W3C Proposed Recommendation

Scalable Vector Graphics (SVG) 1.1 (Second Edition)

W3C Proposed Recommendation *09 June 2011*

This version:
http://www.w3.org/TR/2011/PR-SVG11-20110609/

Latest version:
http://www.w3.org/TR/SVG11/

Previous version:
http://www.w3.org/TR/2011/WD-SVG11-20110512/

Public comments:
www-svg@w3.org (archive)

Editors:
Erik Dahlström, Opera Software · ed@opera.com
Patrick Dengler, Microsoft Corporation · patd@microsoft.com
Anthony Grasso, Canon Inc. · anthony.grasso@cisra.canon.com.au
Chris Lilley, W3C · chris@w3.org
Cameron McCormack, Mozilla Corporation · cam@mcc.id.au
Doug Schepers, W3C · schepers@w3.org
Jonathan Watt, Mozilla Corporation · jwatt@jwatt.org

Jon Ferraiolo, ex Adobe Systems · jferrai@us.ibm.com — Versions 1.0 and 1.1 First Edition; until 10 May 2006
藤沢 淳 (FUJISAWA Jun), Canon Inc. · fujisawa.jun@canon.co.jp — Version 1.1 First Edition
Dean Jackson, ex W3C · dean@w3.org — Version 1.1 First Edition; until February 2007

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: a single-page version and a PDF. See also **translations**, noting that the English version of this specification is the only normative version.

Abstract

This specification defines the features and syntax for Scalable Vector Graphics (SVG) Version 1.1, a modularized 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. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is the 09 June 2011 Proposed Recommendation of the SVG 1.1 Second Edition specification. The Second Edition incorporates a number of corrections that were published as errata against the First Edition, as well as numerous other changes that help make the specification more readable and unambiguous. The Changes appendix lists all of the changes that were made since the first Last Call Working Draft publication of the Second Edition. For the changes made between the First Edition and the Second Edition Working Draft, see that document's Changes appendix.

This specification moved from Last Call phase to Proposed Recommendation phase without a Candidate Recommendation phase because of sufficient implementation experience and wide interoperability.

Comments on this Proposed Recommendation are welcome. Please note however that major corrections against the specification will be published as errata, and subsequently will be incorporated into future editions of SVG 1.1 or into SVG 2.0. Comments can be sent to www-svg@w3.org, the public email list for issues related to vector graphics on the Web. This list is archived and senders must agree to have their message publicly archived from their first posting. To subscribe send an email to www-svg-request@w3.org with the word `subscribe` in the subject line.

W3C Advisory Committee Members are invited to send formal review comments on this Proposed Recommendation to the W3C Team until 07 July 2011. Members of the W3C Advisory Committee will find the appropriate review form for this document by consulting their list of current WBS questionnaires.

The W3C SVG Working Group has released an expanded test suite for SVG 1.1 along with an implementation report. This test suite will continue to be updated with new tests to improve interoperability even after Recommendation phase.

This document has been produced by the W3C SVG Working Group as part of the Graphics Activity within the W3C Interaction Domain. The goals of the W3C SVG Working Group are discussed in the W3C SVG Charter. The W3C SVG Working Group maintains a public Web page, http://www.w3.org/Graphics/SVG/, that contains further background information. The authors of this document are the SVG Working Group participants.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/. W3C publications may be updated, replaced, or obsoleted by other documents at any time.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

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.

Acknowledgments

The SVG Working Group would like to thank the following people for contributing to this specification by raising issues that resulted in errata that were folded in to this document: Tavmjong Bah, Brian Birtles, Tolga Capin, Alex Danilo, Thomas DeWeese, Alexey Feldgendler, Vincent Hardy, Ian Hickson, Olaf Hoffmann, Daniel Holbert, Oliver Hunt, Anne van Kesteren, Takeshi Kurosawa, Paul Libbrecht, Robert Longson, Helder Magalhães, Robert O'Callahan, Olli Pettay, Antoine Quint, Kalle Raita, Tim Rowley, Peter Sorotokin, Henry S. Thompson, Jasper van de Gronde, Mohamed Zergaoui, Boris Zbarsky.

In addition, the SVG Working Group would like to acknowledge the contributions of the editors and authors of SVG 1.0 and SVG 1.1 (First Edition), as much of the text in this document derives from these earlier versions of the SVG specification.

Finally, the SVG Working Group would like to acknowledge the great many people outside of the SVG Working Group who help with the process of developing the SVG specifications. These people are too numerous to list individually. They include but are not limited to the early implementers of the SVG 1.0 and 1.1 languages (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 1.1 is truly a cooperative effort between the SVG Working Group, the rest of the W3C, and the public and benefits greatly from the pioneering work of early implementers and content developers, feedback from the public, and help from the W3C team.

Table of Contents

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.

SVG 1.1 is a modularization of SVG 1.0 [SVG10]. See the Document Type Definition appendix for details on how the DTD is structured to allow profiling and composition with other XML languages.

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

The MIME type for SVG is `"image/svg+xml"` (see *XML Media Types* [RFC3023]). The registration of this MIME type is in progress at the W3C.

It is recommended that SVG files have the extension `".svg"` (all lowercase) on all platforms. It is recommended that gzip-compressed [RFC1952] 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 1.1 namespace, public identifier and system identifier:

SVG Namespace:
http://www.w3.org/2000/svg
Public Identifier for SVG 1.1:
PUBLIC "-//W3C//DTD SVG 1.1//EN"
System Identifier for the SVG 1.1 Recommendation:
http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd

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

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Note that DTD listed in the System Identifier is a modularized DTD (i.e. its contents are spread over multiple files), which means that a validator may have to fetch the multiple modules in order to validate. For that reason, there is a single flattened DTD available that corresponds to the SVG 1.1 modularized DTD. It can be found at http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat.dtd.

While a DTD is provided in this specification, the use of DTDs for validating XML documents is known to be problematic. In particular, DTDs do not handle namespaces gracefully. It is *not* recommended that a DOCTYPE declaration be included in SVG documents.

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 IRI referencing and requires support for base IRI specifications defined in *XML Base* [XML-BASE].
- SVG content can be styled by either CSS (see *Cascading Style Sheets (CSS) level 2* [CSS2]) or XSLT (see *XSL*

Transformations (XSLT) Version 1.0 [XSLT] and *XSL Transformations (XSLT) Version 2.0* [XSLT2]). See Styling with CSS and Styling with XSL for details.

- 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 DOM Level 2, including the CSS object model and event handling [DOM2] [DOM2STYLE] [DOM2EVENTS].
- SVG incorporates some features and approaches that are part of the *Synchronized Multimedia Integration Language (SMIL) 3.0 Specification* [SMIL], 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) 3.0 Specification* [SMIL]. 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 SMIL to use animated or static SVG content as media components.
- SVG attempts to achieve maximum compatibility with both *HTML 4* [HTML4] and XHTML™ 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. Also, see Accessibility Support.

In environments which support DOM 2 Core [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 *Key words for use in RFCs to Indicate Requirement Levels* [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

animation element · An animation element is an element that can be used to animate the attribute or property value of another element. The following elements are animation elements: 'animateColor', 'animateMotion', 'animateTransform', 'animate' and 'set'.

animation event attribute · An animation event attribute is an event attribute that specifies script to run for a particular animation-related event. See Animation event attributes. The animation event attributes are 'onbegin', 'onend', 'onload' and 'onrepeat'.

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

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: 'a', 'defs', 'glyph', 'g', 'marker', 'mask', 'missing-glyph', 'pattern', 'svg', 'switch' and 'symbol'.

conditional processing attribute · A conditional processing attribute is one that controls whether or not the element on which it appears is processed. Most elements, but not all, may have conditional processing attributes specified on them. See Conditional processing for details. The conditional processing attributes defined in SVG 1.1 are 'requiredExtensions', 'requiredFeatures' and 'systemLanguage'.

core attributes · The core attributes are those attributes that can be specified on any SVG element. See Common attributes. The core attributes are 'id', 'xml:base', 'xml:lang' and 'xml:space'.

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 trans-*

formation matrix (CTM) defines the mapping from the user coordinate system into the viewport coordinate system. See Coordinate system transformations.

descriptive element · An element which provides supplementary descriptive information about its parent. Specifically, the following elements are descriptive elements: 'desc', 'metadata' and 'title'.

document event attribute · A document event attribute is an event attribute that specifies script to run for a particular document-wide event. See Document-level event attributes. The document event attributes are 'onabort', 'onerror', 'onresize', 'onscroll', 'onunload' and 'onzoom'.

event attribute · An event attribute is one that specifies some script to run when an event of a certain type is dispatched to the element on which the attribute is specified. See Event attributes.

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

filter primitive attributes · The filter primitive attributes is the set of attributes that are common to all filter primitive elements. They are 'height', 'result', 'width', 'x' and 'y'.

filter primitive element · A filter primitive element is one that can be used as a child of a 'filter' element to specify a node in the filter graph. The following elements are the filter primitive elements defined in SVG 1.1: 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile' and 'feTurbulence'.

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.

gradient element · A gradient element is one that defines a gradient paint server. SVG 1.1 defines the following gradient elements: 'linearGradient' and 'radialGradient'.

graphical event attribute · A graphical event attribute is an event attribute that specifies script to run for a particular user interaction event. See Event attributes on graphics and container elements. The graphical event attributes are 'onactivate', 'onclick', 'onfocusin', 'onfocusout', 'onload', 'onmousedown', 'onmousemove', 'onmouseout', 'onmouseover' and 'onmouseup'.

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

hit-testing · The process of determining whether a pointer intersects a given graphics element. Hit-testing is used in determining which element to dispatch a mouse event to, which might be done in response to the user moving the pointing device, or by changes in the position, shape and other attributes of elements in the document. Hit-testing is also known as *hit detection* or *picking*. See hit-testing and processing order for user interface events and the definition of the '**pointer-events**' property.

IRI reference · An IRI reference is an Internationalized Resource Identifier with an optional fragment identifier, as defined in *Internationalized Resource Identifiers* [RFC3987]. An IRI reference serves as a reference to a resource or (with a fragment identifier) to a secondary resource. See References and the 'defs' element.

light source element · A light source element is one that can specify light source information for an 'feDiffuseLighting' or 'feSpecularLighting' element. The following light source elements are defined in SVG 1.1: 'feDistantLight', 'fePointLight' and 'feSpotLight'.

local IRI reference · An Internationalized Resource Identifier [RFC3987] that does not include an <absoluteIRI> or <relativeIRI> 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 IRI reference · An Internationalized Resource Identifier [RFC3987] that includes an <absoluteIRI> or <relativeIRI> and thus (usually) represents a reference to a different document or an element within a different document. See References and the 'defs' element.

outermost svg element · The furthest 'svg' ancestor element that remains in the current SVG document fragment.

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. Note that although any property may be *specified* on any element, not all properties will *apply to* (affect the rendering of) a given element. The definition of each property states to what set of elements it applies.

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.

rootmost 'svg' element · The rootmost 'svg' element is the furthest 'svg' ancestor element that does not exit an SVG context. See also SVG document fragment.

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

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

structural element · The structural elements are those which define the primary structure of an SVG document. Specifically, the following elements are structural elements: 'defs', 'g', 'svg', 'symbol' and 'use'.

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 context · An SVG context is a document fragment where all elements within the fragment must be subject to processing by an SVG user agent according to the rules in this specification.

If SVG content is embedded inline within parent XML (such as XHTML), the SVG context does not include the ancestors above the rootmost 'svg' element. If the SVG content contains any 'foreignObject' elements which in turn contain non-SVG content, the SVG context does not include the contents of the 'foreignObject' elements.

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 user agent · An SVG user agent is a user agent that is able to retrieve and render SVG content.

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 · A text content element is an SVG element that causes a text string to be rendered onto the canvas. The SVG 1.1 text content elements are the following: 'altGlyph', 'textPath', 'text', 'tref' and 'tspan'

text content child element · A text content child element is a text content element that is allowed as a descendant of another text content element. In SVG 1.1, the text content child elements are the following: 'altGlyph', 'textPath', 'tref' and 'tspan'

text content block element · A text content block element is a text content element that serves as a standalone element for a unit of text, and which may optionally contain certain child text content elements (e.g. 'tspan'). .

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.

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.

XLink attributes · The XLink attributes are the seven attributes defined in the XML Linking Language specification [XLINK], which are used on various SVG elements that can reference resources. The most import XLink attribute is '**xlink:href**', whose definition can be found on each element that allows it. The remaining XLink attributes are '**xlink:type**', '**xlink:role**', '**xlink:arcrole**', '**xlink:title**', '**xlink:show**' and '**xlink:actuate**'.

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 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 CSS or XSL property

When a user agent supports CSS-styled XML content [CSS2] or XSL [XSL] 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 ([CSS2], sections 14.2.1 and 12.6.2) that are included in both CSS and XSL.

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 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' element (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 the '**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.

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" version="1.1">
      <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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" version="1.1"
     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>
```

'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

Categories: 'svg'

Container element, structural element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'
- 'cursor'
- 'filter'
- 'font'
- 'font-face'
- 'foreignObject'
- 'image'
- 'marker'
- 'mask'

'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:

conditional processing attributes
core attributes
document event attributes
graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'x'
'y'
'width'
'height'
'viewBox'
'preserveAspectRatio'
'zoomAndPan'
'version'
'baseProfile'
'contentScriptType'
'contentStyleType'
'x'
'y'
'width'
'height'
'version'
'baseProfile'

DOM Interfaces:

SVGSVGElement

Attribute definitions:

`version` = "**<number>**"

Indicates the SVG language version to which this document fragment conforms.

In SVG 1.0 [SVG10], this attribute was fixed to the value '**1.0**'. For SVG 1.1, the attribute should have the value '**1.1**'.
Animatable: no.

`baseProfile` = **profile-name**
Describes the minimum SVG language profile that the author believes is necessary to correctly render the content. The attribute does not specify any processing restrictions; It can be considered metadata. For example, the value of the attribute could be used by an authoring tool to warn the user when they are modifying the document beyond the scope of the specified base profile. Each SVG profile should define the text that is appropriate for this attribute.
If the attribute is not specified, the effect is as if a value of '**none**' were specified.
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.

`preserveAspectRatio` = "**[defer] <align> [<meetOrSlice>]**"
See 'preserveAspectRatio'.

If the attribute is not specified, then the effect is as if a value of '**xMidYMid meet**' were specified.
Animatable: *yes*.

`contentScriptType` = "***content-type***"
See 'contentScriptType'.

`contentStyleType` = "***content-type***"
See 'contentStyleType'.

`zoomAndPan` = "**disable | magnify**"
See 'zoomAndPan'.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1" width="5cm" height="5cm">
  <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>
```

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1" width="4in" height="3in">
  <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

Categories: 'g'

Container element, structural element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'
- 'cursor'
- 'filter'
- 'font'
- 'font-face'
- 'foreignObject'
- 'image'
- 'marker'
- 'mask'
- 'pattern'
- 'script'
- 'style'
- 'switch'
- 'text'
- 'view'

Attributes:

conditional processing attributes

core attributes
graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'transform'

DOM Interfaces:
SVGGElement

5.3 Defining content for reuse, and the 'defs' element

5.3.1 Overview

SVG allows graphical objects to be defined for later reuse. To do this, it makes extensive use of IRI references [RFC3987] to these 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)"/>
```

Some types of element, such as gradients, will not by themselves produce a graphical result. They can therefore be placed anywhere convenient. However, sometimes it is desired to define a graphical object and prevent it from being directly rendered. it is only there to be referenced elsewhere. To do this, and to allow convenient grouping defined content, SVG provides the 'defs' element.

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 The 'defs' element

Categories: 'defs'
Container element, structural element

Content model:
Any number of the following elements, in any order:

animation elements
descriptive elements
shape elements
structural elements
gradient elements
'a'
'altGlyphDef'
'clipPath'
'color-profile'
'cursor'
'filter'
'font'
'font-face'
'foreignObject'
'image'
'marker'
'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:

conditional processing attributes
core attributes
graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'transform'

DOM Interfaces:

SVGDefsElement

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.

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 IRI 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 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="3cm"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

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

Categories: 'desc'

Descriptive element

Content model:

Any elements or character data.

Attributes:

core attributes

'class'

'style'

DOM Interfaces:

SVGDescElement

Categories: ‘title’

Descriptive element

Content model:

Any elements or character data.

Attributes:

core attributes

‘class’

‘style’

DOM Interfaces:

SVGTitleElement

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.

In conforming SVG document fragments, any ‘title’ element should be the first child element of its parent. Note that those implementations that do use ‘title’ to display a tooltip often will only do so if the ‘title’ is indeed the first child element of its parent.

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/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1" width="4in" height="3in">
  <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 xmlns="http://www.w3.org/2000/svg"
     version="1.1" width="4in" height="3in">
  <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 ([XML10], section 3.2.2) 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

Categories: 'symbol'

Container element, structural element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'

'cursor'
'filter'
'font'
'font-face'
'foreignObject'
'image'
'marker'
'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:
core attributes
graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'preserveAspectRatio'
'viewBox'

DOM Interfaces:
SVGSymbolElement

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.

'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

Categories: 'use'

Graphics element, graphics referencing element, structural element

Content model:

Any number of the following elements, in any order:

animation elements
descriptive elements

Attributes:

core attributes
conditional processing attributes
graphical event attributes
presentation attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'transform'
'x'
'y'
'width'
'height'
'xlink:href'

DOM Interfaces:

SVGUseElement

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], chapter 6) 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 event handling for the non-exposed tree works as if the referenced element had been textually included as a deeply cloned child of the 'use' element, except that events are dispatched to the SVGElementInstance objects. The event's target and currentTarget attributes are set to the SVGElementInstance that corresponds to the target and current target elements in the referenced subtree. An event propagates through the exposed and non-exposed portions of the tree in the same manner as it would in the regular document tree: first going from the root element to the 'use' element and then through non-exposed tree elements in the capture phase, followed by the target phase at the target of the event, then bubbling back through non-exposed tree to the use element and then back through regular tree to the root element in bubbling phase.

An element and all its corresponding SVGElementInstance objects share an event listener list. The currentTarget attribute of the event can be used to determine through which object an event listener was invoked.

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 instances 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 val-

ues 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**:

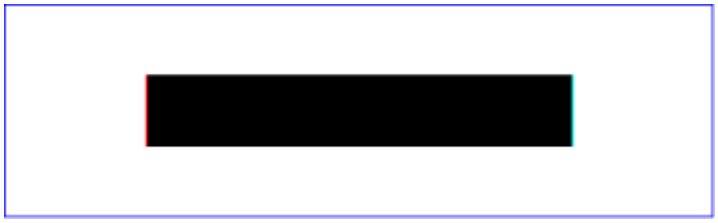
  In the generated content, the 'use' will be replaced by 'g', where all attributes from the 'use' element except for 'x', 'y', 'width', 'height' and 'xlink:href' are transferred to the generated 'g' element. An additional transformation **translate(x,y)** is appended to the end (i.e., right-side) of the '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], chapter 6). 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30" version="1.1"
     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



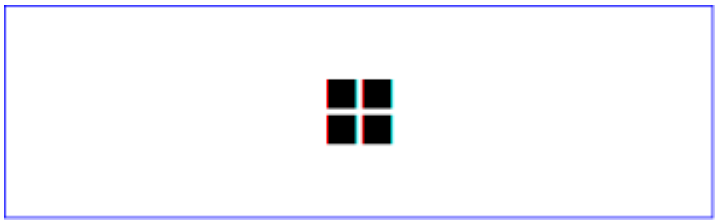
The visual effect would be equivalent to the following document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

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

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30" version="1.1"
     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



The visual effect would be equivalent to the following document:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

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

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30" version="1.1"
     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

The visual effect would be equivalent to the following document:

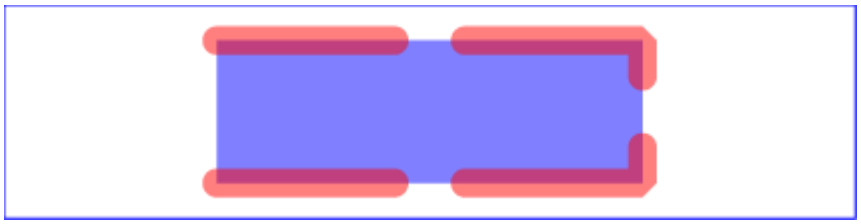
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example Use03-GeneratedContent - 'use' with 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>
```

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3cm" viewBox="0 0 1200 300" version="1.1"
     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



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3cm" viewBox="0 0 1200 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

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. However, a set of references that directly or indirectly reference a element to create a circular dependency is an error, as described in References and the 'defs' element.

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. A negative value is an error (see Error processing). A value of zero disables rendering of this element.
Animatable: yes.

`height = "<length>"`

The height of the rectangular region into which the referenced element is placed. A negative value is an error (see Error processing). A value of zero disables rendering of this element.
Animatable: yes.

`xlink:href = "<iri>"`

A IRI reference to an element/fragment within an SVG document.
Animatable: yes.

5.7 The 'image' element

Categories: 'image'

Graphics element, graphics referencing element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements

Attributes:

- core attributes
- conditional processing attributes
- graphical event attributes
- xlink attributes
- presentation attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'preserveAspectRatio'
- 'transform'
- 'x'
- 'y'
- 'width'
- 'height'
- 'xlink:href'

DOM Interfaces:

SVGImageElement

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.

An 'image' element establishes a new viewport for the referenced file as described in Establishing a new viewport. The bounds for the new viewport are defined by attributes 'x', 'y', 'width' and 'height'. The placement and scaling of the referenced image are controlled by the 'preserveAspectRatio' attribute on the 'image' element.

When an 'image' element references an SVG image, the '**clip**' and '**overflow**' properties on the root element in the referenced SVG image are ignored (in the same manner as the 'x', 'y', 'width' and 'height' attributes are ignored). Unless the value of 'preserveAspectRatio' on the 'image' element starts with 'defer', the 'preserveAspectRatio' attribute on the root element in the referenced SVG image is also ignored (see 'preserveAspectRatio' for details). Instead, the 'preserveAspectRatio' attribute on the referencing 'image' element defines how the SVG image content is fitted into the viewport and the '**clip**' and '**overflow**' properties on the 'image' element define how the SVG image content is clipped (or not) relative to the viewport.

The value of the 'viewBox' attribute to use when evaluating the 'preserveAspectRatio' attribute is defined by the referenced content. For content that clearly identifies a viewBox (e.g. an SVG file with the 'viewBox' attribute on the outermost svg element) that value should be used. For most raster content (PNG, JPEG) the bounds of the image should be used (i.e. the 'image' element has an implicit 'viewBox' of '**0 0 raster-image-width raster-image-height**'). Where no value is readily available (e.g. an SVG file with no 'viewBox' attribute on the outermost svg element) the 'preserveAspectRatio' attribute is ignored, and only the translation due to the 'x' & 'y' attributes of the viewport is used to display the content.

For example, if the image element referenced a PNG or JPEG and **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 as defined by the attributes 'x', 'y', 'width' and 'height' on the 'image' element. If the value of 'preserveAspectRatio' was '**none**' then aspect ratio of the image 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').

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.

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` = "**<iri>**"

A IRI reference.
Animatable: yes.

`preserveAspectRatio` = "**[defer] <align> [<meetOrSlice>]**"

See 'preserveAspectRatio'.

If attribute 'preserveAspectRatio' is not specified, then the effect is as if a value of **xMidYMid meet** were specified.

Animatable: yes.

An example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="3in" version="1.1"
     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 'systemLan-

guage' to provide an ability to specify alternate viewing depending on the capabilities of a given user agent or the user's language.

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.

Similar to the '**display**' property, conditional processing attributes only affect the direct rendering of elements and do not prevent elements from being successfully referenced by other elements (such as via a 'use').

In consequence:

- 'requiredFeatures', 'requiredExtensions' and 'systemLanguage' attributes affect 'a', 'altGlyph', 'foreignObject', 'textPath', 'tref', and 'tspan' elements.
- 'requiredFeatures', 'requiredExtensions' and 'systemLanguage' attributes will have no effect on 'mask', 'clipPath', and 'pattern' elements.
- 'requiredFeatures', 'requiredExtensions' and 'systemLanguage' attributes do not apply to the 'defs', and 'cursor' elements because they are not part of the rendering tree.
- 'requiredFeatures', 'requiredExtensions' and 'systemLanguage' attributes affect 'animate', 'animateColor', 'animateMotion', 'animateTransform', and 'set' elements. If the conditional statement on these animation elements fails, the animation will never be triggered.

5.8.2 The 'switch' element

Categories: 'switch'

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- 'a'
- 'foreignObject'
- 'g'
- 'image'
- 'svg'
- 'switch'
- 'text'
- 'use'

Attributes:

conditional processing attributes

core attributes

graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'transform'

DOM Interfaces:
SVGSwitchElement

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.

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 the Feature String appendix 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.

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 an IRI reference.

Definition of `requiredExtensions`:

`requiredExtensions` = ***list-of-extensions***

The value is a list of IRI 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 IRI 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 IRI 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 BCP 47 [BCP47].

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.

5.8.6 Applicability of test attributes

The following list describes the applicability of the test attributes to the elements that do not directly produce rendering.

- the test attributes do not effect the 'mask', 'clipPath', 'linearGradient', 'radialGradient' and 'pattern' elements. The test attributes on a referenced element do not affect the rendering of the referencing element.
- the test attributes do not effect the 'defs', and 'cursor' elements as they are not part of the rendering tree.
- an animation element ('animate', 'animateMotion', 'animateTransform', 'animateColor' and 'set') will never be triggered if it has a test attribute that evaluates to false.

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, the document's SVGLoad event is not fired and the an-

imation timeline does not begin 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 an IRI 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 will have the effect of disabling progressive display of the contents of that container, if that container includes elements that reference external resources, tools that generate SVG content are cautioned against 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 specifically 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:

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` = "**<iri>**"
: Specifies a base IRI other than the base IRI 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'.

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

5.11.1 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.

```
interface SVGDocument : Document,
                        DocumentEvent {
  readonly attribute DOMString title;
  readonly attribute DOMString referrer;
  readonly attribute DOMString domain;
  readonly attribute DOMString URL;
  readonly attribute SVGSVGElement rootElement;
};
```

Attributes:

- **title** (readonly DOMString)

  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>)

- **referrer** (readonly DOMString)

  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).

- **domain** (readonly DOMString)

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

- **URL** (readonly DOMString)

  The complete URI of the document.

- **rootElement** (readonly SVGSVGElement)

  The root 'svg' in the document hierarchy.

5.11.2 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 *DOM Level 2 Style* [DOM2STYLE].

