Object Model (DOM) Level 1 Specification", V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, L. Wood, editors, 1 October 1998.
Available at http://www.w3.org/TR/REC-DOM-Level-1/

**[DOM2]**
"Document Object Model (DOM) Level 2 Core Specification", A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, editors, 13 November, 2000.
Available at http://www.w3.org/TR/DOM-Level-2/

**[ICC32]**
"Specification ICC.1:1998-09, File Format for Color Profiles", 1998.
Available at http://www.color.org/ICC-1_1998-09.PDF.

"Document ICC.1A:1999-04, Addendum 2 to Spec. ICC.1:1998-09", 1999.
Available at http://www.color.org/ICC-1A_1999-04.PDF.

**[ISO8601]**
"Data elements and interchange formats - Information interchange - Representation of dates and times", International Organization for Standardization, 1998.

**[JPEG]**
ISO/IEC 10918. Available from the International Organization for Standardization (ISO).

**[PNG10]**

"PNG (Portable Network Graphics) Specification, Version 1.0 specification", T. Boutell ed., 1 October 1996.
Available at http://www.w3.org/TR/REC-png-multi.html.

**[PORTERDUFF]**
"Compositing Digital Images", T. Porter, T. Duff, SIGGRAPH '84 Conference Proceedings, Association for Computing Machinery, Volume 18, Number 3, July 1984.

**[RFC1952]**
"GZIP file format specification version 4.3", P. Deutsch, May 1996.
Available at http://www.ietf.org/rfc/rfc1952.txt.

**[RFC2044]**
"UTF-8, a transformation format of Unicode and ISO 10646", F. Yergeau, October 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.
Available at http://www.ietf.org/rfc/rfc2044.txt.

**[RFC2045]**
"Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.
Available at http://www.ietf.org/rfc/rfc2045.txt.

**[RFC2046]**
"Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed and N. Borenstein, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.
Available at http://www.ietf.org/rfc/rfc2046.txt.

**[RFC2119]**
"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Available at http://www.ietf.org/rfc/rfc2119.txt.

**[RFC2141]**
"URN Syntax", R. Moats, May 1997.
Available at http://www.ietf.org/rfc/rfc2141.txt.

**[RFC2279]**
"UTF-8, a transformation format of ISO 10646", F. Yergeau, January 1998.
Available at http://www.ietf.org/rfc/rfc2279.txt.

**[RFC2318]**
"The text/css Media Type", H. Lie, B. Bos, C. Lilley, March 1998.
Available at http://www.ietf.org/rfc/rfc2318.txt.

**[RFC2396]**
'Uniform Resource Identifiers (URI): Generic Syntax', T. Berners-Lee, R. Fielding, L. Masinter, August 1998. Note that RFC 2396 updates [RFC1738] and [RFC1808].
Available at http://www.ietf.org/rfc/rfc2396.txt.

**[RFC2397]**
'The "data" URL scheme', L. Masinter, August 1998.
Available at http://www.ietf.org/rfc/rfc2397.txt.

**[RFC2616]**
"Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. This RFC obsoletes RFC 2068.
Available at http://www.ietf.org/rfc/rfc2616.txt.

**[RFC2732]**
"RFC 2732: Format for Literal IPv6 Addresses in URL's", Internet Engineering Task Force, R. Hinden, B. Carpenter, L. Masinter, December 1999.

Available at http://www.ietf.org/rfc/rfc2732.txt.

**[RFC3023]**
"XML Media Types", M. Murata, S. St.Laurent, D. Kohn, January 2001.
Available at http://www.ietf.org/rfc/rfc3023.txt.

**[RFC3066]**
"Tags for the Identification of Languages", H. Alvestrand, January 2001.
Available at http://www.ietf.org/rfc/rfc3066.txt.

**[SMIL1]**
"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", P. Hoschka, editor, 15 June 1998.
Available at http://www.w3.org/TR/REC-smil/

**[SMILANIM]**
"SMIL Animation", P. Schmitz, A. Cohen, editors, 31-July-2000.
Available at http://www.w3.org/TR/2001/REC-smil-animation-20010904/

**[SRGB]**
IEC 61966-2-1 (1999-10) - "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB", ISBN: 2-8318-4989-6 - ICS codes: 33.160.60, 37.080 - TC 100 - 51 pp.
Available at: http://www.iec.ch/nr1899.htm.

**[UNICODE]**
The Unicode Consortium. "The Unicode Standard, Version 3.1". Refer to http://www.unicode.org/unicode/standard/versions/.

**[URI]**
'Uniform Resource Identifiers (URI): Generic Syntax', T. Berners-Lee, R. Fielding, L. Masinter, August 1998. Note that RFC 2396 updates [RFC1738] and [RFC1808].
Available at http://www.ietf.org/rfc/rfc2396.txt. (The term "URI-reference" is defined in Section 4: URI References.)

**[WCAG]**
"Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, I. Jacobs, editors, 5-May-1999.
Available at:
http://www.w3.org/TR/WAI-WEBCONTENT/.

**[XLINK]**
"XML Linking Language (XLink)", S. DeRose, E. Maler, D. Orchard, editors, 20 December 2000.
Available at http://www.w3.org/TR/xlink/

**[XML10]**
"Extensible Markup Language (XML) 1.0 (Second Edition)", T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, editors, 6 October 2000.
Available at http://www.w3.org/TR/REC-xml/.

**[XML-BASE]**
"XML Base", J. Marsh , editor, 20 December 2000.
Available at http://www.w3.org/TR/xmlbase/.

**[XML-NS]**
"Namespaces in XML", T. Bray, D. Hollander, A. Layman, editors, 14 January 1999.
Available at http://www.w3.org/TR/REC-xml-names/.

**[XML-SS]**
"Associating Style Sheets with XML documents Version 1.0", James Clark, editor, 29 June 1999.

Available at http://www.w3.org/TR/xml-stylesheet/.

**[XPTR]**

"XML Pointer Language (XPointer) Version 1.0", S. DeRose, R. Daniel Jr., E. Maler, editors, 8 January 2001.
Available at http://www.w3.org/TR/xptr

## K.2 Informative references

**[CHARMOD]**

"Character Model for the World Wide Web 1.0", M. Dürst, F. Yergeau, M. Wolf, A. Freytag, T. Texin, editors, 26 January 2001.
Available at http://www.w3.org/TR/charmod/

**[DCORE]**

The Dublin Core. For more information, refer to http://purl.org/DC.

**[FOLEY-VANDAM]**

"Computer Graphics : Principles and Practice, Second Edition", James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, Addison-Wesley, pp. 488-491.

**[HTML4]**

"HTML 4.01 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999.
Available at http://www.w3.org/TR/html401/. The Recommendation defines three document type definitions: Strict, Transitional, and Frameset, all reachable from the Recommendation.

**[MATHML]**

"Mathematical Markup Language (MathML) Version 2.0", D. Carlisle, P. Ion, R. Miner, N. Poppelier, editors, 7 July 1999.
Available at http://www.w3.org/TR/MathML2/.

**[MIMETYPES]**

List of registered content types (MIME types). Download a list of registered content types from ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/.

**[OPENTYPE]**

See http://www.microsoft.com/OpenType/OTSpec/.

**[RDF10]**

"Resource Description Framework (RDF) Model and Syntax Specification", O. Lassila, R. Swick, eds., 22 February 1999. This document is http://www.w3.org/TR/REC-rdf-syntax/.

**[SVG-ACCESS]**

"Accessibility Features of SVG", C. McCathieNevile, M. Koivunen, 7 August 2000, http://www.w3.org/TR/SVG-access/.

**[UAAG]**

"User Agent Accessibility Guidelines 1.0", J. Gunderson, I. Jacobs, E. Hansen, editors, 23 October 2000.
Available at http://www.w3.org/TR/UAAG10/

**[WAI]**

Home page for Web Accessibility Initiative:
http://www.w3.org/WAI/.

**[XHTML]**

"XHTML(tm) 1.0: The Extensible HyperText Markup Language", 26 January 2000,

Available at http://www.w3.org/TR/xhtml1/.

**[XSL]**

"Extensible Stylesheet Language (XSL) Version 1.0", S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, editors, 21 November 2000.
Available at http://www.w3.org/TR/xsl/

**[XSLT]**

"XSL Transformations (XSLT) Version 1.0", J. Clark, editor, 16 November 1999.
Available at http://www.w3.org/TR/xslt

---

04 September 2001

# Appendix L: Element Index

The following are the elements in the SVG language:

- feMorphology
- feOffset
- fePointLight
- feSpecularLighting
- feSpotLight
- feTile
- feTurbulence
- filter
- font
- font-face
- font-face-format
- font-face-name
- font-face-src
- font-face-uri
- foreignObject
- g
- glyph
- glyphRef
- hkern
- image
- line
- linearGradient
- marker
- mask
- metadata
- missing-glyph
- mpath
- path
- pattern
- polygon
- polyline
- radialGradient
- rect
- script
- set
- stop
- style
- svg
- switch
- symbol
- text
- textPath
- title
- tref
- tspan

- use
- view
- vkern

---

04 September 2001

# Appendix M: Attribute Index

| Attribute or entity which defines a collection of attributes | Used in these elements and entities | Value | Type | Default |
|---|---|---|---|---|
| %PresentationAttributes-All; | feImage, svg, g, defs, symbol, use, switch, marker, pattern, mask, filter, a, font, glyph, missing-glyph, foreignObject | | | |
| %PresentationAttributes-Color; | PresentationAttributes-All, image, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, linearGradient, radialGradient, stop, clipPath, feDiffuseLighting, feFlood, feSpecularLighting | | | |
| %PresentationAttributes-Containers; | PresentationAttributes-All | | | |
| %PresentationAttributes-FillStroke; | PresentationAttributes-All, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, clipPath | | | |
| %PresentationAttributes-FilterPrimitives; | PresentationAttributes-All, feBlend, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feFlood, feGaussianBlur, feMerge, feMorphology, feOffset, feSpecularLighting, feTile, feTurbulence | | | |
| %PresentationAttributes-FontSpecification; | PresentationAttributes-All, text, tspan, tref, textPath, altGlyph, glyphRef, clipPath | | | |
| %PresentationAttributes-Gradients; | PresentationAttributes-All, linearGradient, radialGradient, stop | | | |
| %PresentationAttributes-Graphics; | PresentationAttributes-All, image, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, clipPath | | | |
| %PresentationAttributes-Images; | PresentationAttributes-All, image | | | |
| %PresentationAttributes-LightingEffects; | PresentationAttributes-All, feDiffuseLighting, feSpecularLighting | | | |
| %PresentationAttributes-Markers; | PresentationAttributes-All, path, line, polyline, polygon | | | |
| %PresentationAttributes-TextContentElements; | PresentationAttributes-All, text, tspan, tref, textPath, altGlyph, clipPath | | | |
| %PresentationAttributes-TextElements; | PresentationAttributes-All, text, clipPath | | | |
| %PresentationAttributes-Viewports; | PresentationAttributes-All, image | | | |
| %PresentationAttributes-feFlood; | PresentationAttributes-All, feFlood | | | |
| %animAdditionAttrs; | animate, animateColor, animateMotion, animateTransform | | | |
| %animAttributeAttrs; | animate, set, animateColor, animateTransform | | | |
| %animElementAttrs; | animate, set, animateMotion, animateColor, animateTransform | | | |
| %animTimingAttrs; | animate, set, animateMotion, animateColor, animateTransform | | | |
| %animValueAttrs; | animate, animateColor, animateTransform | | | |
| %animationEvents; | animate, set, animateMotion, animateColor, animateTransform | | | |
| %component_transfer_function_attributes; | feFuncR, feFuncG, feFuncB, feFuncA | | | |
| %documentEvents; | svg | | | |