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

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
           attribute DOMString contentScriptType setraises(DOMException);
           attribute DOMString contentStyleType setraises(DOMException);
```

```
  readonly attribute SVGRect viewport;
  readonly attribute float pixelUnitToMillimeterX;
  readonly attribute float pixelUnitToMillimeterY;
  readonly attribute float screenPixelToMillimeterX;
  readonly attribute float screenPixelToMillimeterY;
  readonly attribute boolean useCurrentView;
  readonly attribute SVGViewSpec currentView;
           attribute float currentScale;
  readonly attribute SVGPoint currentTranslate;

  unsigned long suspendRedraw(in unsigned long maxWaitMilliseconds);
  void unsuspendRedraw(in unsigned long suspendHandleID);
  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:

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'svg' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'svg' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'svg' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'svg' element.

- **contentScriptType** (DOMString)

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

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **contentStyleType** (DOMString)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **viewport** (readonly SVGRect)

  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 SVGRect object is read only.

- **pixelUnitToMillimeterX** (readonly float)

  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.

- **pixelUnitToMillimeterY** (readonly float)

  Corresponding size of a pixel unit along the y-axis of the viewport.

- **screenPixelToMillimeterX** (readonly float)

  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.

- **screenPixelToMillimeterY** (readonly float)

  Corresponding size of a screen pixel along the y-axis of the viewport.

- **useCurrentView** (readonly boolean)

  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 'viewBox') or to a "custom" view (i.e., a hyperlink into a particular 'view' or other element - see Linking into SVG content: IRI 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.

- **currentView** (readonly SVGViewSpec)

  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 read only.

- **currentScale** (float)

  On an outermost svg element, 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).
When accessed on an 'svg' element that is not an outermost svg element, it is undefined what behavior this attribute has.

- **currentTranslate** (readonly SVGPoint)

  On an outermost svg element, the corresponding translation factor that takes into account user "magnification".
  When accessed on an 'svg' element that is not an outermost svg element, it is undefined what behavior this attribute has.

Operations:

- unsigned long **suspendRedraw**(in unsigned long *maxWaitMilliseconds*)

  Takes a time-out value which indicates that redraw shall not occur until:

  1. the corresponding unsuspendRedraw() call has been made,
  2. an unsuspendRedrawAll() call has been made, or
  3. its timer has timed out.

  In environments that do not support interactivity (e.g., print media), then redraw shall not be suspended. Calls to suspendRedraw() and unsuspendRedraw() should, but need not be, made in balanced pairs.

  To suspend redraw actions as a collection of SVG DOM changes occur, precede the changes to the SVG DOM with a method call similar to:

  ```
  suspendHandleID = suspendRedraw(maxWaitMilliseconds);
  ```

  and follow the changes with a method call similar to:

  ```
  unsuspendRedraw(suspendHandleID);
  ```

  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

  - unsigned long *maxWaitMilliseconds*
    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.

Returns

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.

- void **unsuspendRedraw**(in unsigned long *suspendHandleID*)

  Cancels a specified suspendRedraw() by providing a unique suspend handle ID that was returned by a previous suspendRedraw() call.

  Parameters

  - unsigned long *suspendHandleID*
    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(). If an invalid handle ID value is provided then the request to unsuspendRedraw() is silently ignored.

- void **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.

- void **forceRedraw**()

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

- void **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.

- void **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.

- boolean **animationsPaused**()

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

  Returns

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

- float **getCurrentTime**()

  Returns the current time in seconds relative to the start time for the current SVG document fragment. If getCurrentTime is called before the document timeline has begun (for example, by script running in a 'script' element before the document's SVGLoad event is dispatched), then 0 is returned.

  Returns
  The current time in seconds, or 0 if the document timeline has not yet begun.

- void **setCurrentTime**(in float *seconds*)

  Adjusts the clock for this SVG document fragment, establishing a new current time. If setCurrentTime is called before the document timeline has begun (for example, by script running in a 'script' element before the document's SVGLoad event is dispatched), then the value of *seconds* in the last invocation of the method gives the time that the document will seek to once the document timeline has begun.

  Parameters

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

- NodeList **getIntersectionList**(in SVGRect *rect*, in SVGElement *referenceElement*)

  Returns the list of graphics elements whose rendered content intersects the supplied rectangle. Each candidate graphics element is to be considered a match only if the same graphics element can be a target of pointer events as defined in '**pointer-events**' processing.

  Parameters

  - SVGRect *rect*
    The test rectangle. The values are in the initial coordinate system for the current 'svg' element.
  - SVGElement *referenceElement*
    If not null, then any intersected element that doesn't have the referenceElement as ancestor must not be included in the returned NodeList.

  Returns
  A list of Elements whose content intersects the supplied rectangle. This NodeList must be implemented identically to the NodeList interface as defined in DOM Level 2 Core ([DOM2], section 1.2) with the exception that the interface is not live.

- NodeList **getEnclosureList**(in SVGRect *rect*, in SVGElement *referenceElement*)

  Returns the list of graphics elements whose rendered content is entirely contained within the supplied rectangle. Each candidate graphics element is to be considered a match only if the same graphics element can be a target of pointer events as defined in '**pointer-events**' processing.

  Parameters

  - SVGRect *rect*
    The test rectangle. The values are in the initial coordinate system for the current 'svg' element.
  - SVGElement *referenceElement*
    If not null, then any intersected element that doesn't have the referenceElement as ancestor must not be included in the returned NodeList.

  Returns

  A list of Elements whose content is enclosed by the supplied rectangle. This NodeList must be implemented identically to the NodeList interface as defined in DOM Level 2 Core ([DOM2], section 1.2) with the exception that the interface is not live.

- boolean **checkIntersection**(in SVGElement *element*, in SVGRect *rect*)

  Returns true if the rendered content of the given element intersects the supplied rectangle. Each candidate graphics element is to be considered a match only if the same graphics element can be a target of pointer events as defined in '**pointer-events**' processing.

  Parameters

  - SVGElement *element*
    The element on which to perform the given test.
  - SVGRect *rect*
    The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

  Returns

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

- boolean **checkEnclosure**(in SVGElement *element*, in SVGRect *rect*)

  Returns true if the rendered content of the given element is entirely contained within the supplied rectangle. Each candidate graphics element is to be considered a match only if the same graphics element can be a target of pointer events as defined in '**pointer-events**' processing.

Parameters

- SVGElement *element*
  The element on which to perform the given test.
- SVGRect *rect*
  The test rectangle. The values are in the initial coordinate system for the current 'svg' element.

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

- void **deselectAll**()

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

- SVGNumber **createSVGNumber**()

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

  Returns
  An SVGNumber object.

- SVGLength **createSVGLength**()

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

  Returns
  An SVGLength object.

- SVGAngle **createSVGAngle**()

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

  Returns
  An SVGAngle object.

- SVGPoint **createSVGPoint**()

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

Returns
An SVGPoint object.

- SVGMatrix **createSVGMatrix**()

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

  Returns
  An SVGMatrix object.

- SVGRect **createSVGRect**()

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

  Returns
  An SVGRect object.

- SVGTransform **createSVGTransform**()

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

  Returns
  An SVGTransform object.

- SVGTransform **createSVGTransformFromMatrix**(in SVGMatrix *matrix*)

  Creates an SVGTransform object outside of any document trees. The object is initialized to the given matrix transform (i.e., SVG_TRANSFORM_MATRIX). The values from the parameter *matrix* are copied, the *matrix* parameter is not adopted as SVGTransform::matrix.

  Parameters

  - SVGMatrix *matrix*
    The transform matrix.

  Returns
  An SVGTransform object.

- Element **getElementById**(in DOMString *elementId*)

  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

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

Returns

The matching element.

5.11.3 Interface SVGGElement

The SVGSVGElement interface corresponds to the 'g' element.

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

5.11.4 Interface SVGDefsElement

The SVGDefsElement interface corresponds to the 'defs' element.

```
interface SVGDefsElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable {
};
```

5.11.5 Interface SVGDescElement

The SVGDescElement interface corresponds to the 'desc' element.

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

5.11.6 Interface SVGTitleElement

The SVGTitleElement interface corresponds to the 'title' element.

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

5.11.7 Interface SVGSymbolElement

The SVGSymbolElement interface corresponds to the 'symbol' element.

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

5.11.8 Interface SVGUseElement

The SVGUseElement interface corresponds to the 'use' element.

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

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'use' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'use' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'use' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'use' element.

- **instanceRoot** (readonly SVGElementInstance)

  The root of the "instance tree". See description of SVGElementInstance for a discussion on the instance tree.

- **animatedInstanceRoot** (readonly SVGElementInstance)

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

5.11.9 Interface SVGElementInstance

For each 'use' element, the SVG DOM maintains a shadow tree (the "instance tree") of objects of type SVGElementInstance. An 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, correspondingElement is the 'rect')
```

```
interface SVGElementInstance : 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:

- **correspondingElement** (readonly SVGElement)

  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 SVGRectElement object for the 'rect' element.

- **correspondingUseElement** (readonly SVGUseElement)

  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).

- **parentNode** (readonly SVGElementInstance)

  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.

- **childNodes** (readonly SVGElementInstanceList)

  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).

- **firstChild** (readonly SVGElementInstance)

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

- **lastChild** (readonly SVGElementInstance)

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

- **previousSibling** (readonly SVGElementInstance)

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

- **nextSibling** (readonly SVGElementInstance)

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

5.11.10 Interface SVGElementInstanceList

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

```
interface SVGElementInstanceList {

  readonly attribute unsigned long length;

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

Attributes:

- **length** (readonly unsigned long)

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

Operations:

- SVGElementInstance **item**(in unsigned long *index*)

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

  Parameters

  - unsigned long *index*
    Index into the collection.

  Returns

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

5.11.11 Interface SVGImageElement

The SVGImageElement interface corresponds to the 'image' element.

```
interface SVGImageElement : SVGElement,
                            SVGURIReference,
                            SVGTests,
                            SVGLangSpace,
                            SVGExternalResourcesRequired,
                            SVGStylable,
                            SVGTransformable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
```

```
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

Attributes:

- **x** (readonly SVGAnimatedLength)

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

- **y** (readonly SVGAnimatedLength)

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

- **width** (readonly SVGAnimatedLength)

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

- **height** (readonly SVGAnimatedLength)

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

- **preserveAspectRatio** (readonly SVGAnimatedPreserveAspectRatio)

  Corresponds to attribute 'preserveAspectRatio' on the given 'image' element.

5.11.12 Interface SVGSwitchElement

The SVGSwitchElement interface corresponds to the 'switch' element.

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

5.11.13 Interface GetSVGDocument

This interface provides access to an SVG document embedded by reference in another DOM-based language. The expectation is that the interface is implemented on DOM objects that allow such SVG document references, such as the DOM Element object that corresponds to an HTML 'object' element. Such DOM objects are often also required to implement the EmbeddingElement defined in the Window specification [WINDOW].

This interface is deprecated and may be dropped from future versions of the SVG specification. Authors are

suggested to use the `contentDocument` attribute on the EmbeddingElement interface to obtain a referenced SVG document, if that interface is available.

```
interface GetSVGDocument {
  SVGDocument getSVGDocument();
};
```

Operations:

- SVGDocument **getSVGDocument**()

  This method must return the Document object embedded content in an embedding element, or null if there is no document.

  Note that this is equivalent to fetching the value of the `EmbeddingElement::contentDocument` attribute of the embedding element, if the EmbeddingElement interface is also implemented. The author is advised to check that the document element of the returned Document is indeed an 'svg' element instead of assuming that that will always be the case.

  Returns

  The Document object for the referenced document, or null if there is no document.

4 Basic Data Types and Interfaces

Contents

4.5.33 Interface SVGRenderingIntent

4.1 Syntax

The EBNF grammar is as used in the XML specification, with the addition of ~, a *case-insensitive literal*: characters in the ASCII range (only) are declared to be case-insensitive. For example, ~"Hello" will match (H|h)(e|e)(l|L)(l|L)(o|O). This makes the productions much easier to read.

?	optional, zero or one
+	one or more
*	zero or more
|	alternation
"string"	literal
~"string"	case-insensitive literal
[]	a character range
[^]	excluded character range
()	grouping

4.2 Basic data types

This section defines a number of common data types used in the definitions of SVG properties and attributes. Some data types that are not referenced by multiple properties and attributes are defined inline in subsequent chapters.

Note that, as noted below, the specification of some types is different for CSS property values in style sheets (in the 'style' attribute, 'style' element or an external style sheet) than it is for for XML attribute values (including presentation attributes). This is due to restrictions in the CSS grammar. For example, scientific notation is allowed in attributes, including presentation attributes, but not in style sheets.

<angle> · Angles are specified in one of two ways depending upon whether they are used in CSS property syntax or SVG presentation attribute syntax:

- When an <angle> is used in a style sheet or with a property in a 'style' attribute, the syntax must match the following pattern:

  ```
  angle ::= number (~"deg" | ~"grad" | ~"rad")?
  ```

  where **deg** indicates degrees, **grad** indicates grads and **rad** indicates radians. The unit identifier may be in lower (recommended) or upper case.

  For properties defined in CSS2 [CSS2], an angle unit identifier must be provided (for non-zero values). For SVG-specific properties the angle unit identifier is optional. If a unit is not provided, the angle value is assumed to be in degrees.
- When an <angle> is used in an SVG presentation attribute, the syntax must match the following pattern:

```
angle ::= number ("deg" | "grad" | "rad")?
```

The unit identifier, if present, must be in lower case; if not present, the angle value is assumed to be in degrees.

In the SVG DOM, <angle> values are represented using SVGAngle or SVGAnimatedAngle objects.

<anything> · The basic type <anything> is a sequence of zero or more characters. Specifically:

```
anything ::= Char*
```

where Char is the production for a character, as defined in XML 1.0 ([XML10], section 2.2).

<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 syntax and basic data types ([CSS2], section 4.3.6), with the exception that SVG allows 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 found in User preferences for colors ([CSS2], section 18.2).

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 an RGB start-function followed by a comma-separated list of three numerical values (either three integer values or three percentage values) followed by ")". An RGB start-function is the case-insensitive string "rgb(", for example "RGB(" or "rGb(". For compatibility, the all-lowercase form "rgb(" is preferred. 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 [SRGB]. Using sRGB provides an unambiguous and objectively measurable definition of the color, which can be related to international standards (see [COLORIMETRY]).

```
color    ::= "#" hexdigit hexdigit hexdigit (hexdigit hexdigit hexdigit)?
             | "rgb(" wsp* integer comma integer comma integer wsp* ")"
             | "rgb(" wsp* integer "%" comma integer "%" comma integer "%" wsp* ")"
             | color-keyword
hexdigit ::= [0-9A-Fa-f]
comma    ::= wsp* "," wsp*
```

where *color-keyword* matches (case insensitively) one of the color keywords listed in Recognized color keyword names below, or one of the system color keywords listed in User preferences for colors ([CSS2], section 18.2).

The corresponding SVG DOM interface definitions for <color> are defined in Document Object Model CSS;

in particular, see RGBColor ([DOM2STYLE], section 2.2). SVG's extension to color, including the ability to specify ICC-based colors, are represented using DOM interface SVGColor.

<coordinate> · A <coordinate> is 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). Its syntax is the same as that for <length>.

```
coordinate ::= length
```

Within the SVG DOM, a <coordinate> is represented as an SVGLength or an SVGAnimatedLength.

<frequency> · Frequency values are used with aural properties. As defined in CSS2, a frequency value is a <number> immediately followed by a frequency unit identifier. The frequency unit identifiers are:

- **Hz**: Hertz
- **kHz**: kilo Hertz

Frequency values may not be negative.

In the SVG DOM, <frequency> values are represented using the CSSPrimitiveValue interface defined in Document Object Model CSS ([DOM2STYLE], section 2.2).

<FuncIRI> · Functional notation for an IRI: "url(" <IRI> ")".

<icccolor> · An <icccolor> is an ICC color specification. In SVG 1.1, an ICC color specification is given by a name, which references a 'color-profile' element, and one or more color component values. The grammar is as follows:

```
icccolor ::= "icc-color(" name (comma-wsp number)+ ")"
name     ::= [^,()#x20#x9#xD#xA] /* any char except ",", "(", ")" or wsp */
```

The corresponding SVG DOM interface for <icccolor> is SVGICCColor.

<integer> · An <integer> is specified as an optional sign character ("+" or "-") followed by one or more digits "0" to "9":

```
integer ::= [+-]? [0-9]+
```

If the sign character is not present, the number is non-negative.

Unless stated otherwise for a particular attribute or property, the range for an <integer> encompasses (at a minimum) -2147483648 to 2147483647.

Within the SVG DOM, an <integer> is represented as a **long** or an SVGAnimatedInteger.

<IRI> · An Internationalized Resource Identifier (see IRI). For the specification of IRI references in SVG, see IRI references.

<length> · A length is a distance measurement, given as a number along with a unit which may be optional. Lengths are specified in one of two ways depending upon whether they are used in CSS property syntax or SVG presentation attribute syntax:

- When a <length> is used in a style sheet or with a property in a 'style' attribute, the syntax must match the following pattern:

  ```
  length ::= number (~"em" | ~"ex" | ~"px" | ~"in" | ~"cm" | ~"mm" | ~"pt" | ~"pc")?
  ```

  See the CSS2 specification for the meanings of the unit identifiers. The unit identifier may be in lower (recommended) or upper case.

  For properties defined in CSS2 [CSS2], a length unit identifier must be provided (for non-zero values). For SVG-specific properties, the length unit identifier is optional. If a unit is not provided, the length value represents a distance in the current user coordinate system.
- When a <length> is used in an SVG presentation attribute, the syntax must match the following pattern:

  ```
  length ::= number ("em" | "ex" | "px" | "in" | "cm" | "mm" | "pt" | "pc" | "%")?
  ```

  The unit identifier, if present, must be in lower case; if not present, the length value represents a distance in the current user coordinate system.

  Note that the non-property <length> definition also allows a percentage unit identifier. The meaning of a percentage length value depends on the attribute for which the percentage length value has been specified. Two common cases are: (a) when a percentage length value represents a percentage of the viewport width or height (refer to the section that discusses units in general), and (b) when a percentage length value represents a percentage of the bounding box width or height on a given object (refer to the section that describes object bounding box units).

In the SVG DOM, <length> values are represented using SVGLength or SVGAnimatedLength objects.

<list-of-family-names> · A <list-of-family-names> is a list of font family names using the same syntax as the '**font-family**' property, excluding the <generic-family> and '**inherit**' values.

<list-of-strings> · A <list-of-strings> consists of a separated sequence of <string>s. String lists are white space-separated, where white space is defined as one or more of the following consecutive characters: "space" (U+0020), "tab" (U+0009), "line feed" (U+000A) and "carriage return" (U+000D).

The following is an EBNF grammar describing the <list-of-strings> syntax:

```
list-of-strings ::= string
                    | string wsp list-of-strings
string          ::= [^#x9#xA#xD#x20]*
wsp             ::= [#x9#xA#xD#x20]+
```

<list-of-*Ts*> · (Where *T* is a type other than <string> and <family-name>.) A list consists of a separated sequence of values. 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" (U+0020), "tab" (U+0009), "line feed" (U+000A), "carriage return" (U+000D) and "form-feed" (U+000C).

The following is a template for an EBNF grammar describing the <list-of-*T*s> syntax:

```
list-of-Ts ::= T
               | T comma-wsp list-of-Ts
comma-wsp  ::= (wsp+ ","? wsp*) | ("," wsp*)
wsp        ::= (#x20 | #x9 | #xD | #xA)
```

Within the SVG DOM, values of a <list-of-*T*s> type are represented by an interface specific for the particular type *T*. For example, a <list-of-lengths> is represented in the SVG DOM using an SVGLengthList or SVGAnimatedLengthList object.

<name> · A name, which is a string where a few characters of syntactic significance are disallowed.

```
name  ::= [^,()#x20#x9#xD#xA] /* any char except ",", "(", ")" or wsp */
```

<number> · Real numbers are specified in one of two ways. When used in a style sheet, a <number> is defined as follows:

```
number ::= integer
           | [+-]? [0-9]* "." [0-9]+
```

This syntax is the same as the definition in CSS ([CSS2], section 4.3.1).

When used in an SVG attribute, a <number> is defined differently, to allow numbers with large magnitudes to be specified more concisely:

```
number ::= integer ([Ee] integer)?
           | [+-]? [0-9]* "." [0-9]+ ([Ee] integer)?
```

Within the SVG DOM, a <number> is represented as a **float**, SVGNumber or a SVGAnimatedNumber.

<number-optional-number> · A pair of <number>s, where the second <number> is optional.

```
number-optional-number ::= number
                           | number comma-wsp number
```

In the SVG DOM, a <number-optional-number> is represented using a pair of SVGAnimatedInteger or SVGAnimatedNumber objects.

<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> values are represented using SVGPaint objects.

<percentage> · Percentages are specified as a number followed by a "%" character:

```
percentage ::= number "%"
```

Note that the definition of <number> depends on whether the percentage is specified in a style sheet or in an attribute that is not also a presentation attribute.

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 represented using an SVGNumber or SVGAnimatedNumber object.

<time> · A time value is a <number> immediately followed by a time unit identifier. The time unit identifiers are:

- **ms**: milliseconds
- **s**: seconds

In the SVG DOM, <time> values are represented using the CSSPrimitiveValue interface defined in Document Object Model CSS ([DOM2STYLE], section 2.2).

<transform-list> · A <transform-list> is used to specify a list of coordinate system transformations. A detailed description of the possible values for a <transform-list> is given in Modifying the User Coordinate System: the transform attribute.

Within the SVG DOM, a <transform-list> value is represented using an SVGTransformList or SVGAnimatedTransformList object.

<XML-Name> · An XML name, as defined by the Name production in *Extensible Markup Language (XML) 1.0* ([XML10], section 2.3).

4.3 Real number precision

Unless stated otherwise for a particular attribute or property, a <number> has the capacity for at least a single-precision floating point number 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 for intermediate calculations on certain numerical operations.

4.4 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)
darkseagreen	rgb(143, 188, 143)
darkslateblue	rgb(72, 61, 139)
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)
orchid	rgb(218, 112, 214)
palegoldenrod	rgb(238, 232, 170)
palegreen	rgb(152, 251, 152)

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)
lightcoral	rgb(240, 128, 128)
lightcyan	rgb(224, 255, 255)
lightgoldenrodyellow	rgb(250, 250, 210)
lightgray	rgb(211, 211, 211)
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)
violet	rgb(238, 130, 238)
wheat	rgb(245, 222, 179)
white	rgb(255, 255, 255)
whitesmoke	rgb(245, 245, 245)
yellow	rgb(255, 255, 0)

lightgreen	rgb(144, 238, 144)	yellowgreen	rgb(154, 205, 50)
lightgrey	rgb(211, 211, 211)		

4.5 Basic DOM interfaces

4.5.1 Interface SVGElement

All of the SVG DOM interfaces that correspond directly to elements in the SVG language (such as the SVGPathElement interface for the 'path' element) derive from the SVGElement interface.

```
interface SVGElement : Element {
          attribute DOMString id setraises(DOMException);
          attribute DOMString xmlbase setraises(DOMException);
  readonly attribute SVGSVGElement ownerSVGElement;
  readonly attribute SVGElement viewportElement;
};
```

Attributes:

- **id** (DOMString)

  The value of the 'id' attribute on the given element, or the empty string if 'id' is not present.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **xmlbase** (DOMString)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **ownerSVGElement** (readonly SVGSVGElement)

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

- **viewportElement** (readonly SVGElement)

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

4.5.2 Interface SVGAnimatedBoolean

Used for attributes of type boolean which can be animated.

```
interface SVGAnimatedBoolean {
          attribute boolean baseVal setraises(DOMException);
  readonly attribute boolean animVal;
};
```

Attributes:

- **baseVal** (boolean)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **animVal** (readonly boolean)

  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.

4.5.3 Interface SVGAnimatedString

Used for attributes of type DOMString which can be animated.

```
interface SVGAnimatedString {
          attribute DOMString baseVal setraises(DOMException);
  readonly attribute DOMString animVal;
};
```

Attributes:

- **baseVal** (DOMString)

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

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **animVal** (readonly DOMString)

  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.

4.5.4 Interface SVGStringList

This interface defines a list of DOMString values.

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.

```
interface SVGStringList {

  readonly attribute unsigned long numberOfItems;

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

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- DOMString **initialize**(in DOMString *newItem*)

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

  Parameters

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

  Returns
  The item being inserted into the list.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- DOMString **getItem**(in unsigned long *index*)

  Returns the specified item from the list.

  Parameters

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

  Returns
  The selected item.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- DOMString **insertItemBefore**(in DOMString *newItem*, in unsigned long *index*)

  Inserts a new item into the list at the specified position. The first item is number 0.

  Parameters

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

- 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.

Returns
The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.

- DOMString **replaceItem**(in DOMString *newItem*, in unsigned long *index*)

  Replaces an existing item in the list with a new item.

  Parameters

  - DOMString *newItem*
    The item which is to be inserted into the list.
  - unsigned long *index*
    The index of the item which is to be replaced. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- DOMString **removeItem**(in unsigned long *index*)

  Removes an existing item from the list.

  Parameters

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

Returns
The removed item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.
- DOMException, code INDEX_SIZE_ERR
  Raised if the index number is greater than or equal to numberOfItems.

- DOMString **appendItem**(in DOMString *newItem*)

  Inserts a new item at the end of the list.

  Parameters

  - DOMString *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

4.5.5 Interface SVGAnimatedEnumeration

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

```
interface SVGAnimatedEnumeration {
          attribute unsigned short baseVal setraises(DOMException);
  readonly attribute unsigned short animVal;
};
```

Attributes:

- **baseVal** (unsigned short)

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

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **animVal** (readonly unsigned short)

  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.

4.5.6 Interface SVGAnimatedInteger

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

```
interface SVGAnimatedInteger {
           attribute long baseVal setraises(DOMException);
  readonly attribute long animVal;
};
```

Attributes:

- **baseVal** (long)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **animVal** (readonly long)

  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.

4.5.7 Interface SVGNumber

Used for attributes of basic type <number>.