| | | | | |
|---|---|---|---|---|
| %filter_primitive_attributes; | feMerge, feTurbulence | | | |
| %filter_primitive_attributes_with_in; | feComponentTransfer, feFlood, feTile, feBlend, feColorMatrix, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feGaussianBlur, feMorphology, feOffset, feSpecularLighting | | | |
| %graphicsElementEvents; | g, defs, symbol, switch, svg, use, image, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, a, foreignObject | | | |
| %langSpaceAttrs; | svg, g, defs, desc, title, symbol, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, marker, pattern, clipPath, mask, filter, feImage, a, foreignObject | | | |
| %stdAttrs; | altGlyphDef, altGlyphItem, font-face-src, metadata, svg, g, defs, desc, title, symbol, use, image, switch, style, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, glyphRef, marker, color-profile, linearGradient, radialGradient, stop, pattern, clipPath, mask, filter, feDistantLight, fePointLight, feSpotLight, feBlend, feColorMatrix, feComponentTransfer, feFuncR, feFuncG, feFuncB, feFuncA, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feFlood, feGaussianBlur, feImage, feMerge, feMergeNode, feMorphology, feOffset, feSpecularLighting, feTile, feTurbulence, cursor, a, view, script, animate, set, animateMotion, mpath, animateColor, animateTransform, font, glyph, missing-glyph, hkern, vkern, font-face, font-face-uri, font-face-format, font-face-name, definition-src, foreignObject | | | |
| %testAttrs; | svg, g, defs, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, pattern, clipPath, mask, cursor, a, animate, set, animateMotion, animateColor, animateTransform, foreignObject | | | |
| %xlinkRefAttrs; | tref, textPath, altGlyph, glyphRef, color-profile, linearGradient, radialGradient, pattern, filter, cursor, script, mpath, font-face-uri, definition-src | | | |
| %xlinkRefAttrsEmbed; | use, image, feImage | | | |
| accent-height | font-face | %Number; | #IMPLIED | |
| accumulate | animAdditionAttrs | (none \| sum) | | none |
| alignment-baseline | PresentationAttributes-TextContentElements | ( baseline \| top \| before-edge \| text-top \| text-before-edge \| middle \| bottom \| after-edge \| text-bottom \| text-after-edge \| ideographic \| lower \| hanging \| mathematical \| inherit) | | |
| alphabetic | font-face | %Number; | #IMPLIED | |
| amplitude | component_transfer_function_attributes | %Number; | #IMPLIED | |
| animate | filter | | | |
| arabic-form | glyph | CDATA | #IMPLIED | |
| ascent | font-face | %Number; | #IMPLIED | |
| attributeType | animAttributeAttrs | CDATA | #IMPLIED | |
| azimuth | feDistantLight | %Number; | #IMPLIED | |
| baseFrequency | feTurbulence | %NumberOptionalNumber; | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| baseline-shift | PresentationAttributes-TextContentElements | %BaselineShiftValue; | #IMPLIED | |
| bbox | font-face | CDATA | #IMPLIED | |
| bias | feConvolveMatrix | %Number; | #IMPLIED | |
| by | animValueAttrs, animateMotion | CDATA | #IMPLIED | |
| calcMode | animateMotion | (discrete \| linear \| paced \| spline) | | paced |
| cap-height | font-face | %Number; | #IMPLIED | |
| class | svg, g, defs, desc, title, symbol, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, glyphRef, marker, linearGradient, radialGradient, stop, pattern, clipPath, mask, filter, feDiffuseLighting, feFlood, feImage, feSpecularLighting, a, font, glyph, missing-glyph, foreignObject | %ClassList; | #IMPLIED | |
| clip | PresentationAttributes-Viewports | %ClipValue; | #IMPLIED | |
| clip-path | PresentationAttributes-Graphics | %ClipPathValue; | #IMPLIED | |
| clip-rule | PresentationAttributes-Graphics | %ClipFillRule; | #IMPLIED | |
| clipPathUnits | clipPath | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| color | PresentationAttributes-Color | %Color; | #IMPLIED | |
| color-interpolation | PresentationAttributes-Color | (auto \| sRGB \| linearRGB \| inherit) | #IMPLIED | |
| color-interpolation-filters | PresentationAttributes-FilterPrimitives | (auto \| sRGB \| linearRGB \| inherit) | #IMPLIED | |
| color-profile | PresentationAttributes-Images | CDATA | #IMPLIED | |
| color-rendering | PresentationAttributes-Color | (auto \| optimizeSpeed \| optimizeQuality \| inherit) | #IMPLIED | |
| contentScriptType | svg | %ContentType; | | text/ecmascript |
| contentStyleType | svg | %ContentType; | | |
| cursor | PresentationAttributes-Graphics | %CursorValue; | #IMPLIED | |
| cx | circle, ellipse, radialGradient | %Coordinate; | #IMPLIED | |
| cy | circle, ellipse, radialGradient | %Coordinate; | #IMPLIED | |
| d | glyph, missing-glyph | %PathData; | #IMPLIED | |
| d | path | %PathData; | #REQUIRED | |
| descent | font-face | %Number; | #IMPLIED | |
| diffuseConstant | feDiffuseLighting | %Number; | #IMPLIED | |
| direction | PresentationAttributes-TextContentElements | (ltr \| rtl \| inherit) | #IMPLIED | |
| display | PresentationAttributes-Graphics | ( inline \| block \| list-item \| run-in \| compact \| marker \| table \| inline-table \| table-row-group \| table-header-group \| table-footer-group \| table-row \| table-column-group \| table-column \| table-cell \| table-caption \| none \| inherit ) | | |
| divisor | feConvolveMatrix | %Number; | #IMPLIED | |
| dominant-baseline | PresentationAttributes-TextContentElements | (auto | | |
| dur | animTimingAttrs | CDATA | #IMPLIED | |
| dx | text, tspan, tref, altGlyph | %Lengths; | #IMPLIED | |
| dx | glyphRef, feOffset | %Number; | #IMPLIED | |
| dy | text, tspan, tref, altGlyph | %Lengths; | #IMPLIED | |
| dy | glyphRef, feOffset | %Number; | #IMPLIED | |
| edgeMode | feConvolveMatrix | (duplicate\|wrap\|none) | | duplicate |
| elevation | feDistantLight | %Number; | #IMPLIED | |
| enable-background | PresentationAttributes-Containers | %EnableBackgroundValue; | #IMPLIED | |
| end | animTimingAttrs | CDATA | #IMPLIED | |
| exponent | component_transfer_function_attributes | %Number; | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| externalResourcesRequired | g, defs, symbol, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, marker, linearGradient, radialGradient, pattern, clipPath, mask, filter, feImage, cursor, a, view, script, animate, set, animateMotion, animateColor, animateTransform, font, foreignObject, svg, mpath | %Boolean; | #IMPLIED | |