```
interface SVGNumber {
  attribute float value setraises(DOMException);
};
```

Attributes:

- **value** (float)

  The value of the given attribute.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

4.5.8 Interface SVGAnimatedNumber

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

```
interface SVGAnimatedNumber {
          attribute float baseVal setraises(DOMException);
  readonly attribute float animVal;
};
```

Attributes:

- **baseVal** (float)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **animVal** (readonly float)

  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.

4.5.9 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.

An SVGNumberList object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
interface SVGNumberList {

  readonly attribute unsigned long numberOfItems;

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

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGNumber **initialize**(in SVGNumber *newItem*)

  Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter. If the inserted item is already in a list, it is removed from its previous list before it is inserted into this list. The inserted item is the item itself and not a copy.

  Parameters

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

  Returns
  The item being inserted into the list.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGNumber **getItem**(in unsigned long *index*)

  Returns the specified item from the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

  Parameters

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

  Returns
  The selected item.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGNumber **insertItemBefore**(in SVGNumber *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to insert before is *before* the removal of the item.

  Parameters

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

  - 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.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGNumber **replaceItem**(in SVGNumber *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to replace is *before* the removal of the item.

  Parameters

  - SVGNumber *newItem*
    The item which is to be inserted into the list.
  - unsigned long *index*
    The index of the item which is to be replaced. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGNumber **removeItem**(in unsigned long *index*)

  Removes an existing item from the list.

  Parameters

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

  Returns
  The removed item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGNumber **appendItem**(in SVGNumber *newItem*)

  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. The inserted item is the item itself and not a copy.

  Parameters

  - SVGNumber *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

4.5.10 Interface SVGAnimatedNumberList

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

```
interface SVGAnimatedNumberList {
  readonly attribute SVGNumberList baseVal;
  readonly attribute SVGNumberList animVal;
};
```

Attributes:

- **baseVal** (readonly SVGNumberList)

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

- **animVal** (readonly SVGNumberList)

  A read only SVGNumberList representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGNumberList will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

4.5.11 Interface SVGLength

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

An SVGLength object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
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 setraises(DOMException);
           attribute float valueInSpecifiedUnits setraises(DOMException);
           attribute DOMString valueAsString setraises(DOMException);

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

Constants in group "Length Unit Types":

- **SVG_LENGTHTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  No unit type was provided (i.e., a unitless value was specified), which indicates a value in user units.

- **SVG_LENGTHTYPE_PERCENTAGE** (unsigned short)

  A percentage value was specified.

- **SVG_LENGTHTYPE_EMS** (unsigned short)

  A value was specified using the em units defined in CSS2.

- **SVG_LENGTHTYPE_EXS** (unsigned short)

  A value was specified using the ex units defined in CSS2.

- **SVG_LENGTHTYPE_PX** (unsigned short)

  A value was specified using the px units defined in CSS2.

- **SVG_LENGTHTYPE_CM** (unsigned short)

  A value was specified using the cm units defined in CSS2.

- **SVG_LENGTHTYPE_MM** (unsigned short)

  A value was specified using the mm units defined in CSS2.

- **SVG_LENGTHTYPE_IN** (unsigned short)

  A value was specified using the in units defined in CSS2.

- **SVG_LENGTHTYPE_PT** (unsigned short)

  A value was specified using the pt units defined in CSS2.

- **SVG_LENGTHTYPE_PC** (unsigned short)

  A value was specified using the pc units defined in CSS2.

Attributes:

- **unitType** (readonly unsigned short)

  The type of the value as specified by one of the SVG_LENGTHTYPE_* constants defined on this interface.

- **value** (float)

  The value as a 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, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the length corresponds to a read only attribute or when the object itself is read only.

- **valueInSpecifiedUnits** (float)

  The 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, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the length corresponds to a read only attribute or when the object itself is read only.

- **valueAsString** (DOMString)

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

  Exceptions on setting

  - DOMException, code SYNTAX_ERR
    Raised if the assigned string cannot be parsed as a valid <length>.
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the length corresponds to a read only attribute or when the object itself is read only.

Operations:

- void **newValueSpecifiedUnits**(in unsigned short *unitType*, in float *valueInSpecifiedUnits*)

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

  Parameters

  - unsigned short *unitType*
    The unit type for the value (e.g., SVG_LENGTHTYPE_MM).
  - float *valueInSpecifiedUnits*
    The new value.

  Exceptions

  - DOMException, code NOT_SUPPORTED_ERR
    Raised if unitType is SVG_LENGTHTYPE_UNKNOWN or not a valid unit type constant (one of the other SVG_LENGTHTYPE_* constants defined on this interface).
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the length corresponds to a read only attribute or when the object itself is read only.

- void **convertToSpecifiedUnits**(in unsigned short *unitType*)

  Preserve the same underlying stored value, but reset the stored unit identifier to the given *unitType*. Object attributes unitType, valueInSpecifiedUnits 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, valueInSpecifiedUnits would be changed to the numeric value 5 and valueAsString would be changed to "5mm".

  Parameters

  - unsigned short *unitType*
    The unit type to switch to (e.g., SVG_LENGTHTYPE_MM).

  Exceptions

  - DOMException, code NOT_SUPPORTED_ERR
    Raised if unitType is SVG_LENGTHTYPE_UNKNOWN or not a valid unit type constant (one of the other SVG_LENGTHTYPE_* constants defined on this interface).
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the length corresponds to a read only attribute or when the object itself is read only.

4.5.12 Interface SVGAnimatedLength

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

```
interface SVGAnimatedLength {
  readonly attribute SVGLength baseVal;
  readonly attribute SVGLength animVal;
};
```

Attributes:

- **baseVal** (readonly SVGLength)

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

- **animVal** (readonly SVGLength)

  A read only SVGLength representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGLength will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

4.5.13 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.

An SVGLengthList object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
interface SVGLengthList {

  readonly attribute unsigned long numberOfItems;

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

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGLength **initialize**(in SVGLength *newItem*)

  Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter. If the inserted item is already in a list, it is removed from its previous list before it is inserted into this list. The inserted item is the item itself and not a copy.

  Parameters

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

Returns
The item being inserted into the list.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGLength **getItem**(in unsigned long *index*)

Returns the specified item from the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

Parameters

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

Returns
The selected item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGLength **insertItemBefore**(in SVGLength *newItem*, in unsigned long *index*)

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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to insert before is *before* the removal of the item.

Parameters

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

- 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.

Returns
The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGLength **replaceItem**(in SVGLength *newItem*, in unsigned long *index*)

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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to replace is *before* the removal of the item.

Parameters

- SVGLength *newItem*
  The item which is to be inserted into the list.
- unsigned long *index*
  The index of the item which is to be replaced. The first item is number 0.

Returns
The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.
- DOMException, code INDEX_SIZE_ERR
  Raised if the index number is greater than or equal to numberOfItems.

- SVGLength **removeItem**(in unsigned long *index*)

Removes an existing item from the list.

Parameters

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

Returns
The removed item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.
- DOMException, code INDEX_SIZE_ERR
  Raised if the index number is greater than or equal to numberOfItems.

- SVGLength **appendItem**(in SVGLength *newItem*)

  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. The inserted item is the item itself and not a copy.

  Parameters

  - SVGLength *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
    The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

4.5.14 Interface SVGAnimatedLengthList

Used for attributes of type SVGLengthList which can be animated.

```
interface SVGAnimatedLengthList {
  readonly attribute SVGLengthList baseVal;
  readonly attribute SVGLengthList animVal;
};
```

Attributes:

- **baseVal** (readonly SVGLengthList)

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

- **animVal** (readonly SVGLengthList)

  A read only SVGLengthList representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGLengthList will have the same contents as baseVal. The

object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

4.5.15 Interface SVGAngle

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

An SVGAngle object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
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 setraises(DOMException);
           attribute float valueInSpecifiedUnits setraises(DOMException);
           attribute DOMString valueAsString setraises(DOMException);

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

Constants in group "Angle Unit Types":

- **SVG_ANGLETYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  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** (unsigned short)

  The unit type was explicitly set to degrees.

- **SVG_ANGLETYPE_RAD** (unsigned short)

  The unit type is radians.

- **SVG_ANGLETYPE_GRAD** (unsigned short)

  The unit type is radians.

Attributes:

- **unitType** (readonly unsigned short)

  The type of the value as specified by one of the SVG_ANGLETYPE_* constants defined on this interface.

- **value** (float)

  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, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the angle corresponds to a read only attribute or when the object itself is read only.

- **valueInSpecifiedUnits** (float)

  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, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the angle corresponds to a read only attribute or when the object itself is read only.

- **valueAsString** (DOMString)

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

  Exceptions on setting

  - DOMException, code SYNTAX_ERR
    Raised if the assigned string cannot be parsed as a valid <angle>.
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the angle corresponds to a read only attribute or when the object itself is read only.

Operations:

- void **newValueSpecifiedUnits**(in unsigned short *unitType*, in float *valueInSpecifiedUnits*)

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

  Parameters

  - unsigned short *unitType*
    The unit type for the value (e.g., SVG_ANGLETYPE_DEG).
  - float *valueInSpecifiedUnits*
    The angle value.

  Exceptions

  - DOMException, code NOT_SUPPORTED_ERR
    Raised if unitType is SVG_ANGLETYPE_UNKNOWN or not a valid unit type constant (one of the other SVG_ANGLETYPE_* constants defined on this interface).
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the angle corresponds to a read only attribute or when the object itself is read only.

- void **convertToSpecifiedUnits**(in unsigned short *unitType*)

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

  Parameters

  - unsigned short *unitType*
    The unit type to switch to (e.g., SVG_ANGLETYPE_DEG).

  Exceptions

  - DOMException, code NOT_SUPPORTED_ERR
    Raised if unitType is SVG_ANGLETYPE_UNKNOWN or not a valid unit type constant (one of the other SVG_ANGLETYPE_* constants defined on this interface).
  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the angle corresponds to a read only attribute or when the object itself is read only.

4.5.16 Interface SVGAnimatedAngle

Used for attributes of basic data type <angle> that can be animated.

```
interface SVGAnimatedAngle {
  readonly attribute SVGAngle baseVal;
  readonly attribute SVGAngle animVal;
};
```

Attributes:

- **baseVal** (readonly SVGAngle)

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

- **animVal** (readonly SVGAngle)

  A read only SVGAngle representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGAngle will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

4.5.17 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 DOM Level 2 Style; in particular, see the RGBColor interface ([DOM2STYLE], section 2.2).

Note: The SVGColor interface is deprecated, and may be dropped from future versions of the SVG specification.

```
interface SVGColor : 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 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);
};
```

Constants in group "Color Types":

- **SVG_COLORTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  An sRGB color has been specified without an alternative ICC color specification.

- **SVG_COLORTYPE_RGBCOLOR_ICCCOLOR** (unsigned short)

  An sRGB color has been specified along with an alternative ICC color specification.

- **SVG_COLORTYPE_CURRENTCOLOR** (unsigned short)

  Corresponds to when keyword **currentColor** has been specified.

Attributes:

- **colorType** (readonly unsigned short)

  The type of the value as specified by one of the SVG_COLORTYPE_* constants defined on this interface.

- **rgbColor** (readonly RGBColor)

  The color specified in the sRGB color space.

- **iccColor** (readonly SVGICCColor)

  The alternate ICC color specification.

Operations:

- void **setRGBColor**(in DOMString *rgbColor*)

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

  Parameters

  - DOMString *rgbColor*
    A string that matches <color>, which specifies the new sRGB color value.

Exceptions

- SVGException, code SVG_INVALID_VALUE_ERR
  Raised if *rgbColor* does not match <color>.

- void **setRGBColorICCColor**(in DOMString *rgbColor*, in DOMString *iccColor*)

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

  Parameters

  - DOMString *rgbColor*
    A string that matches <color>, which specifies the new sRGB color value.
  - DOMString *iccColor*
    A string that matches <icccolor>, which specifies the alternate ICC color specification.

  Exceptions

  - SVGException, code SVG_INVALID_VALUE_ERR
    Raised if *rgbColor* does not match <color> or if *iccColor* does not match <icccolor>.

- void **setColor**(in unsigned short *colorType*, in DOMString *rgbColor*, in DOMString *iccColor*)

  Sets the color value as specified by the parameters. If *colorType* requires an RGBColor, then *rgbColor* must be a string that matches <color>; otherwise, *rgbColor*. must be null. If *colorType* requires an SVGICCColor, then *iccColor* must be a string that matches <icccolor>; otherwise, *iccColor* must be null.

  Parameters

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

  Exceptions

  - SVGException, code SVG_INVALID_VALUE_ERR
    Raised if one of the parameters has an invalid value.

4.5.18 Interface SVGICCColor

The SVGICCColor interface expresses an ICC-based color specification.

Note: The SVGICCColor interface is deprecated, and may be dropped from future versions of the SVG specification.

```
interface SVGICCColor {
          attribute DOMString colorProfile setraises(DOMException);
  readonly attribute SVGNumberList colors;
};
```

Attributes:

- **colorProfile** (DOMString)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **colors** (readonly SVGNumberList)

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

4.5.19 Interface SVGRect

Represents rectangular geometry. 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.

An SVGRect object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
interface SVGRect {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float width setraises(DOMException);
  attribute float height setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The *x* coordinate of the rectangle, in user units.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the rectangle corresponds to a read only attribute or when the object itself is read only.

- **y** (float)

  The *y* coordinate of the rectangle, in user units.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the rectangle corresponds to a read only attribute or when the object itself is read only.

- **width** (float)

  The *width* coordinate of the rectangle, in user units.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the rectangle corresponds to a read only attribute or when the object itself is read only.

- **height** (float)

  The *height* coordinate of the rectangle, in user units.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the rectangle corresponds to a read only attribute or when the object itself is read only.

4.5.20 Interface SVGAnimatedRect

Used for attributes of type SVGRect which can be animated.

```
interface SVGAnimatedRect {
  readonly attribute SVGRect baseVal;
  readonly attribute SVGRect animVal;
};
```

Attributes:

- **baseVal** (readonly SVGRect)

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

- **animVal** (readonly SVGRect)

  A read only SVGRect representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGRect will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

4.5.21 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.

```
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;
};
```

Constants in group "Unit Types":

- **SVG_UNIT_TYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'userSpaceOnUse'**.

- **SVG_UNIT_TYPE_OBJECTBOUNDINGBOX** (unsigned short)

  Corresponds to value **'objectBoundingBox'**.

4.5.22 Interface SVGStylable

The SVGStylable interface is implemented on all objects corresponding to SVG elements that can have 'style', 'class' and presentation attributes specified on them. It is thus an ancestor interface for many of the interfaces defined in this specification.

```
interface SVGStylable {

  readonly attribute SVGAnimatedString className;
  readonly attribute CSSStyleDeclaration style;

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

Attributes:

- **className** (readonly SVGAnimatedString)

  Corresponds to attribute 'class' on the given element.

- **style** (readonly CSSStyleDeclaration)

  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.

Operations:

- CSSValue **getPresentationAttribute**(in DOMString *name*)

  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.
  Note: The `getPresentationAttribute` method is deprecated, and may be dropped from future versions of the SVG specification.

  Parameters

  - DOMString *name*
    The name of the presentation attribute whose value is to be returned.

  Returns

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

4.5.23 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.

```
interface SVGLocatable {

  readonly attribute SVGElement nearestViewportElement;
  readonly attribute SVGElement farthestViewportElement;
```

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

Attributes:

- **nearestViewportElement** (readonly SVGElement)

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

- **farthestViewportElement** (readonly SVGElement)

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

Operations:

- SVGRect **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 stroking, clipping, masking and filter effects). Note that getBBox must return the actual bounding box at the time the method was called, even in case the element has not yet been rendered.

  Returns
  An SVGRect object that defines the bounding box.

- SVGMatrix **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.

  Returns
  An SVGMatrix object that defines the CTM.

- SVGMatrix **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. Note that null is returned if this element is not

hooked into the document tree. This method would have been more aptly named as `getClientCTM`, but the name `getScreenCTM` is kept for historical reasons.

Returns
An SVGMatrix object that defines the given transformation matrix.

- SVGMatrix **getTransformToElement**(in SVGElement *element*)

  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

  - SVGElement *element*
    The target element.

  Returns
  An SVGMatrix object that defines the transformation.

  Exceptions

  - SVGException, code 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).

4.5.24 Interface SVGTransformable

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

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

Attributes:

- **transform** (readonly SVGAnimatedTransformList)

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

4.5.25 Interface SVGTests

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

```
interface SVGTests {

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

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

Attributes:

- **requiredFeatures** (readonly SVGStringList)

  Corresponds to attribute 'requiredFeatures' on the given element.

- **requiredExtensions** (readonly SVGStringList)

  Corresponds to attribute 'requiredExtensions' on the given element.

- **systemLanguage** (readonly SVGStringList)

  Corresponds to attribute 'systemLanguage' on the given element.

Operations:

- boolean **hasExtension**(in DOMString *extension*)

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

  Parameters

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

  Returns
  True or false, depending on whether the given extension is supported.

4.5.26 Interface SVGLangSpace

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

```
interface SVGLangSpace {
  attribute DOMString xmllang setraises(DOMException);
  attribute DOMString xmlspace setraises(DOMException);
};
```

Attributes:

- **xmllang** (DOMString)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **xmlspace** (DOMString)

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

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

4.5.27 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.

```
interface SVGExternalResourcesRequired {
  readonly attribute SVGAnimatedBoolean externalResourcesRequired;
};
```

Attributes:

- **externalResourcesRequired** (readonly SVGAnimatedBoolean)

  Corresponds to attribute 'externalResourcesRequired' on the given element. Note that the SVG DOM defines the attribute 'externalResourcesRequired' as being of type SVGAnimatedBoolean, whereas the SVG language definition says that 'externalResourcesRequired' is not animated. Because the SVG language definition states that 'externalResourcesRequired' cannot be animated, the animVal will always be the same as the baseVal.

4.5.28 Interface SVGFitToViewBox

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

```
interface SVGFitToViewBox {
  readonly attribute SVGAnimatedRect viewBox;
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

Attributes:

- **viewBox** (readonly SVGAnimatedRect)

  Corresponds to attribute 'viewBox' on the given element.

- **preserveAspectRatio** (readonly SVGAnimatedPreserveAspectRatio)

  Corresponds to attribute 'preserveAspectRatio' on the given element.

4.5.29 Interface SVGZoomAndPan

The SVGZoomAndPan interface defines attribute zoomAndPan and associated constants.

```
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 setraises(DOMException);
};
```

Constants in group "Zoom and Pan Types":

- **SVG_ZOOMANDPAN_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value '**disable**'.

- **SVG_ZOOMANDPAN_MAGNIFY** (unsigned short)

  Corresponds to value '**magnify**'.

Attributes:

- **zoomAndPan** (unsigned short)

  Corresponds to attribute 'zoomAndPan' on the given element. The value must be one of the SVG_ZOOMANDPAN_* constants defined on this interface.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

4.5.30 Interface SVGViewSpec

The interface corresponds to an SVG View Specification.

```
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:

- **transform** (readonly SVGTransformList)

  Corresponds to the transform setting on the SVG View Specification.

- **viewTarget** (readonly SVGElement)

  Corresponds to the viewTarget setting on the SVG View Specification.

- **viewBoxString** (readonly DOMString)

  Corresponds to the viewBox setting on the SVG View Specification.

- **preserveAspectRatioString** (readonly DOMString)

  Corresponds to the preserveAspectRatio setting on the SVG View Specification.

- **transformString** (readonly DOMString)

  Corresponds to the transform setting on the SVG View Specification.

- **viewTargetString** (readonly DOMString)

  Corresponds to the viewTarget setting on the SVG View Specification.

4.5.31 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.

```
interface SVGURIReference {
  readonly attribute SVGAnimatedString href;
};
```

Attributes:

- **href** (readonly SVGAnimatedString)

  Corresponds to attribute 'xlink:href' on the given element.

4.5.32 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.

```
interface SVGCSSRule : CSSRule {
  const unsigned short COLOR_PROFILE_RULE = 7;
};
```

Constants:

- **COLOR_PROFILE_RULE** (unsigned short)

  The rule is an @color-profile.

4.5.33 Interface SVGRenderingIntent

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

```
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;
};
```

Constants in group "Rendering Intent Types":

- **RENDERING_INTENT_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to a value of **'auto'**.

- **RENDERING_INTENT_PERCEPTUAL** (unsigned short)

  Corresponds to a value of **'perceptual'**.

- **RENDERING_INTENT_RELATIVE_COLORIMETRIC** (unsigned short)

  Corresponds to a value of **'relative-colorimetric'**.

- **RENDERING_INTENT_SATURATION** (unsigned short)

  Corresponds to a value of **'saturation'**.

- **RENDERING_INTENT_ABSOLUTE_COLORIMETRIC** (unsigned short)

  Corresponds to a value of **'absolute-colorimetric'**.

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 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 element.
  - **'color'**, used to provide a potential indirect value (**currentColor**) for the **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'** and **'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 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]. 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**:
   CSS is a widely implemented declarative language for assigning styling properties to XML content, including SVG [CSS2]. 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 Stylesheets**
  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 also 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 specified on a 'rect' 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="10cm" height="5cm" viewBox="0 0 1000 500">
  <rect x="200" y="100" width="600" height="300"
        fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

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] [XSLT2] or supplemental styling by adding CSS style rules to override some of the presentation attributes.
- **Convenient generation using 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 present-

ation 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 Precedence of non-CSS presentational hints ([CSS2], section 6.4.4), 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 ([CSS2], section 6.4.2) within a presentation attribute definition is an invalid value.

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"** (see 'attributeType').

6.5 Styling with XSL

XSL style sheets [XSLT] [XSLT2] 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 version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg">
  <xsl:output
    method="xml"
    encoding="utf-8"
    doctype-public="-//W3C//DTD SVG 1.1//EN"
    doctype-system="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"/>
  <!-- Add version to topmost 'svg' element -->
  <xsl:template match="/svg:svg">
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:attribute name="version">1.1</xsl:attribute>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
  <!-- Add styling to all 'rect' elements -->
  <xsl:template match="svg: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>
</xsl:stylesheet>
```

SVG file to be transformed by mystyle.xsl

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet href="mystyle.xsl" type="application/xml"?>
<svg xmlns="http://www.w3.org/2000/svg"
     width="10cm" height="5cm">
  <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 1.1//EN"
      "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     width="10cm" height="5cm" version="1.1">
  <rect x="2cm" y="1cm" width="6cm" height="3cm" fill="red" stroke="blue" stroke-width="3"/>
</svg>
```

6.6 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="10cm" height="5cm" viewBox="0 0 1000 500">
```

```
  <rect x="200" y="100" width="600" height="300"/>
</svg>
```

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="10cm" height="5cm" viewBox="0 0 1000 500">
  <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>
```

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". Note that property declarations inside the 'style' attribute must follow CSS style rules, see The 'style' attribute.

The following example shows how the '**fill**' and '**stroke**' properties can be specified on a 'rect' 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="10cm" height="5cm" viewBox="0 0 1000 500">
  <rect x="200" y="100" width="600" height="300"
        style="fill: red; stroke: blue; stroke-width: 3"/>
</svg>
```

In an SVG user agent that supports CSS style sheets, the following facilities from CSS2 must be supported:

- CSS2 selectors within style sheets ([CSS2], chapter 5). Because SVG is intended to be used as one component in a multiple namespace XML application and CSS2 is not namespace aware, type selectors will only match against the local part of the element's qualified name.
- External CSS style sheets [XML-SS], CSS style sheets within 'style' elements and CSS declaration blocks ([CSS2], section 4.1.7) within 'style' attributes attached to specific SVG elements.
- CSS2 rules for assigning property values, cascading and inheritance ([CSS2], chapter 6).
- @font-face, @media, @import and @charset rules within style sheets ([CSS2], sections 15.3.1, 7.2.1, 6.3 and 4.4).
- CSS2's dynamic pseudo-classes :hover, :active and :focus and pseudo-classes :first-child, :visited, :link and

:lang ([CSS2], section 5.11). The remaining CSS2 pseudo-classes, including those having to do with generated content ([CSS2], chapter 12), are not part of the SVG language definition. 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 ([CSS2], chapter 19). (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 section ([XML10], section 2.7).

SVG defines an @color-profile at-rule ([CSS2], section 4.1.6) 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 says ([CSS2], section 4.3.4) that relative URIs (as defined in *Uniform Resource Identifiers (URI): Generic Syntax* [RFC3986]) within style sheets are resolved such that the base URI is that of the style sheet, not that of the referencing document.

6.7 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 ([CSS2], section 4.1.3).

Because presentation attributes are expressed as XML attributes, presentation attributes are case-sensitive and must match the exact name as specified in the DTD (see the `SVG.Presentation.attrib` entity in the DTD, which expands to all of the presentation attributes). When using a presentation attribute to specify a value for the '**fill**' property, the presentation attribute must be 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.

6.8 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; in [CSS2], chapter 4 and appendix D.)
- Allowable data types. (The normative reference is CSS2 syntax and basic data types ([CSS2], chapter 4), with the exception that SVG 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; in [CSS2], section 18.2.)
- 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.)

6.9 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].

6.10 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).

Categories: 'style'
None

Content model:
Any elements or character data.

Attributes:
core attributes
'type'
'media'
'title'

DOM Interfaces:
SVGStyleElement

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 MIME Part Two: Media Types [RFC2046]. If a 'type' is not provided, the value of 'contentStyleType' on the 'svg' element shall be used, which in turn defaults to "text/css" [RFC2046]. If a 'style' element falls outside of the outermost svg element and the 'type' is not provided, the

'type' must default to "text/css" [RFC2046].
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], section 7.3).
Animatable: no.

`title` = ***advisory-title***
(For compatibility with HTML 4 [HTML4].) This attribute specifies an advisory title for the 'style' element.
Animatable: no.

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'. 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).

6.11 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 }
```

6.12 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. Property declarations must follow CSS style rules thus CSS defined properties (e.g. 'font-size') when having a <length> value must include a unit (for non-zero values). See SVG's styling properties for a list of CSS defined properties.

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).

6.13 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` = "***content-type***"

Identifies the default style sheet language for the given document. That language must then be used for all instances of style that do not specify their own style sheet language, such as the 'style' attributes that are available on many elements. The value ***content-type*** specifies a media type, per MIME Part Two: Media Types [RFC2046]. The default value is **"text/css"** [RFC2318].
Animatable: no.

Since the only widely deployed language used for inline styling (in style elements and style attributes) is CSS, and since that is already the default language if contentStyleType is omitted, in practice contentStyleType is not well

supported in user agents. XSL style sheets are typically external. If a new style sheet language becomes popular, it might not use style attributes and could easily declare which language is in use with the type attribute on the style element.

The use of contentStyleType is therefore deprecated; new content should not use it. Future versions of the SVG specification may remove contentStyleType.

6.14 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 the Inheritance section of the CSS2 specification ([CSS2], section 6.2).

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], section 6.1.2). For cases where something other than computed values are inherited, the property definition will describe the inheritance rules. For specified values ([CSS2], section 6.1.1) which are expressed in user units, in pixels (e.g., **20px**) or in absolute values, 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.

6.15 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.

6.16 User agent style sheet

The user agent shall maintain a user agent style sheet ([CSS2], section 6.4) for elements in the SVG namespace for visual media ([CSS2], section 7.3.1). 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 layout ([CSS2], chapter 9).

6.17 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], chapter 19) 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:

Aural property	Definition in [CSS2]
'azimuth'	Section 19.7
'cue'	Section 19.5
'cue-after'	Section 19.5
'cue-before'	Section 19.5
'elevation'	Section 19.7
'pause'	Section 19.4
'pause-after'	Section 19.4
'pause-before'	Section 19.4
'pitch'	Section 19.8
'pitch-range'	Section 19.8
'play-during'	Section 19.6
'richness'	Section 19.8
'speak'	Section 19.3
'speak-header'	Section 17.7.1
'speak-numeral'	Section 19.9
'speak-punctuation'	Section 19.9
'speech-rate'	Section 19.8
'stress'	Section 19.8
'voice-family'	Section 19.8
'volume'	Section 19.2

For user agents that support aural style sheets and also support DOM Level 2 Core [DOM2], the user agent is required to support the DOM interfaces defined in Document Object Model CSS ([DOM2STYLE], chapter 2) that correspond to aural properties. (See Relationship with DOM2 CSS object model.)

6.18 DOM interfaces

6.18.1 Interface SVGStyleElement

The SVGStyleElement interface corresponds to the 'style' element.

```
interface SVGStyleElement : SVGElement,
                            SVGLangSpace {
  attribute DOMString type setraises(DOMException);
  attribute DOMString media setraises(DOMException);
  attribute DOMString title setraises(DOMException);
};
```

Attributes:

- **type** (DOMString)

  Corresponds to attribute 'type' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **media** (DOMString)

  Corresponds to attribute 'media' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **title** (DOMString)

  Corresponds to attribute 'title' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

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], section 7.3.1) 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 ([CSS2], section 4.3.2)

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 [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], section 9.3) 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 specified on the referencing element or on the outermost svg element 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" version="1.1">
      <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 ([CSS2], section 4.3.2) 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 ([CSS2], section 4.3.2).

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300px" height="100px" version="1.1"
     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

7.4 Coordinate system transformations

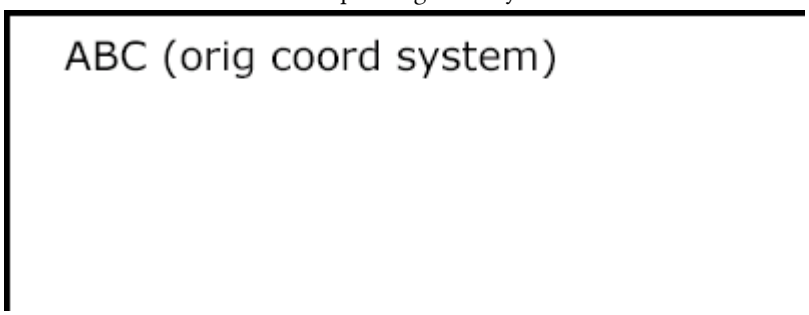
A new user space (i.e., a new current coordinate system) can be established by specifying **transformations** in the form of a 'transform' attribute on a container element or graphics element or a 'viewBox' attribute on an 'svg', 'symbol', 'marker', 'pattern' and the 'view' element. The 'transform' and '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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

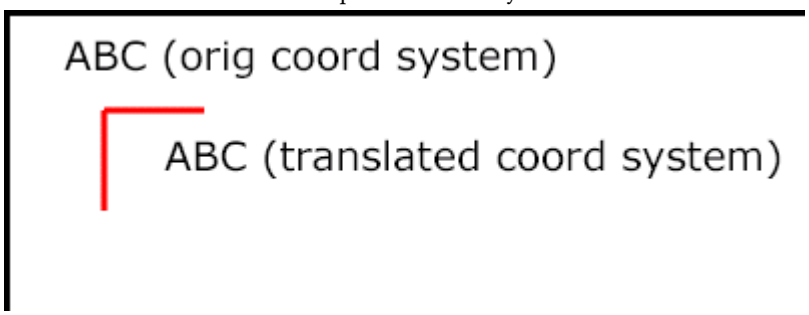
Example OrigCoordSys



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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

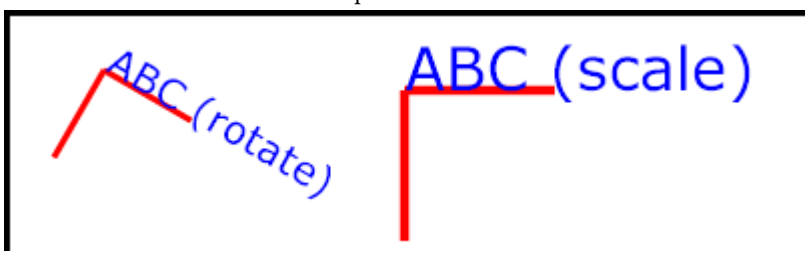


Example RotateScale illustrates simple **rotate** and **scale** transformations. The example defines two new coordinate systems:

- 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="120px" version="1.1"
     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>
```

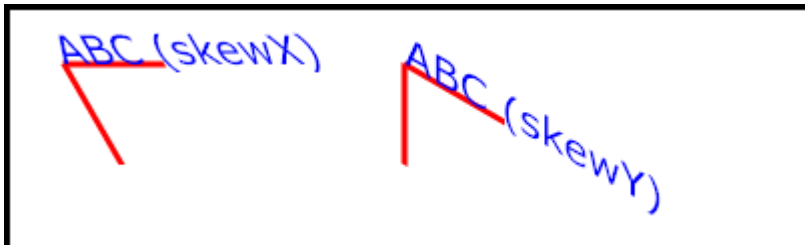
Example RotateScale



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="120px" version="1.1"
     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



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:

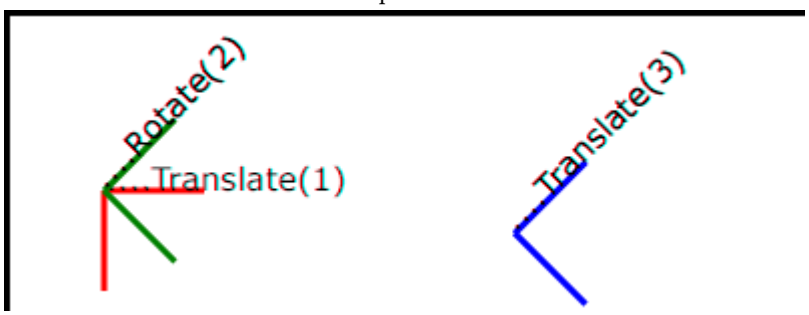
$$CTM = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} a_n & c_n & e_n \\ b_n & d_n & f_n \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{viewport} \\ y_{viewport} \\ 1 \end{bmatrix} = CTM \cdot \begin{bmatrix} x_{userspace} \\ y_{userspace} \\ 1 \end{bmatrix}$$

Example Nested illustrates nested transformations.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="150px" version="1.1"
     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



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:

$$CTM = \text{translate(50,90), rotate(-45), translate(130,160)}$$

$$= \begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 90 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} .707 & .707 & 0 \\ -.707 & .707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 130 \\ 0 & 1 & 160 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} .707 & .707 & 255.03 \\ -.707 & .707 & 111.21 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{initial} \\ y_{initial} \\ 1 \end{bmatrix} = CTM \cdot \begin{bmatrix} x_{userspace} \\ y_{userspace} \\ 1 \end{bmatrix}$$

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 <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.

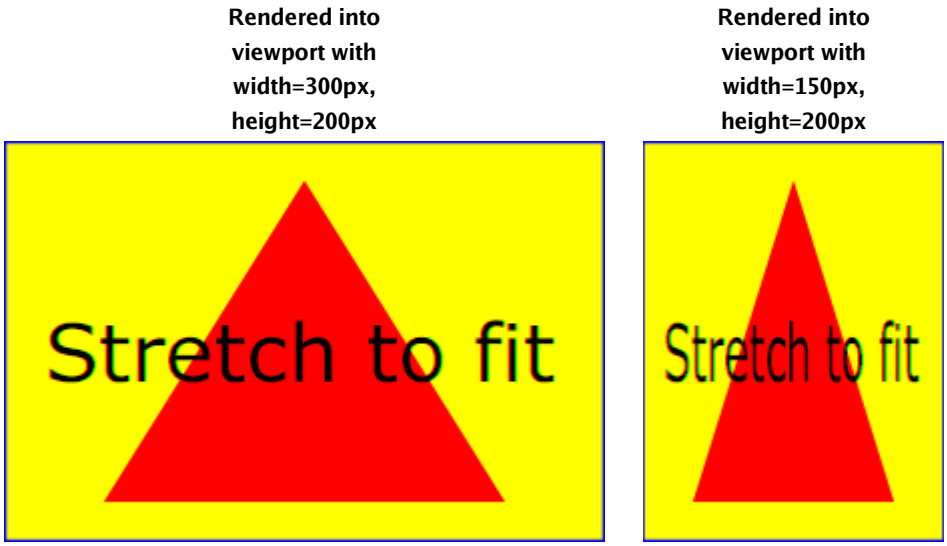
All elements that establish a new viewport (see elements that establish viewports), plus the 'marker', 'pattern' and 'view' elements have attribute 'viewBox'. The value of the 'viewBox' attribute is a list of four numbers **<min-x>**, **<min-y>**, **<width>** and **<height>**, separated by whitespace 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300px" height="200px" version="1.1"
     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>
```

Example ViewBox



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300px" height="200px" version="1.1"
     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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="150px" height="200px" version="1.1"
     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 element, 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="[defer] <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.

If the value of 'preserveAspectRatio' on an 'image' element starts with **'defer'** then the value of the 'preserveAspectRatio' attribute on the referenced content if present should be used. If the referenced content lacks a value for 'preserveAspectRatio' then the 'preserveAspectRatio' attribute should be processed as normal (ignoring **'defer'**). For 'preserveAspectRatio' on all other elements the **'defer'** portion of the attribute is ignored.

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

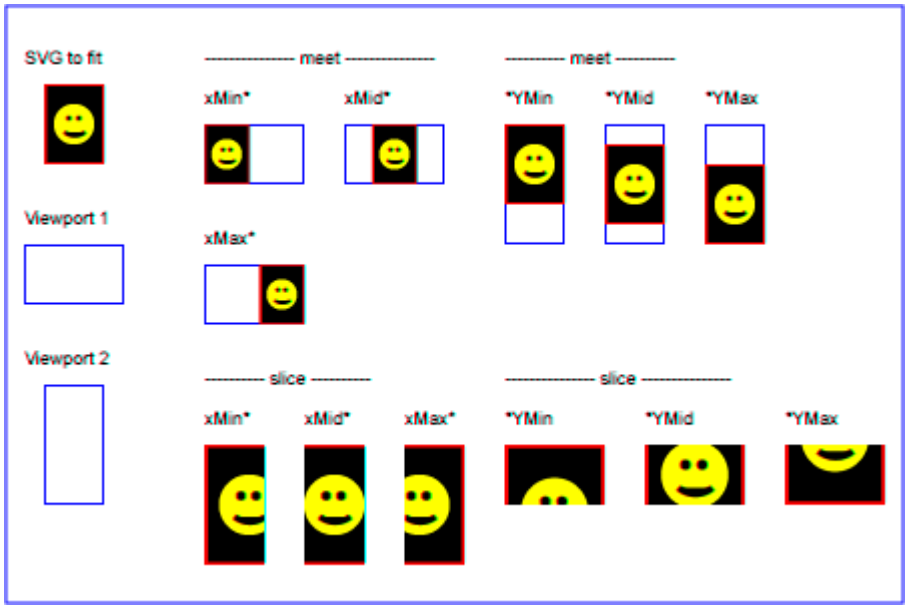
  In this case, if the aspect ratio of the 'viewBox' does not match the viewport, some of the 'viewBox' will extend beyond the bounds of the viewport (i.e., the area into which the 'viewBox' will draw is larger than the viewport).

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.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" version="1.1"
     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



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="3in" version="1.1"
     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 font-size="50">Text size is 50 user units</text>
```

Alternatively, a coordinate or length value can be expressed as a number followed by a unit identifier (e.g., "25cm" or "15em"). (Note that CSS defined properties used in a CSS style sheet or the 'style' attribute require units for non-

zero lengths, see SVG's styling properties.) The list of unit identifiers in SVG matches the list of unit identifiers in CSS: em, ex, px, pt, pc, cm, mm and in. The <length> type can also have a percentage unit identifier. 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 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.

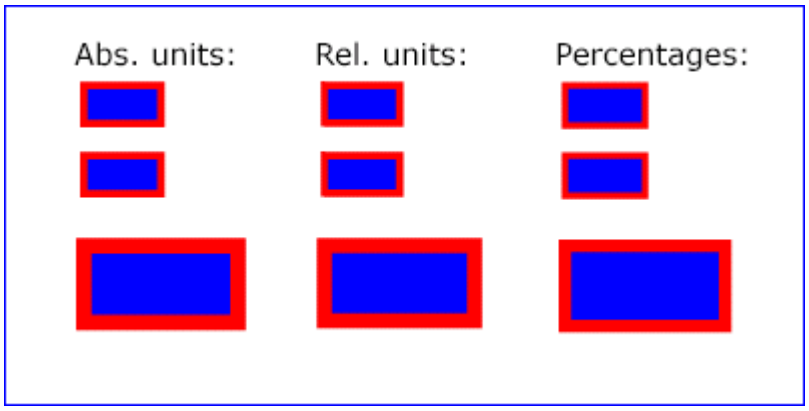
For percentage values that are defined to be relative to the size of viewport:

- 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400px" height="200px" viewBox="0 0 4000 2000"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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**, by setting a specified attribute to '**objectBoundingBox**' on the given element:

Element	Attribute	Effect
'linearGradient'	'gradientUnits'	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'	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'	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'	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'	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'	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'	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'	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'	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 Intrinsic sizing properties of the viewport of SVG content

SVG needs to specify how to calculate some intrinsic sizing properties to enable inclusion within other languages. The intrinsic width and height of the viewport of SVG content must be determined from the 'width' and 'height' attributes. If either of these are not specified, a value of '**100%**' must be assumed. *Note:* the 'width' and 'height' attributes are *not the same* as the CSS width and height properties. Specifically, percentage values *do not provide an intrinsic width or height, and do not indicate a percentage of the containing block*. Rather, once the viewport is established, they indicate the portion of the viewport that is actually covered by image data.

The intrinsic aspect ratio of the viewport of SVG content is necessary for example, when including SVG from

an 'object' element in HTML styled with CSS. It is possible (indeed, common) for an SVG graphic to have an intrinsic aspect ratio but not to have an intrinsic width or height. The intrinsic aspect ratio must be calculated based upon the following rules:

- The aspect ratio is calculated by dividing a width by a height.
- If the 'width' and 'height' of the rootmost 'svg' element are both specified with unit identifiers (in, mm, cm, pt, pc, px, em, ex) or in user units, then the aspect ratio is calculated from the 'width' and 'height' attributes after resolving both values to user units.
- If either/both of the 'width' and 'height' of the rootmost 'svg' element are in percentage units (or omitted), the aspect ratio is calculated from the width and height values of the 'viewBox' specified for the current SVG document fragment. If the 'viewBox' is not correctly specified, or set to '**none**', the intrinsic aspect ratio cannot be calculated and is considered unspecified.

Examples:
Example: Intrinsic Aspect Ratio 1

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="10cm" height="5cm">
  ...
</svg>
```

In this example the intrinsic aspect ratio of the viewport is 2:1. The intrinsic width is 10cm and the intrinsic height is 5cm.
Example: Intrinsic Aspect Ratio 2

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="100%" height="50%" viewBox="0 0 200 200">
  ...
</svg>
```

In this example the intrinsic aspect ratio of the rootmost viewport is 1:1. An aspect ratio calculation in this case allows embedding in an object within a containing block that is only constrained in one direction.
Example: Intrinsic Aspect Ratio 3

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="10cm" viewBox="0 0 200 200">
  ...
</svg>
```

In this case the intrinsic aspect ratio is 1:1.
Example: Intrinsic Aspect Ratio 4

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny"
     width="75%" height="10cm" viewBox="0 0 200 200">
  ...
</svg>
```

In this example, the intrinsic aspect ratio is 1:1.

7.13 Geographic coordinate systems

In order to allow interoperability between SVG content generators and user agents dealing with maps encoded in SVG, the use of a common metadata definition for describing the coordinate system used to generate SVG documents is encouraged.

Such metadata must be added under the 'metadata' element of the topmost 'svg' element describing the map, consisting of an RDF description of the Coordinate Reference System definition used to generate the SVG map [RDF-PRIMER]. Note that the presence of this metadata does not affect the rendering of the SVG in any way; it merely provides added semantic value for applications that make use of combined maps.

The definition must be conformant to the XML grammar described in *GML 3.2.1*, an OpenGIS Standard for encoding common CRS data types in XML [GML]. In order to correctly map the 2-dimensional data used by SVG, the CRS must be of subtype **ProjectedCRS** or **Geographic2dCRS**. The first axis of the described CRS maps the SVG *x*-axis and the second axis maps the SVG *y*-axis.

The main purpose of such metadata is to indicate to the user agent that two or more SVG documents can be overlayed or merged into a single document. Obviously, if two maps reference the same Coordinate Reference System definition and have the same SVG 'transform' attribute value then they can be overlayed without reprojecting the data. If the maps reference different Coordinate Reference Systems and/or have different SVG 'transform' attribute values, then a specialized cartographic user agent may choose to transform the coordinate data to overlay the data. However, typical SVG user agents are not required to perform these types of transformations, or even recognize the metadata. It is described in this specification so that the connection between geographic coordinate systems and the SVG coordinate system is clear.

7.14 The 'svg:transform' attribute

Attribute definition:

```
svg:transform = "<transform>" | "none"
```

<transform>
: Specifies the affine transformation that has been applied to the map data. The syntax is identical to that described in The 'transform' attribute section.

none
: Specifies that no supplemental affine transformation has been applied to the map data. Using this value has the same meaning as specifying the identity matrix, which in turn is just the same as not specifying the 'svg:transform' the attribute at all.

Animatable: no.

This attribute describes an optional additional affine transformation that may have been applied during this mapping. This attribute may be added to the OpenGIS 'CoordinateReferenceSystem' element. Note that, unlike the

'transform' attribute, it does not indicate that a transformation is to *be applied* to the data within the file. Instead, it simply describes the transformation that *was already applied* to the data when being encoded in SVG.

There are three typical uses for the 'svg:transform' global attribute. These are described below and used in the examples.

- Most ProjectedCRS have the north direction represented by positive values of the second axis and conversely SVG has a *y*-down coordinate system. That's why, in order to follow the usual way to represent a map with the north at its top, it is recommended for that kind of ProjectedCRS to use the 'svg:transform' global attribute with a **'scale(1, -1)'** value as in the third example below.
- Most Geographic2dCRS have the latitude as their first axis rather than the longitude, which means that the south-north axis would be represented by the *x*-axis in SVG instead of the usual *y*-axis. That's why, in order to follow the usual way to represent a map with the north at its top, it is recommended for that kind of Geographic2dCRS to use the 'svg:transform' global attribute with a **'rotate(-90)'** value as in the first example (while also adding the **'scale(1, -1)'** as for ProjectedCRS).
- In addition, when converting for profiles which place restrictions on precision of real number values, it may be useful to add an additional scaling factor to retain good precision for a specific area. When generating an SVG document from WGS84 geographic coordinates (EPGS 4326), we recommend the use of an additional 100 times scaling factor corresponding to an 'svg:transform' global attribute with a **'rotate(-90) scale(100)'** value (shown in the second example). Different scaling values may be required depending on the particular CRS.

Below is a simple example of the coordinate metadata, which describes the coordinate system used by the document via a URI.

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="100" height="100" viewBox="0 0 1000 1000">

  <desc>An example that references coordinate data.</desc>

  <metadata>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:crs="http://www.ogc.org/crs"
             xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description rdf:about="">
        <!-- The Coordinate Reference System is described
             through a URI. -->
        <crs:CoordinateReferenceSystem
            svg:transform="rotate(-90)"
            rdf:resource="http://www.example.org/srs/epsg.xml#4326"/>
      </rdf:Description>
    </rdf:RDF>
  </metadata>

  <!-- The actual map content -->
</svg>
```

The second example uses a well-known identifier to describe the coordinate system. Note that the coordinates used in the document have had the supplied transform applied.

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="100" height="100" viewBox="0 0 1000 1000">

  <desc>Example using a well known coordinate system.</desc>

  <metadata>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
            xmlns:crs="http://www.ogc.org/crs"
            xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description rdf:about="">
        <!-- In case of a well-known Coordinate Reference System
             an 'Identifier' is enough to describe the CRS -->
        <crs:CoordinateReferenceSystem svg:transform="rotate(-90) scale(100, 100)">
          <crs:Identifier>
            <crs:code>4326</crs:code>
            <crs:codeSpace>EPSG</crs:codeSpace>
            <crs:edition>5.2</crs:edition>
          </crs:Identifier>
        </crs:CoordinateReferenceSystem>
      </rdf:Description>
    </rdf:RDF>
  </metadata>

  <!-- The actual map content -->
</svg>
```

The third example defines the coordinate system completely within the SVG document.

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="100" height="100" viewBox="0 0 1000 1000">

  <desc>Coordinate metadata defined within the SVG document</desc>

  <metadata>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:crs="http://www.ogc.org/crs"
             xmlns:svg="http://www.w3.org/2000/svg">
      <rdf:Description rdf:about="">
        <!-- For other CRS it should be entirely defined -->
        <crs:CoordinateReferenceSystem svg:transform="scale(1,-1)">
          <crs:NameSet>
            <crs:name>Mercator projection of WGS84</crs:name>
          </crs:NameSet>
          <crs:ProjectedCRS>
            <!-- The actual definition of the CRS -->
            <crs:CartesianCoordinateSystem>
              <crs:dimension>2</crs:dimension>
              <crs:CoordinateAxis>
                <crs:axisDirection>north</crs:axisDirection>
                <crs:AngularUnit>
                  <crs:Identifier>
                    <crs:code>9108</crs:code>
                    <crs:codeSpace>EPSG</crs:codeSpace>
                    <crs:edition>5.2</crs:edition>
                  </crs:Identifier>
                </crs:AngularUnit>
              </crs:CoordinateAxis>
              <crs:CoordinateAxis>
                <crs:axisDirection>east</crs:axisDirection>
                <crs:AngularUnit>
                  <crs:Identifier>
                    <crs:code>9108</crs:code>
                    <crs:codeSpace>EPSG</crs:codeSpace>
                    <crs:edition>5.2</crs:edition>
                  </crs:Identifier>
                </crs:AngularUnit>
              </crs:CoordinateAxis>
            </crs:CartesianCoordinateSystem>
            <crs:CoordinateReferenceSystem>
              <!-- the reference system of that projected system is
                        WGS84 which is EPSG 4326 in EPSG codeSpace -->
              <crs:NameSet>
                <crs:name>WGS 84</crs:name>
              </crs:NameSet>
              <crs:Identifier>
                <crs:code>4326</crs:code>
                <crs:codeSpace>EPSG</crs:codeSpace>
                <crs:edition>5.2</crs:edition>
              </crs:Identifier>
            </crs:CoordinateReferenceSystem>
            <crs:CoordinateTransformationDefinition>
```

```
            <crs:sourceDimensions>2</crs:sourceDimensions>
            <crs:targetDimensions>2</crs:targetDimensions>
            <crs:ParameterizedTransformation>
              <crs:TransformationMethod>
                <!-- the projection is a Mercator projection which is
                      EPSG 9805 in EPSG codeSpace -->
                <crs:NameSet>
                  <crs:name>Mercator</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>9805</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:description>Mercator (2SP)</crs:description>
              </crs:TransformationMethod>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>Latitude of 1st standart parallel</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8823</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>Longitude of natural origin</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8802</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>False Easting</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8806</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
              <crs:Parameter>
                <crs:NameSet>
                  <crs:name>False Northing</crs:name>
                </crs:NameSet>
                <crs:Identifier>
                  <crs:code>8807</crs:code>
                  <crs:codeSpace>EPSG</crs:codeSpace>
                  <crs:edition>5.2</crs:edition>
                </crs:Identifier>
                <crs:value>0</crs:value>
              </crs:Parameter>
            </crs:ParameterizedTransformation>
          </crs:CoordinateTransformationDefinition>
        </crs:ProjectedCRS>
      </crs:CoordinateReferenceSystem>
    </rdf:Description>
  </rdf:RDF>
</metadata>

<!-- the actual map content -->
</svg>
```

7.15 DOM interfaces

7.15.1 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]
```

If an SVGRect object is designated as *read only*, then attempting to assign to one of its attributes will result in an exception being thrown.

```
interface SVGPoint {

  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);

  SVGPoint matrixTransform(in SVGMatrix matrix);
};
```

Attributes:

- **x** (float)

  The x coordinate.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised if the SVGPoint object is read only, or corresponds to a DOM attribute that is read only.

- **y** (float)

  The y coordinate.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised if the SVGPoint object is read only, or corresponds to a DOM attribute that is read only.

Operations:

- SVGPoint **matrixTransform**(in SVGMatrix *matrix*)

  Applies a 2x3 matrix transformation on this SVGPoint object and returns a new, transformed SVGPoint object:

  ```
  newpoint = matrix * thispoint
  ```

  Parameters

  - SVGMatrix *matrix*
    The matrix which is to be applied to this SVGPoint object.

  Returns
  A new SVGPoint object.