| feColorMatrix | filter | | | |
| feComposite | filter | | | |
| feGaussianBlur | filter | | | |
| feMorphology | filter | | | |
| feTile | filter | | | |
| fill | animTimingAttrs | (remove \| freeze) | | remove |
| fill | PresentationAttributes-FillStroke | %Paint; | #IMPLIED | |
| fill-opacity | PresentationAttributes-FillStroke | %OpacityValue; | #IMPLIED | |
| fill-rule | PresentationAttributes-FillStroke | %ClipFillRule; | #IMPLIED | |
| filter | PresentationAttributes-Graphics | %FilterValue; | #IMPLIED | |
| filterRes | filter | %NumberOptionalNumber; | #IMPLIED | |
| filterUnits | filter | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| flood-color | PresentationAttributes-feFlood | %SVGColor; | #IMPLIED | |
| flood-opacity | PresentationAttributes-feFlood | %OpacityValue; | #IMPLIED | |
| font-family | PresentationAttributes-FontSpecification | %FontFamilyValue; | #IMPLIED | |
| font-family | font-face | CDATA | #IMPLIED | |
| font-size | PresentationAttributes-FontSpecification | %FontSizeValue; | #IMPLIED | |
| font-size | font-face | CDATA | #IMPLIED | |
| font-size-adjust | PresentationAttributes-FontSpecification | %FontSizeAdjustValue; | #IMPLIED | |
| font-stretch | PresentationAttributes-FontSpecification | (normal | | |
| font-stretch | font-face | (normal \| wider \| narrower \| ultra-condensed \| extra-condensed \| condensed \| semi-condensed \| semi-expanded \| expanded \| extra-expanded \| ultra-expanded \| inherit) | | |
| font-style | PresentationAttributes-FontSpecification | (normal \| italic \| oblique \| inherit) | #IMPLIED | |
| font-style | font-face | CDATA | #IMPLIED | |
| font-variant | PresentationAttributes-FontSpecification | (normal \| small-caps \| inherit) | #IMPLIED | |
| font-variant | font-face | CDATA | #IMPLIED | |
| font-weight | PresentationAttributes-FontSpecification | (normal | | |
| font-weight | font-face | (normal \| bold \| bolder \| lighter \| 100 \| 200 \| 300 \| 400 \| 500 \| 600 \| 700 \| 800 \| 900 \| inherit) | | |
| format | altGlyph, glyphRef | CDATA | #IMPLIED | |
| from | animValueAttrs, animateMotion | CDATA | #IMPLIED | |
| fx | radialGradient | %Coordinate; | #IMPLIED | |
| fy | radialGradient | %Coordinate; | #IMPLIED | |
| g1 | hkern, vkern | CDATA | #IMPLIED | |
| g2 | hkern, vkern | CDATA | #IMPLIED | |
| glyph-name | glyph | CDATA | #IMPLIED | |
| glyph-orientation-horizontal | PresentationAttributes-TextContentElements | %GlyphOrientationHorizontalValue; | #IMPLIED | |
| glyph-orientation-vertical | PresentationAttributes-TextContentElements | %GlyphOrientationVerticalValue; | #IMPLIED | |
| glyphRef | altGlyph, glyphRef | CDATA | #IMPLIED | |
| gradientTransform | linearGradient, radialGradient | %TransformList; | #IMPLIED | |
| gradientUnits | linearGradient, radialGradient | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| hanging | font-face | %Number; | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| height | svg, filter, filter_primitive_attributes, use, pattern, mask | %Length; | #IMPLIED | |
| height | rect, foreignObject, image | %Length; | #REQUIRED | |
| horiz-adv-x | glyph, missing-glyph | %Number; | #IMPLIED | |
| horiz-adv-x | font | %Number; | #REQUIRED | |
| horiz-origin-x | font | %Number; | #IMPLIED | |
| horiz-origin-y | font | %Number; | #IMPLIED | |
| ideographic | font-face | %Number; | #IMPLIED | |
| image-rendering | PresentationAttributes-Graphics | (auto \| optimizeSpeed \| optimizeQuality \| inherit) | #IMPLIED | |
| in | filter_primitive_attributes_with_in, feMergeNode | CDATA | #IMPLIED | |
| in2 | feBlend, feComposite, feDisplacementMap | CDATA | #REQUIRED | |
| intercept | component_transfer_function_attributes | %Number; | #IMPLIED | |
| k | hkern, vkern | %Number; | #REQUIRED | |
| k1 | feComposite | %Number; | #IMPLIED | |
| k2 | feComposite | %Number; | #IMPLIED | |
| k3 | feComposite | %Number; | #IMPLIED | |
| k4 | feComposite | %Number; | #IMPLIED | |
| kernelMatrix | feConvolveMatrix | CDATA | #REQUIRED | |
| kernelUnitLength | feConvolveMatrix, feDiffuseLighting, feSpecularLighting | %NumberOptionalNumber; | #IMPLIED | |
| kerning | PresentationAttributes-TextContentElements | %KerningValue; | #IMPLIED | |
| keyPoints | animateMotion | CDATA | #IMPLIED | |
| keySplines | animValueAttrs, animateMotion | CDATA | #IMPLIED | |
| keyTimes | animValueAttrs, animateMotion | CDATA | #IMPLIED | |
| lang | glyph | %LanguageCodes; | #IMPLIED | |
| lengthAdjust | textPath, text, tspan, tref | (spacing\|spacingAndGlyphs) | #IMPLIED | |
| letter-spacing | PresentationAttributes-TextContentElements | %SpacingValue; | #IMPLIED | |
| lighting-color | PresentationAttributes-LightingEffects | %SVGColor; | #IMPLIED | |
| limitingConeAngle | feSpotLight | %Number; | #IMPLIED | |
| local | color-profile | CDATA | #IMPLIED | |
| marker-end | PresentationAttributes-Markers | %MarkerValue; | #IMPLIED | |
| marker-mid | PresentationAttributes-Markers | %MarkerValue; | #IMPLIED | |
| marker-start | PresentationAttributes-Markers | %MarkerValue; | #IMPLIED | |
| markerHeight | marker | %Length; | #IMPLIED | |
| markerUnits | marker | (strokeWidth \| userSpaceOnUse) | #IMPLIED | |
| markerWidth | marker | %Length; | #IMPLIED | |
| mask | PresentationAttributes-Graphics | %MaskValue; | #IMPLIED | |
| maskContentUnits | mask | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| maskUnits | mask | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| mathematical | font-face | %Number; | #IMPLIED | |
| max | animTimingAttrs | CDATA | #IMPLIED | |
| media | style | %MediaDesc; | #IMPLIED | |
| method | textPath | (align\|stretch) | #IMPLIED | |
| min | animTimingAttrs | CDATA | #IMPLIED | |
| mode | feBlend | (normal \| multiply \| screen \| darken \| lighten) | | |
| name | color-profile | CDATA | #REQUIRED | |
| numOctaves | feTurbulence | %Integer; | #IMPLIED | |
| offset | stop | %NumberOrPercentage; | #REQUIRED | |
| offset | component_transfer_function_attributes | %Number; | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| onabort | documentEvents | %Script; | #IMPLIED | |
| onactivate | graphicsElementEvents | %Script; | #IMPLIED | |
| onbegin | animationEvents | %Script; | #IMPLIED | |
| onclick | graphicsElementEvents | %Script; | #IMPLIED | |