7.15.2 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.

```
interface SVGPointList {

  readonly attribute unsigned long numberOfItems;

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

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.

- SVGPoint **initialize**(in SVGPoint *newItem*)

  Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter. If the inserted item is already in a list, it is removed from its previous list before it is inserted into this list. The inserted item is the item itself and not a copy.

  Parameters

  - SVGPoint *newItem*
    The item which should become the only member of the list.

  Returns
  The item being inserted into the list.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- SVGPoint **getItem**(in unsigned long *index*)

  Returns the specified item from the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

  Parameters

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

  Returns
  The selected item.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGPoint **insertItemBefore**(in SVGPoint *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to insert before is *before* the removal of the item.

  Parameters

  - SVGPoint *newItem*
    The item which is to be inserted into the list.
  - 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.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- SVGPoint **replaceItem**(in SVGPoint *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to replace is *before* the removal of the item.

  Parameters

  - SVGPoint *newItem*
    The item which is to be inserted into the list.
  - unsigned long *index*
    The index of the item which is to be replaced. The first item is number 0.

  Returns
  The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.
- DOMException, code INDEX_SIZE_ERR
  Raised if the index number is greater than or equal to numberOfItems.

- SVGPoint **removeItem**(in unsigned long *index*)

  Removes an existing item from the list.

  Parameters

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

  Returns
  The removed item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGPoint **appendItem**(in SVGPoint *newItem*)

  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. The inserted item is the item itself and not a copy.

  Parameters

  - SVGPoint *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
  The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.

7.15.3 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]
```

```
interface SVGMatrix {

  attribute float a setraises(DOMException);
  attribute float b setraises(DOMException);
  attribute float c setraises(DOMException);
  attribute float d setraises(DOMException);
  attribute float e setraises(DOMException);
  attribute float f setraises(DOMException);

  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:

- **a** (float)

  The *a* component of the matrix.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **b** (float)

  The *b* component of the matrix.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **c** (float)

  The *c* component of the matrix.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **d** (float)

  The *d* component of the matrix.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **e** (float)

  The *e* component of the matrix.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **f** (float)

  The *f* component of the matrix.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

Operations:

- SVGMatrix **multiply**(in SVGMatrix *secondMatrix*)

  Performs matrix multiplication. This matrix is post-multiplied by another matrix, returning the resulting new matrix.

  Parameters

  - SVGMatrix *secondMatrix*
    The matrix which is post-multiplied to this matrix.

  Returns
  The resulting matrix.

- SVGMatrix **inverse**()

  Returns the inverse matrix.

  Returns
  The inverse matrix.

  Exceptions

  - SVGException, code SVG_MATRIX_NOT_INVERTABLE
    Raised if this matrix is not invertable.

- SVGMatrix **translate**(in float *x*, in float *y*)

  Post-multiplies a translation transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *x*
    The distance to translate along the x-axis.
  - float *y*
    The distance to translate along the y-axis.

Returns
The resulting matrix.

- SVGMatrix **scale**(in float *scaleFactor*)

  Post-multiplies a uniform scale transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *scaleFactor*
    Scale factor in both X and Y.

  Returns
  The resulting matrix.

- SVGMatrix **scaleNonUniform**(in float *scaleFactorX*, in float *scaleFactorY*)

  Post-multiplies a non-uniform scale transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *scaleFactorX*
    Scale factor in X.
  - float *scaleFactorY*
    Scale factor in Y.

  Returns
  The resulting matrix.

- SVGMatrix **rotate**(in float *angle*)

  Post-multiplies a rotation transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *angle*
    Rotation angle.

  Returns
  The resulting matrix.

- SVGMatrix **rotateFromVector**(in float *x*, in float *y*)

  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

  - float *x*
    The X coordinate of the vector (x,y). Must not be zero.
  - float *y*
    The Y coordinate of the vector (x,y). Must not be zero.

  Returns
  The resulting matrix.

  Exceptions

  - SVGException, code SVG_INVALID_VALUE_ERR
    Raised if one of the parameters has an invalid value.

- SVGMatrix **flipX**()

  Post-multiplies the transformation [-1 0 0 1 0 0] and returns the resulting matrix.

  Returns
  The resulting matrix.

- SVGMatrix **flipY**()

  Post-multiplies the transformation [1 0 0 -1 0 0] and returns the resulting matrix.

  Returns
  The resulting matrix.

- SVGMatrix **skewX**(in float *angle*)

  Post-multiplies a skewX transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *angle*
    Skew angle.

Returns
The resulting matrix.

- SVGMatrix **skewY**(in float *angle*)

  Post-multiplies a skewY transformation on the current matrix and returns the resulting matrix.

  Parameters

  - float *angle*
    Skew angle.

  Returns
  The resulting matrix.

7.15.4 Interface SVGTransform

SVGTransform is the interface for one of the component transformations within an SVGTransformList; thus, an SVGTransform object corresponds to a single component (e.g., **'scale(...)'** or **'matrix(...)'**) within a 'transform' attribute specification.

```
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) raises(DOMException);
  void setTranslate(in float tx, in float ty) raises(DOMException);
  void setScale(in float sx, in float sy) raises(DOMException);
  void setRotate(in float angle, in float cx, in float cy) raises(DOMException);
  void setSkewX(in float angle) raises(DOMException);
  void setSkewY(in float angle) raises(DOMException);
};
```

Constants in group "Transform Types":

- **SVG_TRANSFORM_UNKNOWN** (unsigned short)

  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** (unsigned short)

  A '**matrix(**...**)**' transformation.

- **SVG_TRANSFORM_TRANSLATE** (unsigned short)

  A '**translate(**...**)**' transformation.

- **SVG_TRANSFORM_SCALE** (unsigned short)

  A '**scale(**...**)**' transformation.

- **SVG_TRANSFORM_ROTATE** (unsigned short)

  A '**rotate(**...**)**' transformation.

- **SVG_TRANSFORM_SKEWX** (unsigned short)

  A '**skewX(**...**)**' transformation.

- **SVG_TRANSFORM_SKEWY** (unsigned short)

  A '**skewY(**...**)**' transformation.

Attributes:

- **type** (readonly unsigned short)

  The type of the value as specified by one of the SVG_TRANSFORM_* constants defined on this interface.

- **matrix** (readonly SVGMatrix)

  The matrix that represents this transformation. The matrix object is live, meaning that any changes made to the SVGTransform object are immediately reflected in the matrix object and vice versa. In case the matrix object is changed directly (i.e., without using the methods on the SVGTransform interface itself) then the type of the SVGTransform changes to SVG_TRANSFORM_MATRIX.

  - 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 and d=1).
  - For SVG_TRANSFORM_SCALE, a and d represent the scale amounts (b=0, c=0, e=0 and f=0).
  - For SVG_TRANSFORM_SKEWX and SVG_TRANSFORM_SKEWY, a, b, c and d represent the matrix which will result in the given skew (e=0 and f=0).

- For SVG_TRANSFORM_ROTATE, a, b, c, d, e and f together represent the matrix which will result in the given rotation. When the rotation is around the center point (0, 0), e and f will be zero.

- **angle** (readonly float)

  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.

Operations:

- void **setMatrix**(in SVGMatrix *matrix*)

  Sets the transform type to SVG_TRANSFORM_MATRIX, with parameter *matrix* defining the new transformation. The values from the parameter *matrix* are copied, the *matrix* parameter does not replace SVGTransform::matrix.

  Parameters

  - SVGMatrix *matrix*
    The new matrix for the transformation.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- void **setTranslate**(in float *tx*, in float *ty*)

  Sets the transform type to SVG_TRANSFORM_TRANSLATE, with parameters *tx* and *ty* defining the translation amounts.

  Parameters

  - float *tx*
    The translation amount in X.

  - float *ty*
    The translation amount in Y.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- void **setScale**(in float *sx*, in float *sy*)

  Sets the transform type to SVG_TRANSFORM_SCALE, with parameters *sx* and *sy* defining the scale amounts.

  Parameters

  - float *sx*
    The scale amount in X.
  - float *sy*
    The scale amount in Y.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- void **setRotate**(in float *angle*, in float *cx*, in float *cy*)

  Sets the transform type to SVG_TRANSFORM_ROTATE, with parameter *angle* defining the rotation angle and parameters *cx* and *cy* defining the optional center of rotation.

  Parameters

  - float *angle*
    The rotation angle.
  - float *cx*
    The x coordinate of center of rotation.
  - float *cy*
    The y coordinate of center of rotation.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- void **setSkewX**(in float *angle*)

  Sets the transform type to SVG_TRANSFORM_SKEWX, with parameter *angle* defining the amount of skew.

  Parameters

  - float *angle*
    The skew angle.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- void **setSkewY**(in float *angle*)

  Sets the transform type to SVG_TRANSFORM_SKEWY, with parameter *angle* defining the amount of skew.

  Parameters

  - float *angle*
    The skew angle.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

7.15.5 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.

An SVGTransformList object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
interface SVGTransformList {

  readonly attribute unsigned long numberOfItems;

  void clear() raises(DOMException);
  SVGTransform initialize(in SVGTransform newItem) raises(DOMException);
  SVGTransform getItem(in unsigned long index) raises(DOMException);
  SVGTransform insertItemBefore(in SVGTransform newItem, in unsigned long index) raises(DOMException);
```

```
  SVGTransform replaceItem(in SVGTransform newItem, in unsigned long index) raises(DOMException);
  SVGTransform removeItem(in unsigned long index) raises(DOMException);
  SVGTransform appendItem(in SVGTransform newItem) raises(DOMException);
  SVGTransform createSVGTransformFromMatrix(in SVGMatrix matrix);
  SVGTransform consolidate() raises(DOMException);
};
```

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGTransform **initialize**(in SVGTransform *newItem*)

  Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter. If the inserted item is already in a list, it is removed from its previous list before it is inserted into this list. The inserted item is the item itself and not a copy.

  Parameters

  - SVGTransform *newItem*
    The item which should become the only member of the list.

  Returns
  The item being inserted into the list.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGTransform **getItem**(in unsigned long *index*)

  Returns the specified item from the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

  Parameters

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

  Returns
  The selected item.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGTransform **insertItemBefore**(in SVGTransform *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to insert before is *before* the removal of the item.

  Parameters

  - SVGTransform *newItem*
    The item which is to be inserted into the list.

  - 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.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGTransform **replaceItem**(in SVGTransform *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to replace is *before* the removal of the item.

  Parameters

  - SVGTransform *newItem*
    The item which is to be inserted into the list.
  - unsigned long *index*
    The index of the item which is to be replaced. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGTransform **removeItem**(in unsigned long *index*)

  Removes an existing item from the list.

  Parameters

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

  Returns
  The removed item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGTransform **appendItem**(in SVGTransform *newItem*)

  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. The inserted item is the item itself and not a copy.

  Parameters

  - SVGTransform *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list corresponds to a read only attribute or when the object itself is read only.

- SVGTransform **createSVGTransformFromMatrix**(in SVGMatrix *matrix*)

  Creates an SVGTransform object which is initialized to transform of type SVG_TRANSFORM_MATRIX and whose values are the given matrix. The values from the parameter *matrix* are copied, the *matrix* parameter is not adopted as SVGTransform::matrix.

  Parameters

  - SVGMatrix *matrix*
    The matrix which defines the transformation.

  Returns
  The returned SVGTransform object.

- SVGTransform **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. The consolidation operation creates new SVGTransform object as the first and only item in the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

  Returns
  The resulting SVGTransform object which becomes single item in the list. If the list was empty, then a value of null is returned.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list corresponds to a read only attribute or when the object itself is read only.

7.15.6 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.

```
interface SVGAnimatedTransformList {
  readonly attribute SVGTransformList baseVal;
  readonly attribute SVGTransformList animVal;
};
```

Attributes:

- **baseVal** (readonly SVGTransformList)

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

- **animVal** (readonly SVGTransformList)

  A read only SVGTransformList representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGTransformList will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

7.15.7 Interface SVGPreserveAspectRatio

The SVGPreserveAspectRatio interface corresponds to the 'preserveAspectRatio' attribute, which is available for some of SVG's elements.
An SVGPreserveAspectRatio object can be designated as *read only*, which means that attempts to modify the object will result in an exception being thrown, as described below.

```
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 setraises(DOMException);
  attribute unsigned short meetOrSlice setraises(DOMException);
};
```

Constants in group "Alignment Types":

- **SVG_PRESERVEASPECTRATIO_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value '**none**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMINYMIN** (unsigned short)

  Corresponds to value '**xMinYMin**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMIDYMIN** (unsigned short)

  Corresponds to value '**xMidYMin**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMAXYMIN** (unsigned short)

  Corresponds to value '**xMaxYMin**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMINYMID** (unsigned short)

  Corresponds to value '**XMinYMid**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMIDYMID** (unsigned short)

  Corresponds to value '**xMidYMid**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMAXYMID** (unsigned short)

  Corresponds to value '**xMaxYMid**' for attribute '**preserveAspectRatio**'.

- **SVG_PRESERVEASPECTRATIO_XMINYMAX** (unsigned short)

  Corresponds to value **'xMinYMax'** for attribute **'preserveAspectRatio'**.

- **SVG_PRESERVEASPECTRATIO_XMIDYMAX** (unsigned short)

  Corresponds to value **'xMidYMax'** for attribute **'preserveAspectRatio'**.

- **SVG_PRESERVEASPECTRATIO_XMAXYMAX** (unsigned short)

  Corresponds to value **'xMaxYMax'** for attribute **'preserveAspectRatio'**.

Constants in group "Meet-or-slice Types":

- **SVG_MEETORSLICE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'meet'** for attribute **'preserveAspectRatio'**.

- **SVG_MEETORSLICE_SLICE** (unsigned short)

  Corresponds to value **'slice'** for attribute **'preserveAspectRatio'**.

Attributes:

- **align** (unsigned short)

  The type of the alignment value as specified by one of the SVG_PRESERVEASPECTRATIO_* constants defined on this interface.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the object corresponds to a read only attribute or when the object itself is read only.

- **meetOrSlice** (unsigned short)

  The type of the meet-or-slice value as specified by one of the SVG_MEETORSLICE_* constants defined on this interface.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the object corresponds to a read only attribute or when the object itself is read only.

7.15.8 Interface SVGAnimatedPreserveAspectRatio

Used for attributes of type SVGPreserveAspectRatio which can be animated.

```
interface SVGAnimatedPreserveAspectRatio {
  readonly attribute SVGPreserveAspectRatio baseVal;
  readonly attribute SVGPreserveAspectRatio animVal;
};
```

Attributes:

- **baseVal** (readonly SVGPreserveAspectRatio)

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

- **animVal** (readonly SVGPreserveAspectRatio)

  A read only SVGPreserveAspectRatio representing the current animated value of the given attribute. If the given attribute is not currently being animated, then the SVGPreserveAspectRatio will have the same contents as baseVal. The object referenced by animVal will always be distinct from the one referenced by baseVal, even when the attribute is not animated.

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

Categories: 'path'

Graphics element, shape element

Content model:

Any number of the following elements, in any order:

animation elements

descriptive elements

Attributes:

conditional processing attributes

core attributes

graphical event attributes

presentation attributes

'class'

'style'

'externalResourcesRequired'

'transform'

'd'

'pathLength'

DOM Interfaces:

SVGPathElement

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.

8.3 Path data

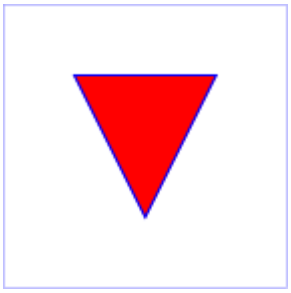
8.3.1 General information about path data

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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 U+0046 FULL STOP (".") character (also referred to in Unicode as PERIOD, dot and decimal point) and no other delimiter characters are allowed [UNICODE]. (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 (if there is one) 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) **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 moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands. Hence, implicit lineto commands will be relative if the moveto is relative, and absolute if the moveto is absolute. If a relative moveto (**m**) appears as the first element of the path, then it is treated as a pair of absolute coordinates. In this case, subsequent pairs of coordinates are treated as relative even though the initial moveto is interpreted as an absolute moveto.

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. Since the Z and z commands take no parameters, they have an identical effect.

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:

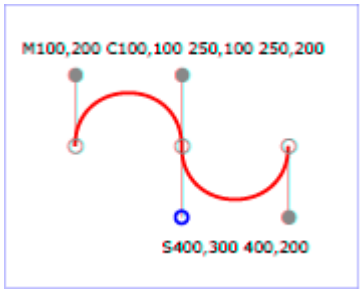
Command	Name	Parameters	Description
C (absolute) **c** (relative)	curveto	(x1 y1 x2 y2 x y)+	Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. **C** (uppercase) indicates 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) **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.

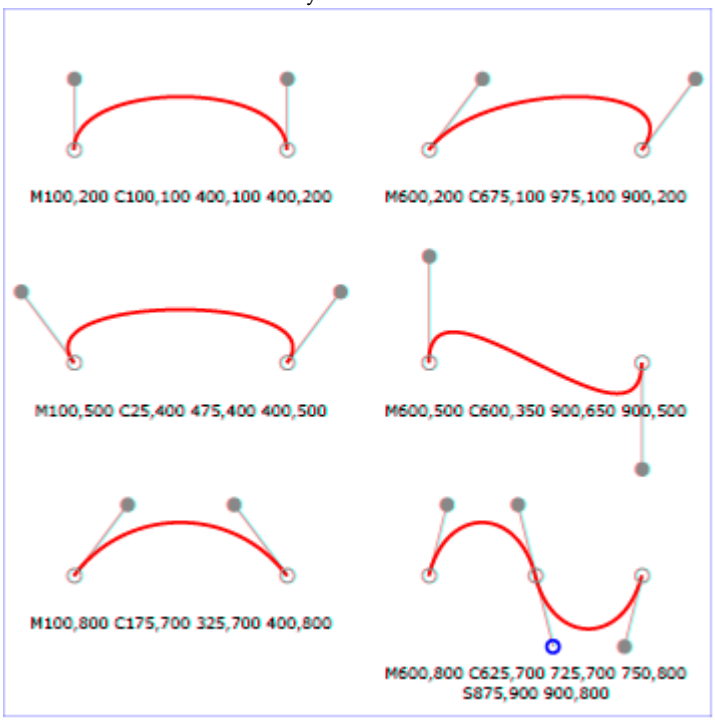
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" viewBox="0 0 500 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <title>Example cubic01- cubic Bézier commands in 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



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.



8.3.7 The quadratic Bézier curve commands

The quadratic Bézier commands are as follows:

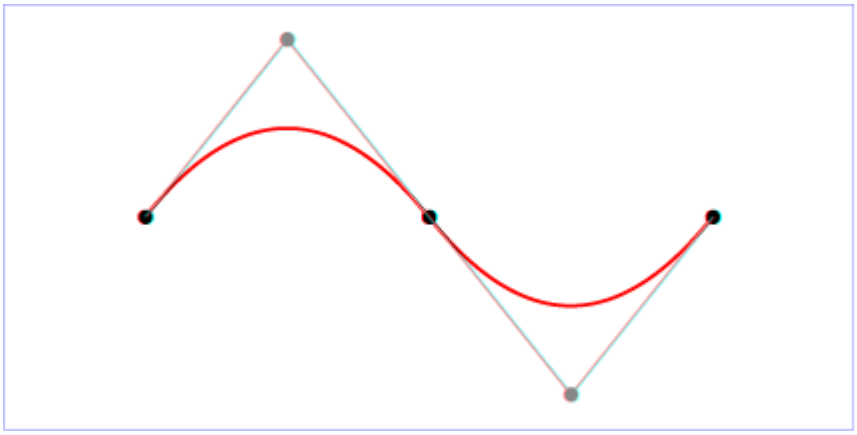
Command	Name	Parameters	Description
Q (absolute) **q** (relative)	quadratic Bézier curveto	(x1 y1 x y)+	Draws a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point. **Q** (uppercase) indicates that absolute coordinates will follow; **q** (lowercase) indicates that 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) **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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="6cm" viewBox="0 0 1200 600"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <title>Example quad01 - quadratic Bézier 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



8.3.8 The elliptical arc curve commands

The elliptical arc commands are as follows:

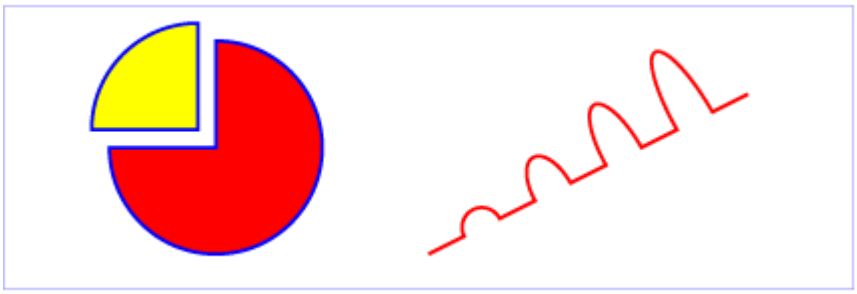
Command	Name	Parameters	Description
A (absolute) **a** (relative)	elliptical arc	(rx ry x-axis-rotation large-arc-flag sweep-flag x y)+	Draws an elliptical arc from the current point to (**x**, **y**). The size and orientation of the ellipse are defined by two radii (**rx**, **ry**) and an **x-axis-rotation**, 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="5.25cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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.

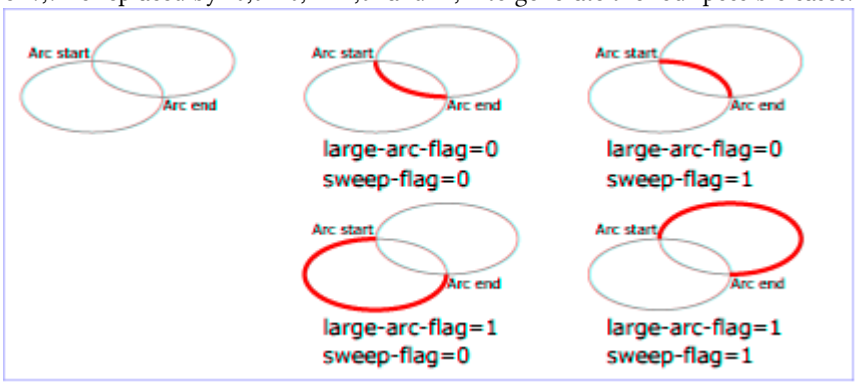
For most situations, there are actually four different arcs (two different ellipses, each with two different arc sweeps) that satisfy these constraints. **large-arc-flag** and **sweep-flag** indicate which one of the four arcs are drawn, as follows:

- 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.



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".

Note that the BNF allows the path 'd' attribute to be empty. This is not an error, instead it disables rendering of the path.

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

8.5.1 Interface SVGPathSeg

The SVGPathSeg interface is a base interface that corresponds to a single command within a path data specification.

```
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;
};
```

Constants in group “Path Segment Types”:

- **PATHSEG_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to a "closepath" (z) path data command.

- **PATHSEG_MOVETO_ABS** (unsigned short)

  Corresponds to a "absolute moveto" (M) path data command.

- **PATHSEG_MOVETO_REL** (unsigned short)

  Corresponds to a "relative moveto" (m) path data command.

- **PATHSEG_LINETO_ABS** (unsigned short)

  Corresponds to a "absolute lineto" (L) path data command.

- **PATHSEG_LINETO_REL** (unsigned short)

  Corresponds to a "relative lineto" (l) path data command.

- **PATHSEG_CURVETO_CUBIC_ABS** (unsigned short)

  Corresponds to a "absolute cubic Bézier curveto" (C) path data command.

- **PATHSEG_CURVETO_CUBIC_REL** (unsigned short)

  Corresponds to a "relative cubic Bézier curveto" (c) path data command.

- **PATHSEG_CURVETO_QUADRATIC_ABS** (unsigned short)

  Corresponds to a "absolute quadratic Bézier curveto" (Q) path data command.

- **PATHSEG_CURVETO_QUADRATIC_REL** (unsigned short)

  Corresponds to a "relative quadratic Bézier curveto" (q) path data command.

- **PATHSEG_ARC_ABS** (unsigned short)

  Corresponds to a "absolute arcto" (A) path data command.

- **PATHSEG_ARC_REL** (unsigned short)

  Corresponds to a "relative arcto" (a) path data command.

- **PATHSEG_LINETO_HORIZONTAL_ABS** (unsigned short)

  Corresponds to a "absolute horizontal lineto" (H) path data command.

- **PATHSEG_LINETO_HORIZONTAL_REL** (unsigned short)

  Corresponds to a "relative horizontal lineto" (h) path data command.

- **PATHSEG_LINETO_VERTICAL_ABS** (unsigned short)

  Corresponds to a "absolute vertical lineto" (V) path data command.

- **PATHSEG_LINETO_VERTICAL_REL** (unsigned short)

  Corresponds to a "relative vertical lineto" (v) path data command.

- **PATHSEG_CURVETO_CUBIC_SMOOTH_ABS** (unsigned short)

  Corresponds to a "absolute smooth cubic curveto" (S) path data command.

- **PATHSEG_CURVETO_CUBIC_SMOOTH_REL** (unsigned short)

  Corresponds to a "relative smooth cubic curveto" (s) path data command.

- **PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS** (unsigned short)

  Corresponds to a "absolute smooth quadratic curveto" (T) path data command.

- **PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL** (unsigned short)

  Corresponds to a "relative smooth quadratic curveto" (t) path data command.

Attributes:

- **pathSegType** (readonly unsigned short)

  The type of the path segment as specified by one of the constants defined on this interface.

- **pathSegTypeAsLetter** (readonly DOMString)

  The type of the path segment, specified by the corresponding one character command name.

8.5.2 Interface SVGPathSegClosePath

The SVGPathSegClosePath interface corresponds to a "closepath" (z) path data command.

```
interface SVGPathSegClosePath : SVGPathSeg {
};
```

8.5.3 Interface SVGPathSegMovetoAbs

The SVGPathSegMovetoAbs interface corresponds to an "absolute moveto" (M) path data command.

```
interface SVGPathSegMovetoAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.4 Interface SVGPathSegMovetoRel

The SVGPathSegMovetoRel interface corresponds to a "relative moveto" (m) path data command.

```
interface SVGPathSegMovetoRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.5 Interface SVGPathSegLinetoAbs

The SVGPathSegLinetoAbs interface corresponds to an "absolute lineto" (L) path data command.

```
interface SVGPathSegLinetoAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.6 Interface SVGPathSegLinetoRel

The SVGPathSegLinetoRel interface corresponds to a "relative lineto" (l) path data command.

```
interface SVGPathSegLinetoRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.7 Interface SVGPathSegCurvetoCubicAbs

The SVGPathSegCurvetoCubicAbs interface corresponds to an "absolute cubic Bézier curveto" (C) path data command.

```
interface SVGPathSegCurvetoCubicAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x1** (float)

  The absolute X coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y1** (float)

  The absolute Y coordinate for the first control point.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **x2** (float)

  The absolute X coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y2** (float)

  The absolute Y coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.8 Interface SVGPathSegCurvetoCubicRel

The SVGPathSegCurvetoCubicRel interface corresponds to a "relative cubic Bézier curveto" (c) path data command.

```
interface SVGPathSegCurvetoCubicRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x1** (float)

  The relative X coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y1** (float)

  The relative Y coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x2** (float)

  The relative X coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y2** (float)

  The relative Y coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.9 Interface SVGPathSegCurvetoQuadraticAbs

The SVGPathSegCurvetoQuadraticAbs interface corresponds to an "absolute quadratic Bézier curveto" (Q) path data command.

```
interface SVGPathSegCurvetoQuadraticAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x1** (float)

  The absolute X coordinate for the first control point.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **y1** (float)

  The absolute Y coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.10 Interface SVGPathSegCurvetoQuadraticRel

The SVGPathSegCurvetoQuadraticRel interface corresponds to a "relative quadratic Bézier curveto" (q) path data command.

```
interface SVGPathSegCurvetoQuadraticRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x1** (float)

  The relative X coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y1** (float)

  The relative Y coordinate for the first control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.11 Interface SVGPathSegArcAbs

The SVGPathSegArcAbs interface corresponds to an "absolute arcto" (A) path data command.

```
interface SVGPathSegArcAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float r1 setraises(DOMException);
  attribute float r2 setraises(DOMException);
  attribute float angle setraises(DOMException);
  attribute boolean largeArcFlag setraises(DOMException);
  attribute boolean sweepFlag setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **r1** (float)

  The x-axis radius for the ellipse (i.e., r1).

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **r2** (float)

  The y-axis radius for the ellipse (i.e., r2).

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **angle** (float)

  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, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **largeArcFlag** (boolean)

  The value of the large-arc-flag parameter.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **sweepFlag** (boolean)

  The value of the sweep-flag parameter.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.12 Interface SVGPathSegArcRel

The SVGPathSegArcRel interface corresponds to a "relative arcto" (a) path data command.