| onend | animationEvents | %Script; | #IMPLIED | |
| onerror | documentEvents | %Script; | #IMPLIED | |
| onfocusin | graphicsElementEvents | %Script; | #IMPLIED | |
| onfocusout | graphicsElementEvents | %Script; | #IMPLIED | |
| onload | graphicsElementEvents | %Script; | #IMPLIED | |
| onmousedown | graphicsElementEvents | %Script; | #IMPLIED | |
| onmousemove | graphicsElementEvents | %Script; | #IMPLIED | |
| onmouseout | graphicsElementEvents | %Script; | #IMPLIED | |
| onmouseover | graphicsElementEvents | %Script; | #IMPLIED | |
| onmouseup | graphicsElementEvents | %Script; | #IMPLIED | |
| onrepeat | animationEvents | %Script; | #IMPLIED | |
| onresize | documentEvents | %Script; | #IMPLIED | |
| onscroll | documentEvents | %Script; | #IMPLIED | |
| onunload | documentEvents | %Script; | #IMPLIED | |
| onzoom | documentEvents | %Script; | #IMPLIED | |
| opacity | PresentationAttributes-Graphics | %OpacityValue; | #IMPLIED | |
| operator | feMorphology | (erode \| dilate) | | erode |
| operator | feComposite | (over \| in \| out \| atop \| xor \| arithmetic) | | over |
| order | feConvolveMatrix | %NumberOptionalNumber; | #REQUIRED | |
| orient | marker | CDATA | #IMPLIED | |
| orientation | glyph | CDATA | #IMPLIED | |
| origin | animateMotion | CDATA | #IMPLIED | |
| overflow | PresentationAttributes-Viewports | (visible \| hidden \| scroll \| auto \| inherit) | #IMPLIED | |
| overline-position | font-face | %Number; | #IMPLIED | |
| overline-thickness | font-face | %Number; | #IMPLIED | |
| panose-1 | font-face | CDATA | #IMPLIED | |
| path | animateMotion | CDATA | #IMPLIED | |
| pathLength | path | %Number; | #IMPLIED | |
| patternContentUnits | pattern | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| patternTransform | pattern | %TransformList; | #IMPLIED | |
| patternUnits | pattern | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| pointer-events | PresentationAttributes-Graphics | ( visiblePainted \| visibleFill \| visibleStroke \| visible \| painted \| fill \| stroke \| all \| none \| inherit) | | |
| points | polyline, polygon | %Points; | #REQUIRED | |
| pointsAtX | feSpotLight | %Number; | #IMPLIED | |
| pointsAtY | feSpotLight | %Number; | #IMPLIED | |
| pointsAtZ | feSpotLight | %Number; | #IMPLIED | |
| preserveAlpha | feConvolveMatrix | %Boolean; | #IMPLIED | |
| preserveAspectRatio | svg, symbol, image, marker, pattern, view | %PreserveAspectRatioSpec; | | xMidYMid meet |
| primitiveUnits | filter | (userSpaceOnUse \| objectBoundingBox) | #IMPLIED | |
| r | radialGradient | %Length; | #IMPLIED | |
| r | circle | %Length; | #REQUIRED | |
| radius | feMorphology | %NumberOptionalNumber; | #IMPLIED | |
| refX | marker | %Coordinate; | #IMPLIED | |
| refY | marker | %Coordinate; | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| rendering-intent | color-profile | (auto \| perceptual \| relative-colorimetric \| saturation \| absolute-colorimetric) | | |
| repeatCount | animTimingAttrs | CDATA | #IMPLIED | |
| repeatDur | animTimingAttrs | CDATA | #IMPLIED | |
| requiredExtensions | testAttrs | %ExtensionList; | #IMPLIED | |
| restart | animTimingAttrs | (always \| never \| whenNotActive) | | always |
| result | filter_primitive_attributes | CDATA | #IMPLIED | |
| rotate | animateMotion | CDATA | #IMPLIED | |
| rotate | text, tspan, tref, altGlyph | %Numbers; | #IMPLIED | |
| rx | rect | %Length; | #IMPLIED | |
| rx | ellipse | %Length; | #REQUIRED | |
| ry | rect | %Length; | #IMPLIED | |
| ry | ellipse | %Length; | #REQUIRED | |
| scale | feDisplacementMap | %Number; | #IMPLIED | |
| seed | feTurbulence | %Number; | #IMPLIED | |
| shape-rendering | PresentationAttributes-Graphics | (auto \| optimizeSpeed \| crispEdges \| geometricPrecision \| inherit) | #IMPLIED | |
| slope | component_transfer_function_attributes, font-face | %Number; | #IMPLIED | |
| spacing | textPath | (auto\|exact) | #IMPLIED | |
| specularConstant | feSpecularLighting | %Number; | #IMPLIED | |
| specularExponent | feSpotLight, feSpecularLighting | %Number; | #IMPLIED | |
| spreadMethod | linearGradient, radialGradient | (pad \| reflect \| repeat) | #IMPLIED | |
| startOffset | textPath | %Length; | #IMPLIED | |
| stdDeviation | feGaussianBlur | %NumberOptionalNumber; | #IMPLIED | |
| stemh | font-face | %Number; | #IMPLIED | |
| stemv | font-face | %Number; | #IMPLIED | |
| stitchTiles | feTurbulence | (stitch \| noStitch) | | noStitch |
| stop-color | PresentationAttributes-Gradients | %SVGColor; | #IMPLIED | |
| stop-opacity | PresentationAttributes-Gradients | %OpacityValue; | #IMPLIED | |
| strikethrough-position | font-face | %Number; | #IMPLIED | |
| strikethrough-thickness | font-face | %Number; | #IMPLIED | |
| stroke | PresentationAttributes-FillStroke | %Paint; | #IMPLIED | |
| stroke-dasharray | PresentationAttributes-FillStroke | %StrokeDashArrayValue; | #IMPLIED | |
| stroke-dashoffset | PresentationAttributes-FillStroke | %StrokeDashOffsetValue; | #IMPLIED | |
| stroke-linecap | PresentationAttributes-FillStroke | (butt \| round \| square \| inherit) | #IMPLIED | |
| stroke-linejoin | PresentationAttributes-FillStroke | (miter \| round \| bevel \| inherit) | #IMPLIED | |
| stroke-miterlimit | PresentationAttributes-FillStroke | %StrokeMiterLimitValue; | #IMPLIED | |
| stroke-opacity | PresentationAttributes-FillStroke | %OpacityValue; | #IMPLIED | |
| stroke-width | PresentationAttributes-FillStroke | %StrokeWidthValue; | #IMPLIED | |
| style | svg, g, defs, desc, title, symbol, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, tspan, tref, textPath, altGlyph, glyphRef, marker, linearGradient, radialGradient, stop, pattern, clipPath, mask, filter, feImage, a, font, glyph, missing-glyph, foreignObject | %StyleSheet; | #IMPLIED | |
| surfaceScale | feDiffuseLighting, feSpecularLighting | %Number; | #IMPLIED | |
| systemLanguage | testAttrs | %LanguageCodes; | #IMPLIED | |
| tableValues | component_transfer_function_attributes | CDATA | #IMPLIED | |
| target | a | %LinkTarget; | #IMPLIED | |
| targetX | feConvolveMatrix | %Integer; | #IMPLIED | |
| targetY | feConvolveMatrix | %Integer; | #IMPLIED | |
| text-anchor | PresentationAttributes-TextContentElements | (start \| middle \| end \| inherit) | #IMPLIED | |