```
interface SVGPathSegArcRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float r1 setraises(DOMException);
  attribute float r2 setraises(DOMException);
  attribute float angle setraises(DOMException);
  attribute boolean largeArcFlag setraises(DOMException);
  attribute boolean sweepFlag setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **r1** (float)

  The x-axis radius for the ellipse (i.e., r1).

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **r2** (float)

  The y-axis radius for the ellipse (i.e., r2).

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **angle** (float)

  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, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **largeArcFlag** (boolean)

  The value of the large-arc-flag parameter.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **sweepFlag** (boolean)

  The value of the sweep-flag parameter.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.13 Interface SVGPathSegLinetoHorizontalAbs

The SVGPathSegLinetoHorizontalAbs interface corresponds to an "absolute horizontal lineto" (H) path data command.

```
interface SVGPathSegLinetoHorizontalAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.14 Interface SVGPathSegLinetoHorizontalRel

The SVGPathSegLinetoHorizontalRel interface corresponds to a "relative horizontal lineto" (h) path data command.

```
interface SVGPathSegLinetoHorizontalRel : SVGPathSeg {
  attribute float x setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.15 Interface SVGPathSegLinetoVerticalAbs

The SVGPathSegLinetoVerticalAbs interface corresponds to an "absolute vertical lineto" (V) path data command.

```
interface SVGPathSegLinetoVerticalAbs : SVGPathSeg {
```

```
  attribute float y setraises(DOMException);
};
```

Attributes:

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.16 Interface SVGPathSegLinetoVerticalRel

The SVGPathSegLinetoVerticalRel interface corresponds to a "relative vertical lineto" (v) path data command.

```
interface SVGPathSegLinetoVerticalRel : SVGPathSeg {
  attribute float y setraises(DOMException);
};
```

Attributes:

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.17 Interface SVGPathSegCurvetoCubicSmoothAbs

The SVGPathSegCurvetoCubicSmoothAbs interface corresponds to an "absolute smooth cubic curveto" (S) path data command.

```
interface SVGPathSegCurvetoCubicSmoothAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x2** (float)

  The absolute X coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y2** (float)

  The absolute Y coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.18 Interface SVGPathSegCurvetoCubicSmoothRel

The SVGPathSegCurvetoCubicSmoothRel interface corresponds to a "relative smooth cubic curveto" (s) path data command.

```
interface SVGPathSegCurvetoCubicSmoothRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x2** (float)

  The relative X coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y2** (float)

  The relative Y coordinate for the second control point.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.19 Interface SVGPathSegCurvetoQuadraticSmoothAbs

The SVGPathSegCurvetoQuadraticSmoothAbs interface corresponds to an "absolute smooth cubic curveto" (T) path data command.

```
interface SVGPathSegCurvetoQuadraticSmoothAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The absolute X coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The absolute Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.20 Interface SVGPathSegCurvetoQuadraticSmoothRel

The SVGPathSegCurvetoQuadraticSmoothRel interface corresponds to a "relative smooth cubic curveto" (t) path data command.

```
interface SVGPathSegCurvetoQuadraticSmoothRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};
```

Attributes:

- **x** (float)

  The relative X coordinate for the end point of this path segment.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  The relative Y coordinate for the end point of this path segment.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

8.5.21 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.

```
interface SVGPathSegList {

  readonly attribute unsigned long numberOfItems;

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

Attributes:

- **numberOfItems** (readonly unsigned long)

  The number of items in the list.

Operations:

- void **clear**()

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

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.

- SVGPathSeg **initialize**(in SVGPathSeg *newItem*)

  Clears all existing current items from the list and re-initializes the list to hold the single item specified by the parameter. If the inserted item is already in a list, it is removed from its previous list before it is inserted into this list. The inserted item is the item itself and not a copy.

  Parameters

  - SVGPathSeg *newItem*
    The item which should become the only member of the list.

  Returns
    The item being inserted into the list.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- SVGPathSeg **getItem**(in unsigned long *index*)

  Returns the specified item from the list. The returned item is the item itself and not a copy. Any changes made to the item are immediately reflected in the list.

  Parameters

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

  Returns
    The selected item.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGPathSeg **insertItemBefore**(in SVGPathSeg *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to insert before is *before* the removal of the item.

  Parameters

  - SVGPathSeg *newItem*
    The item which is to be inserted into the list.
  - 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.

  Returns
  The inserted item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.

- SVGPathSeg **replaceItem**(in SVGPathSeg *newItem*, in unsigned long *index*)

  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. The inserted item is the item itself and not a copy. If the item is already in this list, note that the index of the item to replace is *before* the removal of the item.

  Parameters

  - SVGPathSeg *newItem*
    The item which is to be inserted into the list.
  - unsigned long *index*
    The index of the item which is to be replaced. The first item is number 0.

  Returns
  The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.
- DOMException, code INDEX_SIZE_ERR
  Raised if the index number is greater than or equal to numberOfItems.

- SVGPathSeg **removeItem**(in unsigned long *index*)

  Removes an existing item from the list.

  Parameters

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

  Returns
  The removed item.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised when the list cannot be modified.
  - DOMException, code INDEX_SIZE_ERR
    Raised if the index number is greater than or equal to numberOfItems.

- SVGPathSeg **appendItem**(in SVGPathSeg *newItem*)

  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. The inserted item is the item itself and not a copy.

  Parameters

  - SVGPathSeg *newItem*
    The item which is to be inserted. The first item is number 0.

  Returns
  The inserted item.

Exceptions

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised when the list cannot be modified.

8.5.22 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.

```
interface SVGAnimatedPathData {
  readonly attribute SVGPathSegList pathSegList;
  readonly attribute SVGPathSegList normalizedPathSegList;
  readonly attribute SVGPathSegList animatedPathSegList;
  readonly attribute SVGPathSegList animatedNormalizedPathSegList;
};
```

Attributes:

- **pathSegList** (readonly SVGPathSegList)

  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.

- **normalizedPathSegList** (readonly SVGPathSegList)

  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_LINETO_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).

- **animatedPathSegList** (readonly SVGPathSegList)

  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 read only. If the given attribute or property is not currently being animated, contains the same value as pathSegList.

- **animatedNormalizedPathSegList** (readonly SVGPathSegList)

  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 read only. If the given attribute or property is not currently being animated, contains the same value as normalizedPathSegList.

8.5.23 Interface SVGPathElement

The SVGPathElement interface corresponds to the 'path' element.

```
interface SVGPathElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable,
                           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:

- **pathLength** (readonly SVGAnimatedNumber)

  Corresponds to attribute pathLength on the given ‘path’ element.

Operations:

- float **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.

  Returns

  The total length of the path.

- SVGPoint **getPointAtLength**(in float *distance*)

  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

- float *distance*
  The distance along the path, relative to the start of the path, as a distance in the current user coordinate system.

Returns
The returned point in user space.

- unsigned long **getPathSegAtLength**(in float *distance*)

  Returns the index into pathSegList which is *distance* units along the path, utilizing the user agent's distance-along-a-path algorithm.

  Parameters

  - float *distance*
    The distance along the path, relative to the start of the path, as a distance in the current user coordinate system.

  Returns
  The index of the path segment, where the first path segment is number 0.

- SVGPathSegClosePath **createSVGPathSegClosePath**()

  Returns a stand-alone, parentless SVGPathSegClosePath object.

  Returns
  A stand-alone, parentless SVGPathSegClosePath object.

- SVGPathSegMovetoAbs **createSVGPathSegMovetoAbs**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegMovetoAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.
  - float *y*
    The absolute Y coordinate for the end point of this path segment.

Returns
A stand-alone, parentless SVGPathSegMovetoAbs object.

- SVGPathSegMovetoRel **createSVGPathSegMovetoRel**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegMovetoRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*
    The relative Y coordinate for the end point of this path segment.

  Returns
  A stand-alone, parentless SVGPathSegMovetoRel object.

- SVGPathSegLinetoAbs **createSVGPathSegLinetoAbs**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegLinetoAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.
  - float *y*
    The absolute Y coordinate for the end point of this path segment.

  Returns
  A stand-alone, parentless SVGPathSegLinetoAbs object.

- SVGPathSegLinetoRel **createSVGPathSegLinetoRel**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegLinetoRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*

The relative Y coordinate for the end point of this path segment.

Returns
A stand-alone, parentless SVGPathSegLinetoRel object.

- SVGPathSegCurvetoCubicAbs **createSVGPathSegCurvetoCubicAbs**(in float *x*, in float *y*, in float *x1*, in float *y1*, in float *x2*, in float *y2*)

  Returns a stand-alone, parentless SVGPathSegCurvetoCubicAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.
  - float *y*
    The absolute Y coordinate for the end point of this path segment.
  - float *x1*
    The absolute X coordinate for the first control point.
  - float *y1*
    The absolute Y coordinate for the first control point.
  - float *x2*
    The absolute X coordinate for the second control point.
  - float *y2*
    The absolute Y coordinate for the second control point.

  Returns
  A stand-alone, parentless SVGPathSegCurvetoCubicAbs object.

- SVGPathSegCurvetoCubicRel **createSVGPathSegCurvetoCubicRel**(in float *x*, in float *y*, in float *x1*, in float *y1*, in float *x2*, in float *y2*)

  Returns a stand-alone, parentless SVGPathSegCurvetoCubicRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*

The relative Y coordinate for the end point of this path segment.

- float *x1*
  The relative X coordinate for the first control point.
- float *y1*
  The relative Y coordinate for the first control point.
- float *x2*
  The relative X coordinate for the second control point.
- float *y2*
  The relative Y coordinate for the second control point.

Returns

A stand-alone, parentless SVGPathSegCurvetoCubicRel object.

- SVGPathSegCurvetoQuadraticAbs **createSVGPathSegCurvetoQuadraticAbs**(in float *x*, in float *y*, in float *x1*, in float *y1*)

  Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.
  - float *y*
    The absolute Y coordinate for the end point of this path segment.
  - float *x1*
    The absolute X coordinate for the first control point.
  - float *y1*
    The absolute Y coordinate for the first control point.

  Returns

  A stand-alone, parentless SVGPathSegCurvetoQuadraticAbs object.

- SVGPathSegCurvetoQuadraticRel **createSVGPathSegCurvetoQuadraticRel**(in float *x*, in float *y*, in float *x1*, in float *y1*)

  Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticRel object.

Parameters

- float *x*
  The relative X coordinate for the end point of this path segment.
- float *y*
  The relative Y coordinate for the end point of this path segment.
- float *x1*
  The relative X coordinate for the first control point.
- float *y1*
  The relative Y coordinate for the first control point.

Returns

A stand-alone, parentless SVGPathSegCurvetoQuadraticRel object.

- SVGPathSegArcAbs **createSVGPathSegArcAbs**(in float *x*, in float *y*, in float *r1*, in float *r2*, in float *angle*, in boolean *largeArcFlag*, in boolean *sweepFlag*)

  Returns a stand-alone, parentless SVGPathSegArcAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.
  - float *y*
    The absolute Y coordinate for the end point of this path segment.
  - float *r1*
    The x-axis radius for the ellipse (i.e., r1).
  - float *r2*
    The y-axis radius for the ellipse (i.e., r2).
  - float *angle*
    The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.
  - boolean *largeArcFlag*
    The value of the large-arc-flag parameter.
  - boolean *sweepFlag*
    The value of the large-arc-flag parameter.

Returns
A stand-alone, parentless SVGPathSegArcAbs object.

- SVGPathSegArcRel **createSVGPathSegArcRel**(in float *x*, in float *y*, in float *r1*, in float *r2*, in float *angle*, in boolean *largeArcFlag*, in boolean *sweepFlag*)

  Returns a stand-alone, parentless SVGPathSegArcRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*
    The relative Y coordinate for the end point of this path segment.
  - float *r1*
    The x-axis radius for the ellipse (i.e., r1).
  - float *r2*
    The y-axis radius for the ellipse (i.e., r2).
  - float *angle*
    The rotation angle in degrees for the ellipse's x-axis relative to the x-axis of the user coordinate system.
  - boolean *largeArcFlag*
    The value of the large-arc-flag parameter.
  - boolean *sweepFlag*
    The value of the large-arc-flag parameter.

  Returns
  A stand-alone, parentless SVGPathSegArcRel object.

- SVGPathSegLinetoHorizontalAbs **createSVGPathSegLinetoHorizontalAbs**(in float *x*)

  Returns a stand-alone, parentless SVGPathSegLinetoHorizontalAbs object.

  Parameters

  - float *x*
    The absolute X coordinate for the end point of this path segment.

Returns
A stand-alone, parentless SVGPathSegLinetoHorizontalAbs object.

- SVGPathSegLinetoHorizontalRel **createSVGPathSegLinetoHorizontalRel**(in float *x*)

  Returns a stand-alone, parentless SVGPathSegLinetoHorizontalRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.

  Returns
  A stand-alone, parentless SVGPathSegLinetoHorizontalRel object.

- SVGPathSegLinetoVerticalAbs **createSVGPathSegLinetoVerticalAbs**(in float *y*)

  Returns a stand-alone, parentless SVGPathSegLinetoVerticalAbs object.

  Parameters

  - float *y*
    The absolute Y coordinate for the end point of this path segment.

  Returns
  A stand-alone, parentless SVGPathSegLinetoVerticalAbs object.

- SVGPathSegLinetoVerticalRel **createSVGPathSegLinetoVerticalRel**(in float *y*)

  Returns a stand-alone, parentless SVGPathSegLinetoVerticalRel object.

  Parameters

  - float *y*
    The relative Y coordinate for the end point of this path segment.

  Returns
  A stand-alone, parentless SVGPathSegLinetoVerticalRel object.

- SVGPathSegCurvetoCubicSmoothAbs **createSVGPathSegCurvetoCubicSmoothAbs**(in float *x*, in float *y*, in float *x2*, in float *y2*)

  Returns a stand-alone, parentless SVGPathSegCurvetoCubicSmoothAbs object.

Parameters

- float *x*
  The absolute X coordinate for the end point of this path segment.
- float *y*
  The absolute Y coordinate for the end point of this path segment.
- float *x2*
  The absolute X coordinate for the second control point.
- float *y2*
  The absolute Y coordinate for the second control point.

Returns

A stand-alone, parentless SVGPathSegCurvetoCubicSmoothAbs object.

- SVGPathSegCurvetoCubicSmoothRel **createSVGPathSegCurvetoCubicSmoothRel**(in float *x*, in float *y*, in float *x2*, in float *y2*)

  Returns a stand-alone, parentless SVGPathSegCurvetoCubicSmoothRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*
    The relative Y coordinate for the end point of this path segment.
  - float *x2*
    The relative X coordinate for the second control point.
  - float *y2*
    The relative Y coordinate for the second control point.

  Returns

  A stand-alone, parentless SVGPathSegCurvetoCubicSmoothRel object.

- SVGPathSegCurvetoQuadraticSmoothAbs **createSVGPathSegCurvetoQuadraticSmoothAbs**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothAbs object.

Parameters

- float *x*
  The absolute X coordinate for the end point of this path segment.
- float *y*
  The absolute Y coordinate for the end point of this path segment.

Returns

A stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothAbs object.

- SVGPathSegCurvetoQuadraticSmoothRel **createSVGPathSegCurvetoQuadraticSmoothRel**(in float *x*, in float *y*)

  Returns a stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothRel object.

  Parameters

  - float *x*
    The relative X coordinate for the end point of this path segment.
  - float *y*
    The relative Y coordinate for the end point of this path segment.

  Returns

  A stand-alone, parentless SVGPathSegCurvetoQuadraticSmoothRel object.

9 Basic Shapes

Contents

9.1 Introduction

SVG contains the following set of basic shape elements:

- rectangles (including optional rounded corners), created with the 'rect' element,
- circles, created with the 'circle' element,
- ellipses, created with the 'ellipse' element,
- straight lines, created with the 'line' element,
- polylines, created with the 'polyline' element, and
- polygons, created with the 'polygon' element.

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'.

Categories: 'rect'
Basic shape element, graphics element, shape element

Content model:
Any number of the following elements, in any order:
animation elements
descriptive elements

Attributes:
conditional processing attributes
core attributes
graphical event attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'transform'
'x'
'y'
'width'
'height'
'rx'
'ry'

DOM Interfaces:
SVGRectElement

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.

The values used for the x- and y-axis rounded corner radii are determined implicitly if the 'rx' or 'ry' attributes (or both) are not specified, or are specified but with invalid values. The values are also subject to clamping so that the lengths of the straight segments of the rectangle are never negative. The effective values for 'rx' and 'ry' are determined by following these steps in order:

1. Let *rx* and *ry* be length values.
2. If neither 'rx' nor 'ry' are properly specified, then set both *rx* and *ry* to 0. (This will result in square corners.)
3. Otherwise, if a properly specified value is provided for 'rx', but not for 'ry', then set both *rx* and *ry* to the value of 'rx'.
4. Otherwise, if a properly specified value is provided for 'ry', but not for 'rx', then set both *rx* and *ry* to the value of 'ry'.
5. Otherwise, both 'rx' and 'ry' were specified properly. Set *rx* to the value of 'rx' and *ry* to the value of 'ry'.
6. If *rx* is greater than half of 'width', then set *rx* to half of 'width'.
7. If *ry* is greater than half of 'height', then set *ry* to half of 'height'.
8. The effective values of 'rx' and 'ry' are *rx* and *ry*, respectively.

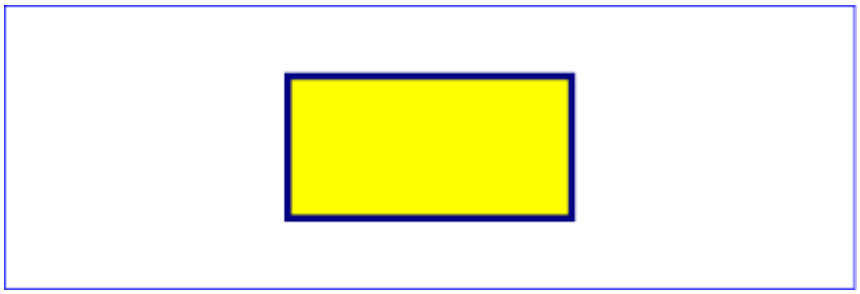
Mathematically, a 'rect' element can be mapped to an equivalent 'path' element as follows: (Note: all coordinate and length values are first converted into user space coordinates according to Units.)

- 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

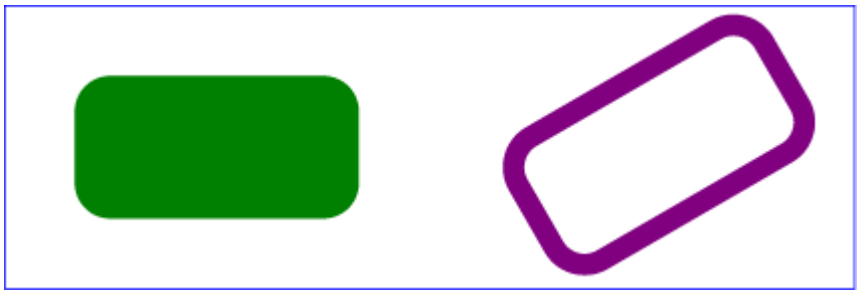
Example rect01



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



9.3 The 'circle' element

The 'circle' element defines a circle based on a center point and a radius.

Categories: 'circle'

Basic shape element, graphics element, shape element

Content model:

Any number of the following elements, in any order:

animation elements

descriptive elements

Attributes:

conditional processing attributes

core attributes

graphical event attributes

presentation attributes

‘class’
‘style’
‘externalResourcesRequired’
‘transform’
‘cx’
‘cy’
‘r’

DOM Interfaces:
SVGCircleElement

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.

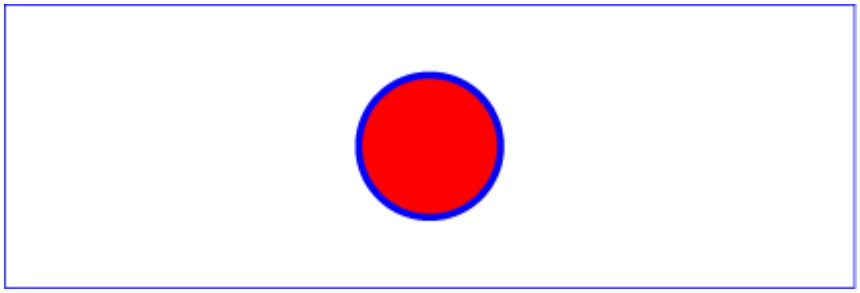
`r` = "**<length>**"
The radius of the circle.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
Animatable: yes.

The arc of a ‘circle’ element begins at the "3 o'clock" point on the radius and progresses towards the "9 o'clock" point. The starting point and direction of the arc are affected by the user space transform in the same manner as the geometry of the element.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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.

Categories: 'ellipse'

Basic shape element, graphics element, shape element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements

Attributes:

- conditional processing attributes
- core attributes
- graphical event attributes
- presentation attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'transform'
- 'cx'
- 'cy'
- 'rx'
- 'ry'

DOM Interfaces:

SVGEllipseElement

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.

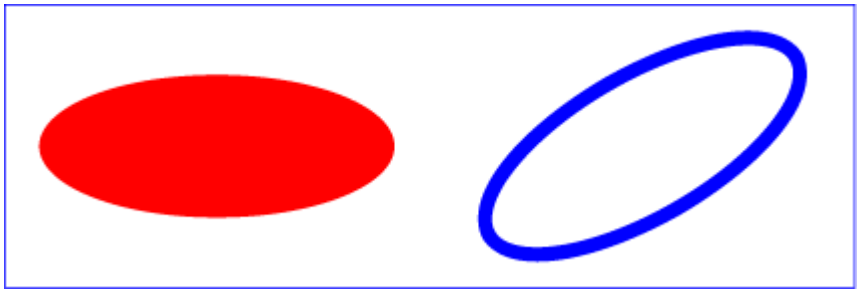
ry = "**<length>**"
: The y-axis radius of the ellipse.
A negative value is an error (see Error processing). A value of zero disables rendering of the element.
Animatable: yes.

The arc of an **'ellipse'** element begins at the "3 o'clock" point on the radius and progresses towards the "9 o'clock" point. The starting point and direction of the arc are affected by the user space transform in the same manner as the geometry of the element.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



9.5 The 'line' element

The 'line' element defines a line segment that starts at one point and ends at another.

Categories: 'line'

Basic shape element, graphics element, shape element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements

Attributes:

- conditional processing attributes
- core attributes
- graphical event attributes
- presentation attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'transform'
- 'x1'
- 'y1'
- 'x2'
- 'y2'

DOM Interfaces:

SVGLineElement

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.

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

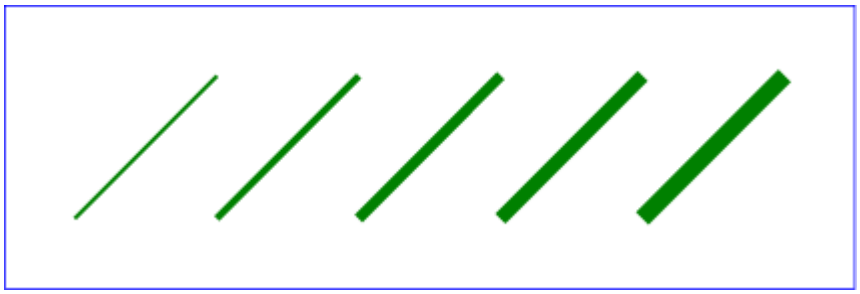
Because 'line' elements are single lines and thus are geometrically one-dimensional, they have no interior; thus, 'line' elements are never filled (see the '**fill**' property).

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



9.6 The 'polyline' element

The 'polyline' element defines a set of connected straight line segments. Typically, 'polyline' elements define open shapes.

Categories: 'polyline'

Basic shape element, graphics element, shape element

Content model:

Any number of the following elements, in any order:

animation elements

descriptive elements

Attributes:

conditional processing attributes

core attributes

graphical event attributes

presentation attributes

'class'

'style'

'externalResourcesRequired'

'transform'

'points'

DOM Interfaces:

SVGPolylineElement

Attribute definitions:

`points = "<list-of-points>"`

The points that make up the polyline. All coordinate values are in the user coordinate system.
Animatable: yes.

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.

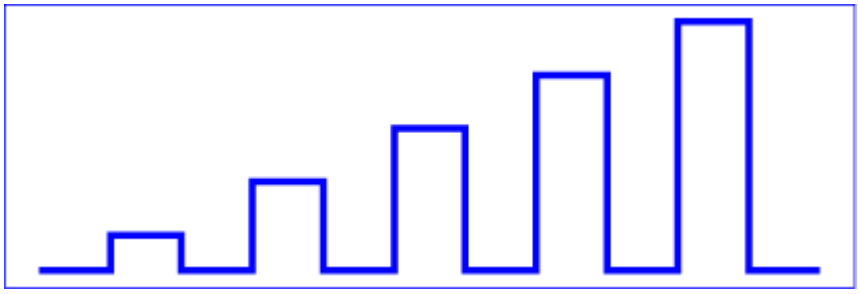
Mathematically, a 'polyline' element can be mapped to an equivalent 'path' element as follows:

- 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



9.7 The 'polygon' element

The 'polygon' element defines a closed shape consisting of a set of connected straight line segments.

Categories: ‘polygon’

Basic shape element, graphics element, shape element

Content model:

Any number of the following elements, in any order:

animation elements

descriptive elements

Attributes:

conditional processing attributes

core attributes

graphical event attributes

presentation attributes

‘class’

‘style’

‘externalResourcesRequired’

‘transform’

‘points’

DOM Interfaces:

SVGPolygonElement

Attribute definitions:

`points = "<list-of-points>"`

The points that make up the polygon. All coordinate values are in the user coordinate system.

Animatable: yes.

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:

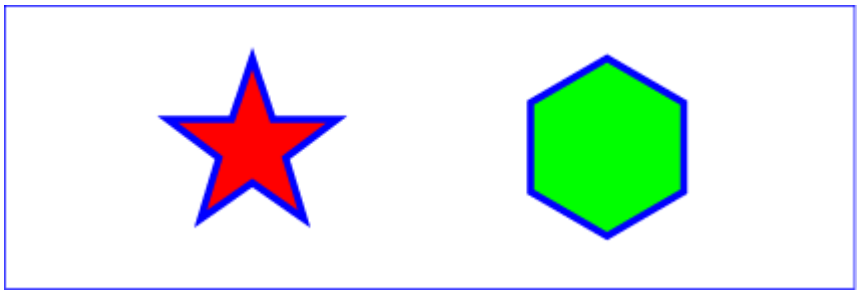
- perform an absolute moveto operation to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute lineto operation to that coordinate pair
- perform a closepath command

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



9.7.1 The grammar for points specifications in 'polyline' and 'polygon' elements

The following is the Extended Backus-Naur Form (EBNF) 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 negative-coordinate
coordinate:
    number
number:
    sign? integer-constant
    | sign? floating-point-constant
negative-coordinate:
    "-" integer-constant
    | "-" 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)+
```

9.8 DOM interfaces

9.8.1 Interface SVGRectElement

The SVGRectElement interface corresponds to the 'rect' element.

```
interface SVGRectElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable {
  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:

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'rect' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'rect' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'rect' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'rect' element.

- **rx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'rx' on the given 'rect' element.

- **ry** (readonly SVGAnimatedLength)

  Corresponds to attribute 'ry' on the given 'rect' element.

9.8.2 Interface SVGCircleElement

The SVGCircleElement interface corresponds to the 'circle' element.