| | | | | |
|---|---|---|---|---|
| text-decoration | PresentationAttributes-TextContentElements | %TextDecorationValue; | #IMPLIED | |
| text-rendering | PresentationAttributes-Graphics | (auto \| optimizeSpeed \| optimizeLegibility \| geometricPrecision \| inherit) | #IMPLIED | |
| textLength | text, tspan, tref, textPath | %Length; | #IMPLIED | |
| title | style | %Text; | #IMPLIED | |
| to | animValueAttrs, animateMotion, set | CDATA | #IMPLIED | |
| transform | g, defs, use, image, switch, path, rect, circle, ellipse, line, polyline, polygon, text, clipPath, a, foreignObject | %TransformList; | #IMPLIED | |
| type | style, script | %ContentType; | #REQUIRED | |
| type | feTurbulence | (fractalNoise \| turbulence) | | |
| type | component_transfer_function_attributes | (identity \| table \| discrete \| linear \| gamma) | | |
| type | feColorMatrix | (matrix \| saturate \| hueRotate \| luminanceToAlpha) | | matrix |
| type | animateTransform | (translate \| scale \| rotate \| skewX \| skewY) | | |
| u1 | hkern, vkern | CDATA | #IMPLIED | |
| u2 | hkern, vkern | CDATA | #IMPLIED | |
| underline-position | font-face | %Number; | #IMPLIED | |
| underline-thickness | font-face | %Number; | #IMPLIED | |
| unicode | glyph | CDATA | #IMPLIED | |
| unicode-bidi | PresentationAttributes-TextContentElements | (normal \| embed \| bidi-override \| inherit) | #IMPLIED | |
| unicode-range | font-face | CDATA | #IMPLIED | |
| units-per-em | font-face | %Number; | #IMPLIED | |
| v-alphabetic | font-face | %Number; | #IMPLIED | |
| v-hanging | font-face | %Number; | #IMPLIED | |
| v-ideographic | font-face | %Number; | #IMPLIED | |
| v-mathematical | font-face | %Number; | #IMPLIED | |
| values | animValueAttrs, animateMotion, feColorMatrix | CDATA | #IMPLIED | |
| version | svg | %Number; | #FIXED | 1.0 |
| vert-adv-y | font, glyph, missing-glyph | %Number; | #IMPLIED | |
| vert-origin-x | font, glyph, missing-glyph | %Number; | #IMPLIED | |
| vert-origin-y | font, glyph, missing-glyph | %Number; | #IMPLIED | |
| viewBox | svg, symbol, marker, pattern, view | %ViewBoxSpec; | #IMPLIED | |
| viewTarget | view | CDATA | #IMPLIED | |
| visibility | PresentationAttributes-Graphics | (visible \| hidden \| inherit) | #IMPLIED | |
| width | svg, use, pattern, mask, filter, filter_primitive_attributes | %Length; | #IMPLIED | |
| width | image, rect, foreignObject | %Length; | #REQUIRED | |
| widths | font-face | CDATA | #IMPLIED | |
| word-spacing | PresentationAttributes-TextContentElements | %SpacingValue; | #IMPLIED | |
| writing-mode | PresentationAttributes-TextElements | (lr-tb \| rl-tb \| tb-rl \| lr \| rl \| tb \| inherit) | #IMPLIED | |
| x | svg, use, image, rect, pattern, mask, filter, filter_primitive_attributes, cursor, foreignObject | %Coordinate; | #IMPLIED | |
| x | text, tspan, tref, altGlyph | %Coordinates; | #IMPLIED | |
| x | glyphRef, fePointLight, feSpotLight | %Number; | #IMPLIED | |
| x-height | font-face | %Number; | #IMPLIED | |
| x1 | line, linearGradient | %Coordinate; | #IMPLIED | |
| x2 | line, linearGradient | %Coordinate; | #IMPLIED | |
| xChannelSelector | feDisplacementMap | (R \| G \| B \| A) | | A |
| xlink:actuate | xlinkRefAttrs, xlinkRefAttrsEmbed | (onLoad) | #FIXED | onLoad |
| xlink:actuate | a | (onRequest) | #FIXED | onRequest |

| | | | | |
|---|---|---|---|---|
| xlink:arcrole | xlinkRefAttrs, xlinkRefAttrsEmbed, a | %URI; | #IMPLIED | |
| xlink:href | glyphRef, color-profile, linearGradient, radialGradient, pattern, filter, script, animElementAttrs, altGlyph | %URI; | #IMPLIED | |
| xlink:href | use, image, tref, textPath, feImage, cursor, a, mpath, font-face-uri, definition-src | %URI; | #REQUIRED | |
| xlink:role | xlinkRefAttrs, xlinkRefAttrsEmbed, a | %URI; | #IMPLIED | |
| xlink:show | xlinkRefAttrsEmbed | (embed) | | embed |
| xlink:show | a | (new\|replace) | | replace |
| xlink:show | xlinkRefAttrs | (other) | | other |
| xlink:title | xlinkRefAttrs, xlinkRefAttrsEmbed, a | CDATA | #IMPLIED | |
| xlink:type | xlinkRefAttrs, xlinkRefAttrsEmbed, a | (simple) | #FIXED | simple |
| xml:base | stdAttrs | %URI; | #IMPLIED | |
| xml:space | langSpaceAttrs | (default\|preserve) | #IMPLIED | |
| xml:space | style | (preserve) | #FIXED | preserve |
| xmlns | svg | CDATA | #FIXED | http://www.w3.org/2000/svg |
| xmlns:xlink | a | CDATA | #FIXED | http://www.w3.org/1999/xlink |
| y | svg, use, image, rect, pattern, mask, filter, filter_primitive_attributes, foreignObject, cursor | %Coordinate; | #IMPLIED | |
| y | text, tspan, tref, altGlyph | %Coordinates; | #IMPLIED | |
| y | glyphRef, fePointLight, feSpotLight | %Number; | #IMPLIED | |
| y1 | line, linearGradient | %Coordinate; | #IMPLIED | |
| y2 | linearGradient, line | %Coordinate; | #IMPLIED | |
| yChannelSelector | feDisplacementMap | (R \| G \| B \| A) | | |
| z | feSpotLight, fePointLight | %Number; | #IMPLIED | |
| zoomAndPan | svg, view | (disable \| magnify) | | magnify |