```
interface SVGCircleElement : SVGElement,
                             SVGTests,
                             SVGLangSpace,
                             SVGExternalResourcesRequired,
                             SVGStylable,
                             SVGTransformable {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
};
```

Attributes:

- **cx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cx' on the given 'circle' element.

- **cy** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cy' on the given 'circle' element.

- **r** (readonly SVGAnimatedLength)

  Corresponds to attribute 'r' on the given 'circle' element.

9.8.3 Interface SVGEllipseElement

The SVGEllipseElement interface corresponds to the 'ellipse' element.

```
interface SVGEllipseElement : SVGElement,
                              SVGTests,
                              SVGLangSpace,
                              SVGExternalResourcesRequired,
                              SVGStylable,
                              SVGTransformable {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};
```

Attributes:

- **cx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cx' on the given 'ellipse' element.

- **cy** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cy' on the given 'ellipse' element.

- **rx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'rx' on the given 'ellipse' element.

- **ry** (readonly SVGAnimatedLength)

  Corresponds to attribute 'ry' on the given 'ellipse' element.

9.8.4 Interface SVGLineElement

The SVGLineElement interface corresponds to the 'line' element.

```
interface SVGLineElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable {
  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};
```

Attributes:

- **x1** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x1' on the given 'line' element.

- **y1** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y1' on the given 'line' element.

- **x2** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x2' on the given 'line' element.

- **y2** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y2' on the given 'line' element.

9.8.5 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.

```
interface SVGAnimatedPoints {
  readonly attribute SVGPointList points;
  readonly attribute SVGPointList animatedPoints;
};
```

Attributes:

- **points** (readonly SVGPointList)

  Provides access to the base (i.e., static) contents of the 'points' attribute.

- **animatedPoints** (readonly SVGPointList)

  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.

9.8.6 Interface SVGPolylineElement

The SVGPolylineElement interface corresponds to the 'polyline' element.

```
interface SVGPolylineElement : SVGElement,
                               SVGTests,
                               SVGLangSpace,
                               SVGExternalResourcesRequired,
                               SVGStylable,
                               SVGTransformable,
                               SVGAnimatedPoints {
};
```

9.8.7 Interface SVGPolygonElement

The SVGPolygonElement interface corresponds to the 'polygon' element.

```
interface SVGPolygonElement : SVGElement,
                              SVGTests,
                              SVGLangSpace,
                              SVGExternalResourcesRequired,
                              SVGStylable,
                              SVGTransformable,
                              SVGAnimatedPoints {
};
```

10 Text

Contents

10.17.10 Interface SVGGlyphRefElement

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], section 2.4) 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:

- 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, consist 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 ([CSS2], section 15.1) 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 Coordinate units on the em square; in [CSS2], section 15.4.3.) 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 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.

Categories: 'text'

Graphics element, text content element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- text content child elements
- 'a'

Attributes:

- conditional processing attributes
- core attributes
- graphical event attributes
- presentation attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'transform'
- 'lengthAdjust'
- 'x'
- 'y'
- 'dx'
- 'dy'
- 'rotate'
- 'textLength'

DOM Interfaces:

SVGTextElement

Attribute definitions:

x = "**<list-of-coordinates>**"

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* charac-

ters 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 = "<list-of-coordinates>"`
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 = "<list-of-lengths>"`
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 = "<list-of-lengths>"`
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 = "<list-of-numbers>"`
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).

`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.

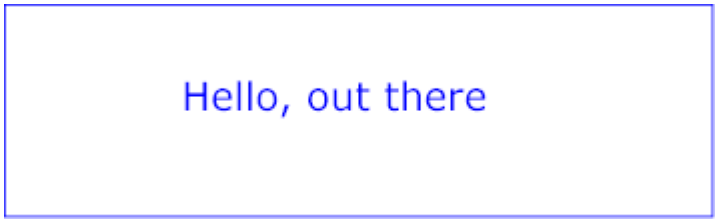
`lengthAdjust` = **"spacing|spacingAndGlyphs"**

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.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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 test01



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.

Categories: 'tspan'

Text content element, text content child element

Content model:

Any number of the following elements, in any order:

- descriptive elements
- 'a'
- 'altGlyph'
- 'animate'
- 'animateColor'
- 'set'
- 'tref'
- 'tspan'

Attributes:

- conditional processing attributes
- core attributes
- graphical event attributes
- presentation attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'x'
- 'y'
- 'dx'
- 'dy'
- 'rotate'
- 'textLength'

'lengthAdjust'

DOM Interfaces:
SVGTSpanElement

Attribute definitions:

x = "**<list-of-coordinates>**"
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 = "**<list-of-coordinates>**"
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 = "**<list-of-lengths>**"
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 = "<list-of-lengths>"`
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 = "<list-of-numbers>"`
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 the rotation value specified by the last number must be used.
If the attribute is not specified and 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). If there are more characters than <number>s specified in the ancestor's 'rotate' attribute, then for each of these extra characters the rotation value specified by the last number must be used.
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).

`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.

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 orresponding 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</tspan>
```

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.

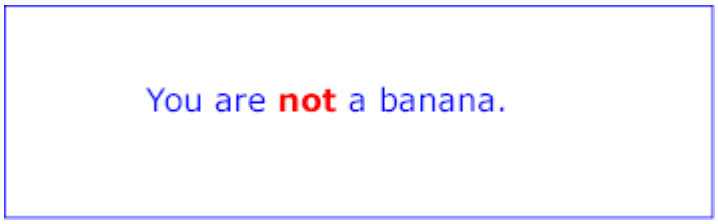
The following examples show basic use of the '**tspan**' element.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

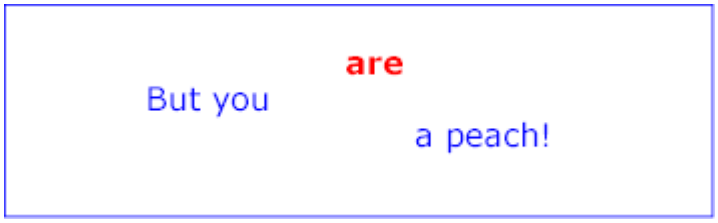
Example tspan01



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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.

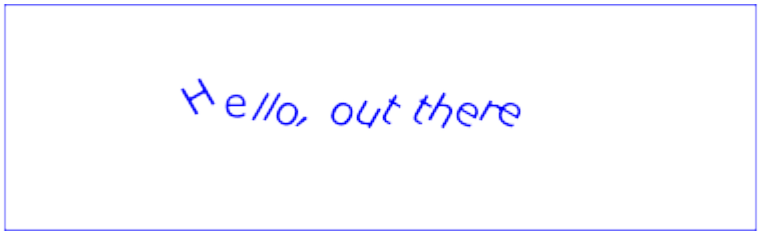
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example tspan03 - using tspan's x and y attributes
        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

Example tspan04 uses the 'rotate' attribute on the 'tspan' element to rotate the glyphs to be rendered. This example shows a single text string in a 'tspan' element that contains more characters than the number of values specified in the 'rotate' attribute. In this case the last value specified in the 'rotate' attribute of the 'tspan' must be applied to the remaining characters in the string.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300"
  xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>
    Example tspan04 - The number of rotate values is less than the number of
    characters in the string.
  </desc>
  <text font-family="Verdana" font-size="55" fill="blue" >
    <tspan x="250" y="150" rotate="-30,0,30">
      Hello, out there
    </tspan>
  </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 tspan04



Example tspan05 specifies the 'rotate' attribute on the 'text' element and on all but one of the child 'tspan' elements to rotate the glyphs to be rendered. The example demonstrates the propagation of the 'rotate' attribute.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" viewBox="0 0 500 120"
  xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>
    Example tspan05 - propagation of rotation values to nested tspan elements.
  </desc>
  <text id="parent" font-family="Arial, sans-serif" font-size="32" fill="red" x="40" y="40"
    rotate="5,15,25,35,45,55">
    Not

    <tspan id="child1" rotate="-10,-20,-30,-40" fill="orange">
      all characters

      <tspan id="child2" rotate="70,60,50,40,30,20,10" fill="yellow">
        in

        <tspan id="child3">
          the
        </tspan>
      </tspan>

      <tspan id="child4" fill="orange" x="40" y="90">
        text
      </tspan>

      have a
    </tspan>

    <tspan id="child5" rotate="-10" fill="blue">
      specified
    </tspan>

    rotation
  </text>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="498" height="118" fill="none"
        stroke="blue" stroke-width="2" />
</svg>
```

Example tspan05

Rotation of red text inside the 'text' element:

- The 'rotate' value will rotate the characters in the text *"Not "* by 5, 15, 25 and 35 degrees respectively.
- A 'rotate' value is applied to the space that follows the text *"Not"*, to the space in between the text in the "child1" and "child5" 'tspan' elements, and to the space before the text *"rotation"*.
- The next current 'rotate' value specified is 45 followed by 55. The current 'rotate' value in the 'text' element is incremented as subsequent characters in the text of the child 'tspan' elements are processed.
- The next immediate 'tspan' element specifies rotate values for the text, hence the current 'rotate' value will change to the next value in the list (but is not used) as each character is processed until the last value of 55 degrees is reached.
- The last 'rotate' value of 55 degrees will be applied to all the characters in the text *"rotation"*.

Rotation of the orange text inside the "child1" 'tspan' element:

- The 'rotate' value will rotate the first 4 characters in the text *"all characters "* by -10, -20, -30 and -40 respectively.
- The last 'rotate' value of -40 becomes the current 'rotate' value and will be applied to all subsequent characters in the 'tspan' element and to any child 'tspan' elements that do not specify 'rotate' values.
- The "child4" 'tspan' element does not specify any 'rotate' values and thus uses the current 'rotate' of its ancestor ("child1" 'tspan' element). All the characters in the text *"text"* specified within the "child4" 'tspan' element will be rotated by -40 degrees.
- The last 'rotate' value of -40 degrees will be applied to all the characters in the text *"have a"*.
- A 'rotate' value is applied to the space in between the text in the "child2" and "child4" 'tspan' elements, and to the space before the text *"have a"*.

Rotation of the yellow text inside the "child2" 'tspan' element:

- The 'rotate' value will rotate the characters in the (yellow) text *"in "* by 70, 60, and 50 degrees respectively.
- A 'rotate' value is applied to the space that follows the text *"in"*.
- There are more 'rotate' values specified than characters, thus the additional 'rotate' values will be applied to the "child3" 'tspan' element which does not specified any 'rotate' values.

- The characters in the text *"the"* specified within the "child3" 'tspan' element will be rotated 40, 30 and 20 degrees respectively.

Rotation of the blue text inside the "child5" 'tspan' element:

- The 'rotate' value will rotate all the characters in text *"specified"* by -10 degrees.
- Only one 'rotate' value is specified and is thus applied to all characters in the 'tspan' element.

The following diagram illustrates how the rotation values propagate to 'tspan' elements nested withing a 'text' element

```
<text id="parent" font-family="Arial, sans-serif" font-size="18"
    x="30" y="200" rotate="5,15,25,35,45,55">
```

N o t

5 15 25 35

```
    <tspan id="child1" rotate="-10,-20,-30,-40">
```

a l l c h a r a c t e r s

-10 -20 -30 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40

```
        <tspan id="child2" rotate="70,60,50,40,30,20,10">
```

i n

70 60 50

```
            <tspan id="child3">
```

t h e

40 30 20

```
            </tspan>
        </tspan>
```

-40

```
        <tspan id="child4">
```

t e x t

-40 -40 -40 -40

```
        </tspan>
```

-40

h a v e a

-40 -40 -40 -40 -40 -40

```
    </tspan>
```

55

```
    <tspan id="child5" rotate="-10">
```

s p e c i f i e d

-10 -10 -10 -10 -10 -10 -10 -10 -10

```
    </tspan>
```

55

r o t a t i o n

55 55 55 55 55 55 55 55

```
</text>
```

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.

Categories: 'tref'
Text content element, text content child element

Content model:
Any number of the following elements, in any order:
descriptive elements
'animate'
'animateColor'
'set'

Attributes:
conditional processing attributes
core attributes
graphical event attributes
presentation attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'xlink:href'

DOM Interfaces:
SVGTRefElement

Attribute definitions:

`xlink:href = "<iri>"`
An IRI reference to an element whose character data content shall be used as character data for this 'tref' element.
Animatable: yes.

All character data within the referenced element, including character data enclosed within additional markup, will be rendered.

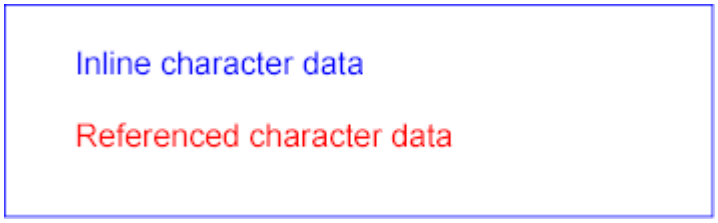
The 'x', 'y', 'dx', 'dy' and 'rotate' attributes have the same meanings as for the 'tspan' element. The attributes are applied as if the 'tref' element was replaced by a 'tspan' with the referenced character data (stripped of all supplemental markup) embedded within the hypothetical 'tspan' element.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300" version="1.1"
     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



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 Top Baseline; in [CSS2], section 15.4.18). 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 defined in CSS2, ([CSS2], section 16.4), 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.

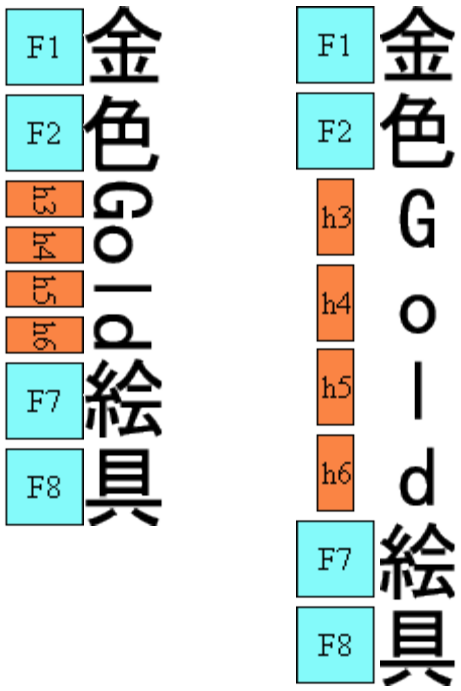
The text layout diagrams in this section use the following symbols:

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.

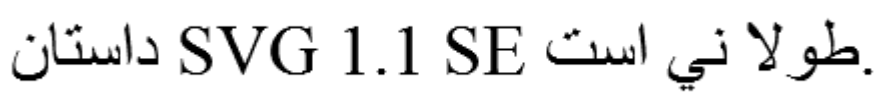
The Unicode standard ([UNICODE], specifically [UAX9]) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. The SVG user agent applies this bidirectional algorithm when determining the layout of characters within a text content block 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 whose glyphs are perpendicular to the inline-progression-direction.

In many cases, the bidirectional algorithm from Unicode [UNICODE] produces the desired result automatically, and in such cases the author does not need to use these properties. For other cases, such as when using right-to-left languages, it may be sufficient to add the '**direction**' property to the rootmost 'svg' element, and allow that direction to inherit to all text elements, as in the following example (which may be used as a template):

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400"
     direction="rtl" xml:lang="fa">

  <title direction="ltr" xml:lang="en">Right-to-left Text</title>
  <desc direction="ltr" xml:lang="en">
    A simple example for using the 'direction' property in documents
    that predominantly use right-to-left languages.
  </desc>

  <text x="200" y="200" font-size="20">داستان SVG 1.1 SE است نی طولا.</text>

</svg>
```

Example

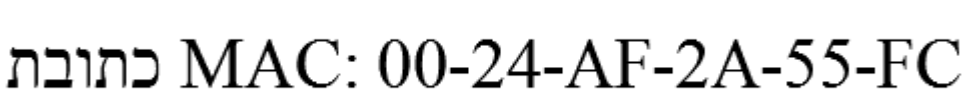


Below is another example, where where implicit bidi reordering is not sufficient:

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
     width="100%" height="100%" viewBox="0 0 400 400"
     direction="rtl" xml:lang="he">

  <title direction="ltr" xml:lang="en">Right-to-left Text</title>
  <desc direction="ltr" xml:lang="en">
    An example for using the 'direction' and 'unicode-bidi' properties
    in documents that predominantly use right-to-left languages.
  </desc>

  <text x="200" y="200" font-size="20"> כתובת
        MAC:&#x200F;
            <tspan direction="ltr" unicode-bidi="embed">00-24-AF-2A-55-FC</tspan>
        </text>
```

```
</svg>
```

Example



Within text content elements, the alignment of text with regards to the '**text-anchor**' property is determined by the value of the '**direction**' property. For example, given a '**text**' element with a '**text-anchor**' value of **"end"**, for a '**direction**' value of **"ltr"**, the text will extend to the left of the position of the '**text**' element's '**x**' attribute value, while for '**direction**' value of **"rtl"**, the text will extend to the right of the position of the '**text**' element's '**x**' attribute value.

A more complete discussion of bidirectionality can be found in the Text direction section of CSS 2 ([CSS2], section 9.10).

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 bi-directional 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 on an element that does not by itself establish a new text chunk (such as a '**tspan**' element without absolute position adjustments due to '**x**' or '**y**' attributes), 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 '**direction**' property is in CSS2 ([CSS2], section 9.10).

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 '**unicode-bidi**' property is in CSS2 ([CSS2], section 9.10).

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 resulting rendered text is at the initial current text position. For an element with a 'direction' property value of **"ltr"** (typical for most European languages), the left side of the text is rendered at the initial text position. For an element with a 'direction' property value of **"rtl"** (typical for Arabic and Hebrew), the right side of the text is rendered at the initial text position. For an element with a vertical primary text direction (often typical for Asian text), the top side of the text is rendered at the initial text position.

middle
: The rendered characters are aligned such that the geometric middle of the resulting rendered text is at the initial current text position.

end
: The rendered characters are aligned such that the end of the resulting rendered text is at the initial current text position. For an element with a 'direction' property value of **"ltr"** (typical for most European languages), the right side of the text is rendered at the initial text position. For an element with a 'direction' property value of **"rtl"** (typical for Arabic and Hebrew), the left side of the text is rendered at the initial text position. For an element with a vertical primary text direction (often typical for Asian text), the bottom of the text is rendered at the initial text position.

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], section 15.3.4). 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 ([XSL], section 7.9.1). 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 | ideographic | alphabetic | hanging | mathematical | central | middle | text-after-edge | text-before-edge | 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**'.

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.

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.

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 construc-

ted 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.

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 | central | 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.

middle
: The alignment-point of the object being aligned is aligned with the "middle" 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.

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.

'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 these properties is in CSS2 ([CSS2], chapter section 15.2). Any SVG-specific notes about these properties are contained in the descriptions below.

Note also the rules for expressing the syntax of CSS property values ([CSS2], section 1.3.2).

'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. Unless the family name corresponds to a CSS IDENT, it must be quoted. Except for any additional information provided in this specification, the normative definition of the property is in CSS2 ([CSS2], section 15.2.2).

'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 ([CSS2], section 15.2.3).

'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 ([CSS2], section 15.2.3).

'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 ([CSS2], section 15.2.3).

'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 ([CSS2], section 15.2.3).

'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 ([CSS2], section 15.2.4).

'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)

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 ([CSS2], section 15.2.4).

'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)

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 ([CSS2], section 15.2.5).

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 ([CSS2], section 16.4).

'**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 ([CSS2], section 16.4).

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 ([CSS2], section 16.3.1).

The CSS2 specification 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.

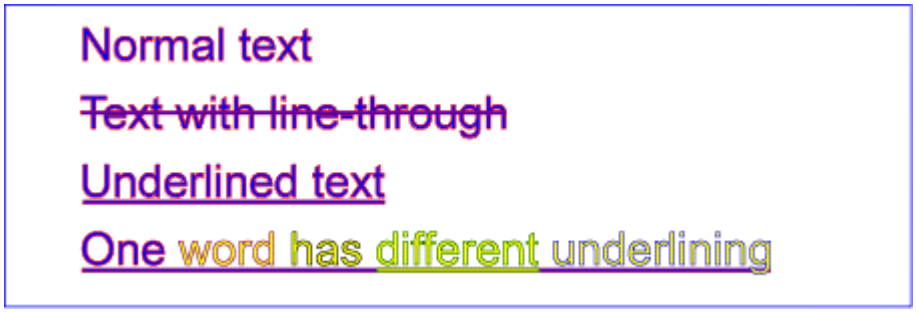
Because SVG allows text to be both filled and stroked, drawing order matters in some circumstances with text decorations. Text decoration drawing order should be as follows:

- 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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 an IRI reference to a 'path' element.

10.13.2 The 'textPath' element

Categories: 'textPath'

Text content element, text content child element

Content model:

Any number of the following elements, in any order:

- descriptive elements
- 'a'
- 'altGlyph'
- 'animate'
- 'animateColor'
- 'set'
- 'tref'
- 'tspan'

Attributes:

- conditional processing attributes
- core attributes
- graphical event attributes
- presentation attributes
- xlink attributes

'class'
'style'
'externalResourcesRequired'
'xlink:href'
'startOffset'
'method'
'spacing'

DOM Interfaces:
SVGTextPathElement

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'.

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 = "<iri>"`
An IRI reference to the 'path' element onto which the glyphs will be rendered. If <iri> 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.

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" version="1.1">
  <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" version="1.1">
  <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>
```

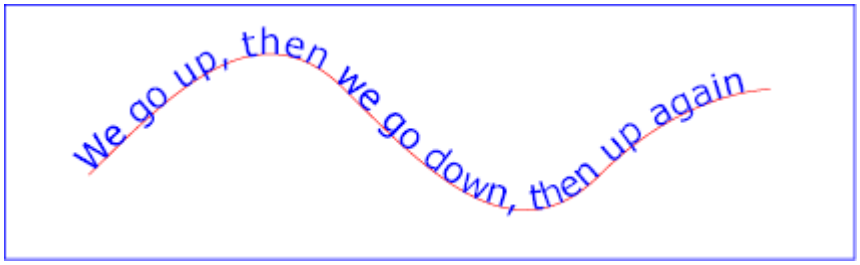
Note that the `transform="translate(25,25)"` has no effect on the 'textPath' element, whereas the `trans-`

`form="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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300" version="1.1"
     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