04 September 2001

# Appendix N: Property Index

| Name | Values | Initial value | Applies to (Default: all) | Inherited? | Percentages (Default: N/A) | Media groups | Animatable |
|---|---|---|---|---|---|---|---|
| 'alignment-baseline' | auto \| baseline \| before-edge \| text-before-edge \| middle \| after-edge \| text-after-edge \| ideographic \| alphabetic \| hanging \| mathematical \| inherit | see property description | 'tspan', 'tref', 'altGlyph', 'textPath' elements | no | | visual | yes |
| 'baseline-shift' | baseline \| sub \| super \| <percentage> \| <length> \| inherit | baseline | 'tspan', 'tref', 'altGlyph', 'textPath' elements | no | refers to the 'line-height' of the 'text' element, which in the case of SVG is defined to be equal to the 'font-size' | visual | yes (non-additive, 'set' and 'animate' elements only) |
| 'clip' | <shape> \| auto \| inherit | auto | elements which establish a new viewport, 'pattern' elements and 'marker' elements | no | | visual | yes |
| 'clip-path' | <uri> \| none \| inherit | none | container elements and graphics elements | no | | visual | yes |
| 'clip-rule' | nonzero \| evenodd \| inherit | nonzero | graphics elements within a 'clipPath' element | yes | | visual | yes |
| 'color' | <color> \| inherit | depends on user agent | elements to which properties 'fill', 'stroke', 'stop-color', 'flood-color', 'lighting-color' apply | yes | | visual | yes |
| 'color-interpolation' | auto \| sRGB \| linearRGB \| inherit | sRGB | container elements, graphics elements and 'animateColor' | yes | | visual | yes |
| 'color-interpolation-filters' | auto \| sRGB \| linearRGB \| inherit | linearRGB | filter primitives | yes | | visual | yes |
| 'color-profile' | auto \| sRGB \| <name> \| <uri> \| inherit | auto | 'image' elements that refer to raster images | yes | | visual | yes |
| 'color-rendering' | auto \| optimizeSpeed \| optimizeQuality \| inherit | auto | container elements, graphics elements and 'animateColor' | yes | | visual | yes |
| 'cursor' | [ [<uri> ,]* [ auto \| crosshair \| default \| pointer \| move \| e-resize \| ne-resize \| nw-resize \| n-resize \| se-resize \| sw-resize \| s-resize \| w-resize\| text \| wait \| help ] ] \| inherit | auto | container elements and graphics elements | yes | | visual, interactive | yes |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 'direction' | ltr \| rtl \| inherit | ltr | text content elements | yes | | visual | no |
| 'display' | inline \| block \| list-item \| run-in \| compact \| marker \| table \| inline-table \| table-row-group \| table-header-group \| table-footer-group \| table-row \| table-column-group \| table-column \| table-cell \| table-caption \| none \| inherit | inline | **'svg'**, **'g'**, **'switch'**, **'a'**, **'foreignObject'**, graphics elements (including the **'text'** element) and text sub-elements (i.e., **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'**) | no | | all | yes |
| 'dominant-baseline' | auto \| use-script \| no-change \| reset-size \| alphabetic \| hanging \| ideographic \| mathematical \| central \| middle \| text-after-edge \| text-before-edge \| text-top \| text-bottom \| inherit | auto | text content elements | no | | visual | yes |
| 'enable-background' | accumulate \| new [ <x> <y> <width> <height> ] \| inherit | accumulate | container elements | no | | visual | no |
| 'fill' | <paint> (See Specifying paint) | black | shapes and text content elements | yes | | visual | yes |
| 'fill-opacity' | <opacity-value> \| inherit | 1 | shapes and text content elements | yes | | visual | yes |
| 'fill-rule' | nonzero \| evenodd \| inherit | nonzero | shapes and text content elements | yes | | visual | yes |
| 'filter' | <uri> \| none \| inherit | none | container elements and graphics elements | no | | visual | yes |
| 'flood-color' | currentColor \| <color> [icc-color(<name>[,<icccolorvalue>]*)] \| inherit | black | **'feFlood'** elements | no | | visual | yes |
| 'flood-opacity' | <alphavalue> \| inherit | 1 | **'feFlood'** elements | no | | visual | yes |
| 'font' | [ [ 'font-style' \|\| 'font-variant' \|\| 'font-weight' ]? 'font-size' [ / 'line-height' ]? 'font-family' ] \| caption \| icon \| menu \| message-box \| small-caption \| status-bar \| inherit | see individual properties | text content elements | yes | allowed on 'font-size' and 'line-height' ('line-height' same as 'font-size' in SVG) | visual | yes (non-additive, 'set' and 'animate' elements only) |
| 'font-family' | [[ <family-name> \| <generic-family> ],]* [<family-name> \| <generic-family>] \| inherit | depends on user agent | text content elements | yes | | visual | yes |
| 'font-size' | <absolute-size> \| <relative-size> \| <length> \| <percentage> \| inherit | medium | text content elements | yes, the computed value is inherited | refer to parent element's font size | visual | yes |
| 'font-size-adjust' | <number> \| none \| inherit | none | text content elements | yes | | visual | yes |
| 'font-stretch' | normal \| wider \| narrower \| ultra-condensed \| extra-condensed \| condensed \| semi-condensed \| semi-expanded \| expanded \| extra-expanded \| ultra-expanded \| inherit | normal | text content elements | yes | | visual | yes |
| 'font-style' | normal \| italic \| oblique \| inherit | normal | text content elements | yes | | visual | yes |
| 'font-variant' | normal \| small-caps \| inherit | normal | text content elements | yes | | visual | yes |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 'font-weight' | normal \| bold \| bolder \| lighter \| 100 \| 200 \| 300 \| 400 \| 500 \| 600 \| 700 \| 800 \| 900 \| inherit | normal | text content elements | yes | | visual | yes |
| 'glyph-orientation-horizontal' | <angle> \| inherit | 0deg | text content elements | yes | | visual | no |
| 'glyph-orientation-vertical' | auto \| <angle> \| inherit | auto | text content elements | yes | | visual | no |
| 'image-rendering' | auto \| optimizeSpeed \| optimizeQuality \| inherit | auto | images | yes | | visual | yes |
| 'kerning' | auto \| <length> \| inherit | auto | text content elements | yes | | visual | yes |
| 'letter-spacing' | normal \| <length> \| inherit | normal | text content elements | yes | | visual | yes |
| 'lighting-color' | currentColor \| <color> [icc-color(<name>[,<icccolorvalue>]*)] \| inherit | white | **'feDiffuseLighting'** and **'feSpecularLighting'** elements | no | | visual | yes |
| 'marker' | see individual properties | see individual properties | **'path'**, **'line'**, **'polyline'** and **'polygon'** elements | yes | | visual | yes |
| 'marker-end' 'marker-mid' 'marker-start' | none \| inherit \| <uri> | none | **'path'**, **'line'**, **'polyline'** and **'polygon'** elements | yes | | visual | yes |
| 'mask' | <uri> \| none \| inherit | none | container elements and graphics elements | no | | visual | yes |
| 'opacity' | <alphavalue> \| inherit | 1 | container elements and graphics elements | no | | visual | yes |
| 'overflow' | visible \| hidden \| scroll \| auto \| inherit | see prose | elements which establish a new viewport, **'pattern'** elements and **'marker'** elements | no | | visual | yes |
| 'pointer-events' | visiblePainted \| visibleFill \| visibleStroke \| visible \| painted \| fill \| stroke \| all \| none \| inherit | visiblePainted | graphics elements | yes | | visual | yes |
| 'shape-rendering' | auto \| optimizeSpeed \| crispEdges \| geometricPrecision \| inherit | auto | shapes | yes | | visual | yes |
| 'stop-color' | currentColor \| <color> [icc-color(<name>[,<icccolorvalue>]*)] \| inherit | black | **'stop'** elements | no | | visual | yes |
| 'stop-opacity' | <alphavalue> \| inherit | 1 | **'stop'** elements | no | | visual | yes |
| 'stroke' | <paint> (See Specifying paint) | none | shapes and text content elements | yes | | visual | yes |
| 'stroke-dasharray' | none \| <dasharray> \| inherit | none | shapes and text content elements | yes | | visual | |
| 'stroke-dashoffset' | <length> \| inherit | 0 | shapes and text content elements | yes | see prose | visual | yes |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 'stroke-linecap' | butt \| round \| square \| inherit | butt | shapes and text content elements | yes | | visual | yes |
| 'stroke-linejoin' | miter \| round \| bevel \| inherit | miter | shapes and text content elements | yes | | visual | yes |
| 'stroke-miterlimit' | <miterlimit> \| inherit | 4 | shapes and text content elements | yes | | visual | yes |
| 'stroke-opacity' | <opacity-value> \| inherit | 1 | shapes and text content elements | yes | | visual | yes |
| 'stroke-width' | <length> \| inherit | 1 | shapes and text content elements | yes | | visual | yes |
| 'text-anchor' | start \| middle \| end \| inherit | start | text content elements | yes | | visual | yes |
| 'text-decoration' | none \| [ underline \|\| overline \|\| line-through \|\| blink ] \| inherit | none | text content elements | no (see prose) | | visual | yes |
| 'text-rendering' | auto \| optimizeSpeed \| optimizeLegibility \| geometricPrecision \| inherit | auto | **'text'** elements | yes | | visual | yes |
| 'unicode-bidi' | normal \| embed \| bidi-override \| inherit | normal | text content elements | no | | visual | no |
| 'visibility' | visible \| hidden \| collapse \| inherit | visible | graphics elements (including the **'text'** element) and text sub-elements (i.e., **'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** and **'a'**) | yes | | visual | yes |
| 'word-spacing' | normal \| <length> \| inherit | normal | text content elements | yes | | visual | yes |
| 'writing-mode' | lr-tb \| rl-tb \| tb-rl \| lr \| rl \| tb \| inherit | lr-tb | **'text'** elements | yes | | visual | no |

04 September 2001

# Appendix O: Index

---