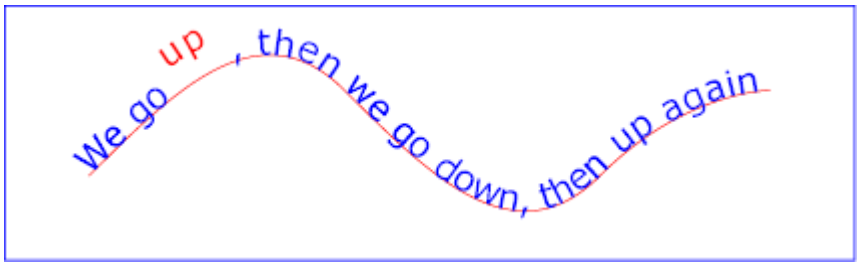


Example toap02 shows how 'tspan' elements can be included within 'textPath' elements to adjust styling attributes and adjust the current text position before rendering a particular glyph. The first occurrence of the word "up" is filled with the color red. Attribute 'dy' is used to lift the word "up" from the baseline.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300" version="1.1"
     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>
```

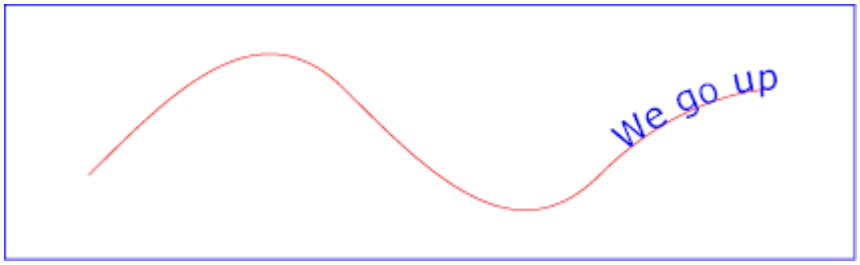
Example toap02



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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300" version="1.1"
     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



10.13.3 Text on a path layout rules

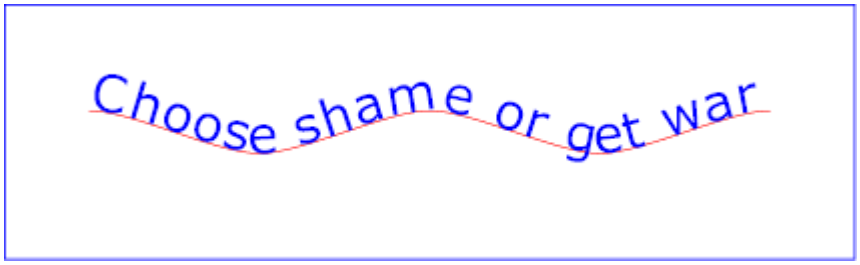
Conceptually, for text on a path the target path is stretched out into either a horizontal or vertical straight line segment. For horizontal text layout flows, the path is stretched out into a hypothetical horizontal line segment such that the start of the path is mapped to the left of the line segment. 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.

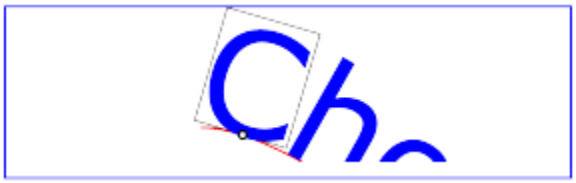
Example toap04 will be used to illustrate the particular layout rules for text on a path that supplement the basic text layout rules for straight line horizontal or vertical text.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3.6cm" viewBox="0 0 1000 300" version="1.1"
     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

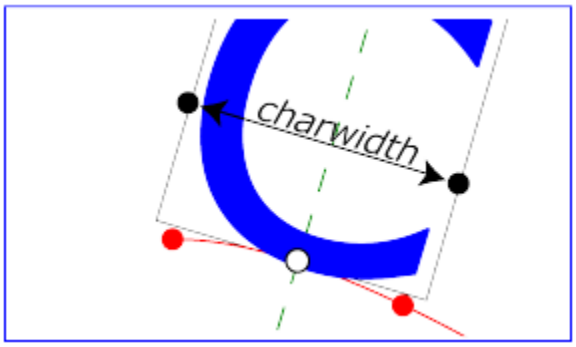


The following picture does an initial zoom in on the first glyph in the 'text' element.



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 vertical, 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.

10.14.1 The 'altGlyph' element

The 'altGlyph' element provides control over the glyphs used to render particular character data.

Categories: ‘altGlyph’
Text content element, text content child element

Content model:
Any elements or character data.

Attributes:
conditional processing attributes
core attributes
graphical event attributes
presentation attributes
xlink attributes
‘class’
‘style’
‘externalResourcesRequired’
‘x’
‘y’
‘dx’
‘dy’
‘glyphRef’
‘format’
‘rotate’
‘xlink:href’

DOM Interfaces:
SVGAltGlyphElement

Attribute definitions:

`xlink:href = "<iri>"`
An IRI 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 CSS2 ([CSS2], section 15.3.5), such as *TrueDoc™ Portable Font Resource* or *Embedded OpenType*, then the <string> must contain the corresponding font format string, such as *truedoc-pfr* or *embedded-opentype*. (This attribute has the same meaning as the 'format' attribute on the 'glyphRef' element.)
Animatable: no.

`x = "<list-of-coordinates>"`

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 = "<list-of-coordinates>"`

The corresponding absolute Y coordinates for rendering the 'altGlyph' element.
Animatable: yes.

`dx = "<list-of-lengths>"`

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 = "<list-of-lengths>"`

The corresponding relative Y coordinates for rendering the 'altGlyph' element.
Animatable: yes.

`rotate = "<list-of-numbers>"`

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).

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.

10.14.2 The 'altGlyphDef', 'altGlyphItem' and 'glyphRef' elements

The 'altGlyphDef' element defines a set of possible glyph substitutions.

Categories: 'altGlyphDef'
None

Content model:
Either:
- one or more 'glyphRef' elements, or
- one or more 'altGlyphItem' elements.

Attributes:
core attributes

DOM Interfaces:
SVGAltGlyphDefElement

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.

Categories: 'altGlyphItem'

None

Content model:

One or more 'glyphRef' elements.

Attributes:

core attributes

DOM Interfaces:

SVGAltGlyphItemElement

The 'glyphRef' element defines a possible glyph to use.

Categories: 'glyphRef'

None

Content model:

Empty.

Attributes:

core attributes
presentation attributes
xlink attributes
'class'
'style'
'x'
'y'
'dx'
'dy'
'glyphRef'

'format'
'xlink:href'

DOM Interfaces:
SVGGlyphRefElement

Attribute definitions:

`xlink:href = "<iri>"`
An IRI 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 ([CSS2], section 15.3.5), such as *TrueDoc™ Portable Font Resource* or *Embedded OpenType*, then the <string> must contain the corresponding font format string, such as *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 CSS2 ([CSS2], chapter 15).
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 CSS2 ([CSS2],

chapter 15).
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.

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. Note that any child element of a 'text' element may also have an 'xml:space' attribute which will apply to that child element's text content. 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 DOM.

'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' and '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.

Note that a glyph corresponding to a whitespace character should only be displayed as a visible but blank space, even if the glyph itself happens to be non-blank. See display of unsupported characters [UNICODE].

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

10.17.1 Interface SVGTextContentElement

The SVGTextContentElement is inherited by various text-related interfaces, such as SVGTextElement, SVGTSpanElement, SVGTRefElement, SVGAltGlyphElement and SVGTextPathElement.

For the methods on this interface that refer to an index to a character or a number of characters, these references are to be interpreted as an index to a UTF-16 code unit or a number of UTF-16 code units, respectively. This is for consistency with DOM Level 2 Core, where methods on the CharacterData interface use UTF-16 code units as indexes and counts within the character data. Thus for example, if the text content of a 'text' element is a single non-BMP character, such as U+10000, then invoking getNumberOfChars on that element will return 2 since there are two UTF-16 code units (the surrogate pair) used to represent that one character.

```
interface SVGTextContentElement : SVGElement,
                                  SVGTests,
                                  SVGLangSpace,
                                  SVGExternalResourcesRequired,
                                  SVGStylable {

  // 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);
};
```

Constants in group "lengthAdjust Types":

- **LENGTHADJUST_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'spacing'**.

- **LENGTHADJUST_SPACINGANDGLYPHS** (unsigned short)

  Corresponds to value **'spacingAndGlyphs'**.

Attributes:

- **textLength** (readonly SVGAnimatedLength)

  Corresponds to attribute **'textLength'** on the given element.

- **lengthAdjust** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'lengthAdjust'** on the given element. The value must be one of the length adjust constants defined on this interface.

Operations:

- long **getNumberOfChars**()

  Returns the total number of characters available for rendering within the current element, which includes referenced characters from **'tref'** reference, regardless of whether they will be rendered. Effectively, this is equivalent to the length of the Node::textContent attribute from DOM Level 3 Core ([DOM3], section 1.4), if that attribute also expanded **'tref'** elements.

  Returns
  Total number of characters.

- float **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.

  Returns
  The text advance distance.

- float **getSubStringLength**(in unsigned long *charnum*, in unsigned long *nchars*)

  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. If multiple consecutive characters are rendered inseparably (e.g., as a single glyph or a sequence of glyphs, or because the range en-

compasses half of a surrogate pair), and nchars is greater than 0 then the measured range shall be expanded so that each of the inseparable characters are included.

Parameters

- unsigned long *charnum*
  The index of the first character in the substring, where the first character has an index of 0.
- 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.

Returns

The text advance distance.

Exceptions

- DOMException, code INDEX_SIZE_ERR
  Raised if *charnum* or *nchars* is negative or if *charnum* is greater than or equal to the number of characters at this node.

- SVGPoint **getStartPositionOfChar**(in unsigned long *charnum*)

  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

  - unsigned long *charnum*
    The index of the character, where the first character has an index of 0.

  Returns

  The character's start position.

  Exceptions

  - DOMException, code 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.

- SVGPoint **getEndPositionOfChar**(in unsigned long *charnum*)

  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

  - unsigned long *charnum*
    The index of the character, where the first character has an index of 0.

  Returns
  The character's end position.

  Exceptions

  - DOMException, code 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.

- SVGRect **getExtentOfChar**(in unsigned long *charnum*)

  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

  - unsigned long *charnum*
    The index of the character, where the first character has an index of 0.

  Returns
  The rectangle which encloses all of the rendered glyph(s).

  Exceptions

  - DOMException, code 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.

- float **getRotationOfChar**(in unsigned long *charnum*)

  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

  - unsigned long *charnum*
    The index of the character, where the first character has an index of 0.

  Returns
  The rotation angle.

  Exceptions

  - DOMException, code 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.

- long **getCharNumAtPosition**(in SVGPoint *point*)

  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

  - SVGPoint *point*
    A point in user space.

  Returns
  The index of the character which is at the given point, where the first character has an index of 0.

- void **selectSubString**(in unsigned long *charnum*, in unsigned long *nchars*)

  Causes the specified substring to be selected just as if the user selected the substring interactively.

  Parameters

  - unsigned long *charnum*
    The index of the start character which is at the given point, where the first character has an index of 0.
  - 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.

  Exceptions

  - DOMException, code INDEX_SIZE_ERR
    Raised if *charnum* or *nchars* is negative or if *charnum* is greater than or equal to the number of characters at this node.

10.17.2 Interface SVGTextPositioningElement

The SVGTextPositioningElement interface is inherited by text-related interfaces: SVGTextElement, SVGTSpanElement, SVGTRefElement and SVGAltGlyphElement.

```
interface SVGTextPositioningElement : SVGTextContentElement {
  readonly attribute SVGAnimatedLengthList x;
  readonly attribute SVGAnimatedLengthList y;
  readonly attribute SVGAnimatedLengthList dx;
  readonly attribute SVGAnimatedLengthList dy;
  readonly attribute SVGAnimatedNumberList rotate;
};
```

Attributes:

- **x** (readonly SVGAnimatedLengthList)

  Corresponds to attribute 'x' on the given element.

- **y** (readonly SVGAnimatedLengthList)

  Corresponds to attribute 'y' on the given element.

- **dx** (readonly SVGAnimatedLengthList)

  Corresponds to attribute 'dx' on the given element.

- **dy** (readonly SVGAnimatedLengthList)

  Corresponds to attribute 'dy' on the given element.

- **rotate** (readonly SVGAnimatedNumberList)

  Corresponds to attribute 'rotate' on the given element.

10.17.3 Interface SVGTextElement

The SVGTextElement interface corresponds to the 'text' element.

```
interface SVGTextElement : SVGTextPositioningElement,
                           SVGTransformable {
};
```

10.17.4 Interface SVGTSpanElement

The SVGTSpanElement interface corresponds to the 'tspan' element.

```
interface SVGTSpanElement : SVGTextPositioningElement {
};
```

10.17.5 Interface SVGTRefElement

The SVGTRefElement interface corresponds to the 'tref' element.

```
interface SVGTRefElement : SVGTextPositioningElement,
                           SVGURIReference {
};
```

10.17.6 Interface SVGTextPathElement

The SVGTextPathElement interface corresponds to the 'textPath' element.

```
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;
};
```

Constants in group "textPath Method Types":

- **TEXTPATH_METHODTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'align'**.

- **TEXTPATH_METHODTYPE_STRETCH** (unsigned short)

  Corresponds to value **'stretch'**.

Constants in group "textPath Spacing Types":

- **TEXTPATH_SPACINGTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'auto'**.

- **TEXTPATH_SPACINGTYPE_EXACT** (unsigned short)

  Corresponds to value **'exact'**.

Attributes:

- **startOffset** (readonly SVGAnimatedLength)

  Corresponds to attribute **'startOffset'** on the given **'textPath'** element.

- **method** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'method'** on the given **'textPath'** element.

- **spacing** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'spacing' on the given 'textPath' element.

10.17.7 Interface SVGAltGlyphElement

The SVGAltGlyphElement interface corresponds to the 'altGlyph' element.

```
interface SVGAltGlyphElement : SVGTextPositioningElement,
                               SVGURIReference {
  attribute DOMString glyphRef setraises(DOMException);
  attribute DOMString format setraises(DOMException);
};
```

Attributes:

- **glyphRef** (DOMString)

  Corresponds to attribute 'glyphRef' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **format** (DOMString)

  Corresponds to attribute 'format' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

10.17.8 Interface SVGAltGlyphDefElement

The SVGAltGlyphDefElement interface corresponds to the 'altGlyphDef' element.

```
interface SVGAltGlyphDefElement : SVGElement {
};
```

10.17.9 Interface SVGAltGlyphItemElement

The SVGAltGlyphItemElement interface corresponds to the 'altGlyphItem' element.

```
interface SVGAltGlyphItemElement : SVGElement {
};
```

10.17.10 Interface SVGGlyphRefElement

The SVGGlyphRefElement interface corresponds to the 'glyphRef' element.

```
interface SVGGlyphRefElement : SVGElement,
                               SVGURIReference,
                               SVGStylable {
  attribute DOMString glyphRef setraises(DOMException);
  attribute DOMString format setraises(DOMException);
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float dx setraises(DOMException);
  attribute float dy setraises(DOMException);
};
```

Attributes:

- **glyphRef** (DOMString)

  Corresponds to attribute 'glyphRef' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **format** (DOMString)

  Corresponds to attribute 'format' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **x** (float)

  Corresponds to attribute 'x' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **y** (float)

  Corresponds to attribute 'y' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **dx** (float)

  Corresponds to attribute 'dx' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **dy** (float)

  Corresponds to attribute 'dy' on the given element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

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 IRI 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:

```
                none |
                currentColor |
<paint>:        <color> [<icccolor>] |
                <funciri> [ none | currentColor | <color> [<icccolor>] ] |
                inherit
```

none
: Indicates that no paint is applied.

currentColor
: Indicates that painting is done using the current animated value of 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> [<icccolor>]
: <color> is the explicit color (in the sRGB color space [SRGB]) to be used to paint the current object. SVG supports all of the syntax alternatives for <color> defined in CSS2 ([CSS2], section 4.3.6), with the exception that SVG contains an expanded list of recognized color keywords names. If an optional ICC color specification [ICC42] is provided, then the user agent searches the color profile description database for a color profile description entry whose name descriptor matches the <name> part of the <icccolor> and uses the last matching entry that is found. (If no match is found, then the ICC color specification is ignored.) The comma and/or whitespace separated list of <number>s is a set of ICC-profile-specific color values. (In most cases, the <number>s will be in the range 0 to 1.) On platforms which support ICC-based color management, the <icccolor> 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**'). For more on ICC-based colors, refer to Color profile descriptions.

<funciri>
[none |
currentColor |
<color> [<icccolor>]]
: The <funciri> is used to identify a paint server such as a gradient, a pattern or a custom paint defined by an extension (see Extensibility). The <funciri> points to the paint server (e.g., a gradient or pattern) to be used to paint the current object. If the IRI 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 <funciri> (i.e., **none | currentColor | <color> [<icccolor>]** 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'

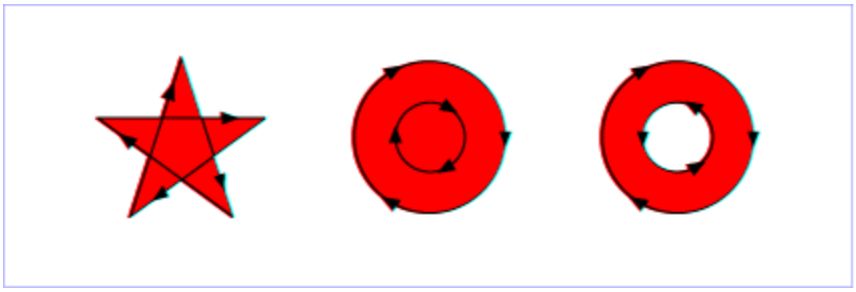
Value:	nonzero | evenodd | inherit
Initial:	nonzero
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

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:

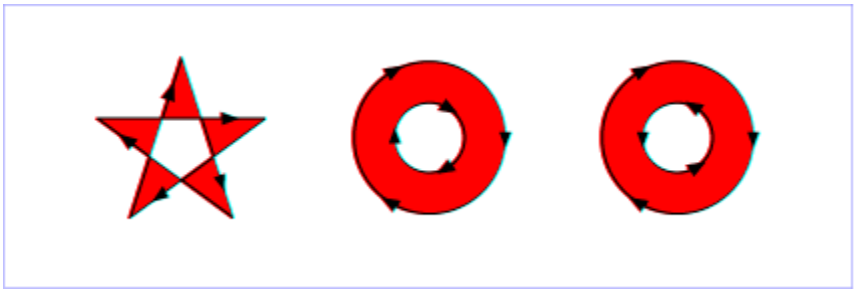
nonzero

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:



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:



(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, as a <number>. 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 shall not be stroked. Any zero length subpath shall not be stroked if the '**stroke-linecap**' property has a value of **butt** but shall be stroked if the '**stroke-linecap**' property has a value of **round** or **square**, producing respectively a circle or a square centered at the given point. Examples of zero length subpaths include **'M 10,10 L 10,10'**, **'M 20,20 h 0'**, **'M 30,30 z'** and **'M 40,40 c 0,0 0,0 0,0'**.

'**stroke-width**'

Value:	<percentage> | <length> | inherit
Initial:	1
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	Yes
Media:	visual
Animatable:	yes

This property specifies 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'

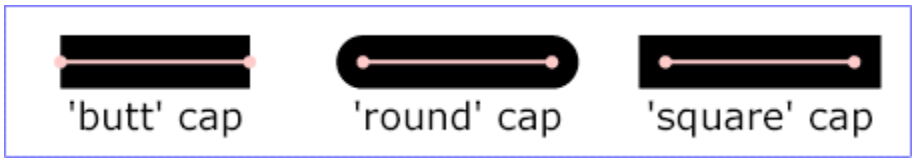
Value:	butt | round | square | inherit
Initial:	butt
Applies to:	shapes and text content elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

'**stroke-linecap**' specifies the shape to be used at the end of open subpaths when they are stroked. For further details see the path implementation notes.

butt
See drawing below.

round
See drawing below.

square
See drawing below.



'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. For further details see the path implementation notes.

miter
See drawing below.

round
See drawing below.

bevel

See drawing below.

'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-width**'. When the limit is exceeded, the join is converted from a miter to a bevel.

<miterlimit>

The limit on the ratio of the miter length to the '**stroke-width**'. The value of **<miterlimit>** must be a <number> greater than or equal to 1. Any other value is an error (see Error processing).

The ratio of miter length (distance between the outer tip and the inner corner of the miter) to '**stroke-width**' is directly related to the angle (theta) between the segments in user space by the formula:

```
miterLength / stroke-width = 1 / sin ( theta / 2 )
```

For example, a miter limit of 1.414 converts miters to bevels for theta less than 90 degrees, a limit of 4.0 converts them for theta less than approximately 29 degrees, and a limit of 10.0 converts them for theta less than approximately 11.5 degrees.

'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 and/or white space separated <length>s and <percentage>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 and/or white space separated <length>s (which can have a unit identifier) and <percentage>s. A percentage represents a distance as a percentage of the current viewport (see Units). A negative value is an error (see Error processing). If the sum of the values is zero, then the stroke is rendered as if a value of **none** were specified. For further details see the path implementation notes.

The grammar for <dasharray> is as follows:

```
dasharray ::= (length | percentage) (comma-wsp dasharray)?
```

'stroke-dashoffset'

Value:	<percentage> | <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.

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, as a <number>. 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 of the '**display**' property is the CSS2 definition ([CSS2], section 9.2.6).

'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 of the '**visibility**' property is the CSS2 definition ([CSS2], section 11.2).

11.6 Markers

11.6.1 Introduction

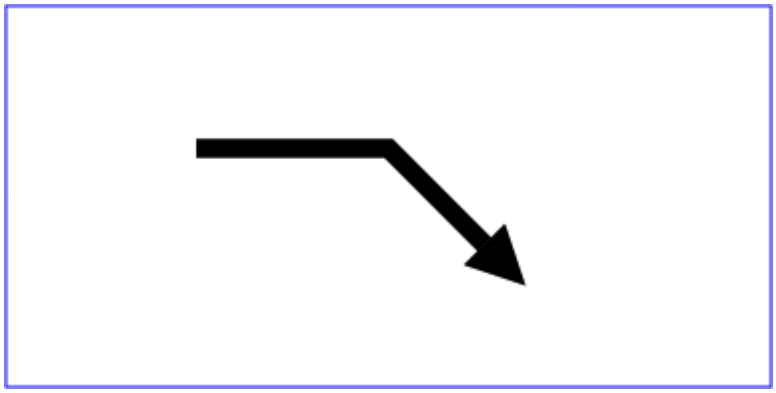
A marker is a symbol which is attached to one or more vertices of 'path', 'line', 'polyline' and 'polygon' elements. Typically, markers are used to make arrowheads or polymarkers. Arrowheads can be defined by attaching a marker to the start or end vertices of 'path', 'line' or 'polyline' elements. Polymarkers 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000" version="1.1"
     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



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.

Categories: 'marker'

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'
- 'cursor'
- 'filter'
- 'font'
- 'font-face'
- 'foreignObject'
- 'image'
- 'marker'

'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:
core attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'viewBox'
'preserveAspectRatio'
'refX'
'refY'
'markerUnits'
'markerWidth'
'markerHeight'
'orient'

DOM Interfaces:
SVGMarkerElement

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 co-

ordinate 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.
   - If an incoming and an outgoing vertex produce a zero vector the direction of marker is undefined.
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 <angle> value 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).

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-mid**'
'**marker-end**'

Value:	none | <funciri> | inherit
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).

<funciri>
: The <funciri> is a Functional IRI reference to the 'marker' element which shall be used as the arrowhead symbol or polymarker at the given vertex or vertices. If the IRI 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], chapter 6) 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000" version="1.1"
     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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000" version="1.1"
     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>
</svg>
```

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 with either ‘animate’ or ‘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. Having chosen the appropriate color space, component-wise linear interpolation is used.

The conversion formulas between the sRGB color space (i.e., nonlinear with 2.2 gamma curve) and the lin-

earized RGB color space (i.e., color values expressed as sRGB tristimulus values without a gamma curve) can be found in the sRGB specification [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, **'animate'** 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, 'animate' 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="7cm" height="2cm" viewBox="0 0 700 200"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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

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

11.9.1 Interface SVGPaint

The SVGPaint interface corresponds to basic type <paint> and represents the values of properties '**fill**' and '**stroke**'.

Note: The SVGPaint interface is deprecated, and may be dropped from future versions of the SVG specification.

```
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);
};
```

Constants in group "Paint Types":

- **SVG_PAINTTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  An sRGB color has been specified without an alternative ICC color specification.

- **SVG_PAINTTYPE_RGBCOLOR_ICCCOLOR** (unsigned short)

  An sRGB color has been specified along with an alternative ICC color specification.

- **SVG_PAINTTYPE_NONE** (unsigned short)

  Corresponds to a **none** value on a <paint> specification.

- **SVG_PAINTTYPE_CURRENTCOLOR** (unsigned short)

  Corresponds to a **currentColor** value on a <paint> specification.

- **SVG_PAINTTYPE_URI_NONE** (unsigned short)

  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** (unsigned short)

  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** (unsigned short)

  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** (unsigned short)

  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** (unsigned short)

  Only a URI has been specified.

Attributes:

- **paintType** (readonly unsigned short)

  The type of paint, identified by one of the SVG_PAINTTYPE_* constants defined on this interface.

- **uri** (readonly DOMString)

  When the paintType specifies a URI, this attribute holds the URI string. When the paintType does not specify a URI, this attribute is null.

Operations:

- void **setUri**(in DOMString *uri*)

  Sets the paintType to SVG_PAINTTYPE_URI_NONE and sets uri to the specified value.

  Parameters

  - DOMString *uri*
    The URI for the desired paint server.

- void **setPaint**(in unsigned short *paintType*, in DOMString *uri*, in DOMString *rgbColor*, in DOMString *iccColor*)

  Sets the paint as specified by the parameters. If *paintType* requires a URI, then *uri* must be non-null; otherwise, *uri* must be null. If *paintType* requires an RGBColor, then *rgbColor* must be a string that matches <color>; otherwise, *rgbColor* must be null. If *paintType* requires an SVGICCColor, then *iccColor* must be a string that matches <icccolor>; otherwise, *iccColor* must be null.

  Parameters

  - unsigned short *paintType*
    One of the defined constants for paintType.
  - DOMString *uri*
    The URI for the desired paint server, or null.
  - DOMString *rgbColor*
    The specification of an sRGB color, or null.

- DOMString *iccColor*
  The specification of an ICC color, or null.

Exceptions

- SVGException, code SVG_INVALID_VALUE_ERR
  Raised if one of the parameters has an invalid value.

11.9.2 Interface SVGMarkerElement

The SVGMarkerElement interface corresponds to the 'marker' element.

```
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() raises(DOMException);
  void setOrientToAngle(in SVGAngle angle) raises(DOMException);
};
```

Constants in group "Marker Unit Types":

- **SVG_MARKERUNITS_UNKNOWN** (unsigned short)

  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** (unsigned short)

  The value of attribute 'markerUnits' is **'userSpaceOnUse'**.

- **SVG_MARKERUNITS_STROKEWIDTH** (unsigned short)

  The value of attribute 'markerUnits' is **'strokeWidth'**.

Constants in group "Marker Orientation Types":

- **SVG_MARKER_ORIENT_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Attribute 'orient' has value **'auto'**.

- **SVG_MARKER_ORIENT_ANGLE** (unsigned short)

  Attribute 'orient' has an angle value.

Attributes:

- **refX** (readonly SVGAnimatedLength)

  Corresponds to attribute 'refX' on the given 'marker' element.

- **refY** (readonly SVGAnimatedLength)

  Corresponds to attribute 'refY' on the given 'marker' element.

- **markerUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'markerUnits' on the given 'marker' element. One of the Marker Unit Types defined on this interface.

- **markerWidth** (readonly SVGAnimatedLength)

  Corresponds to attribute 'markerWidth' on the given 'marker' element.

- **markerHeight** (readonly SVGAnimatedLength)

  Corresponds to attribute 'markerHeight' on the given 'marker' element.

- **orientType** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'orient' on the given 'marker' element. One of the Marker Orientation Types defined on this interface.

- **orientAngle** (readonly SVGAnimatedAngle)

  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.

Operations:

- void **setOrientToAuto**()

  Sets the value of attribute 'orient' to **'auto'**.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- void **setOrientToAngle**(in SVGAngle *angle*)

  Sets the value of attribute 'orient' to the given angle.

  Parameters

  - SVGAngle *angle*
    The angle value to use for attribute 'orient'.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

12 Color

Contents

12.1 Introduction

All SVG colors are specified in the sRGB color space [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 [ICC42] as described in Specifying paint. 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; otherwise, the RGB fallback colors must be used. Note that, in this specification, 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**’ and ‘**lighting-color**’ properties.

'color'

Value:	<color> | inherit
Initial:	depends on user agent
Applies to:	elements to which properties **'fill'**, **'stroke'**, **'stop-color'**, **'flood-color'** and **'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 ([CSS2], section 14.1).

12.3 Color profile descriptions

12.3.1 Overview of color profile descriptions

The International Color Consortium has established a standard, the ICC Profile [ICC42], 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 of 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 to style the SVG content [CSS2])

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

'color-profile'

Categories:
None

Content model:
Any number of the following elements, in any order:
descriptive elements

Attributes:
core attributes
xlink attributes
'local'
'name'
'rendering-intent'
'xlink:href'

DOM Interfaces:
SVGColorProfileElement

Attribute definitions:

`xlink:href = "<iri>"`
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 ([CSS2], section D.2), 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.

12.3.4 The CSS @**color-profile** rule

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'

Values: sRGB | <local-profile> | <iri> | (<local-profile> <iri>) | inherit
Initial: sRGB
Media: visual

sRGB

The source profile is the sRGB color space. For consistency with CSS lexical scanning and parsing rules ([CSS2], section D.2), 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.)

<iri>

The source profile is a IRI reference to a color profile.

(<local-profile> <iri>)

Two profiles are specified. If <local-profile> cannot be found on the local system, then the <iri> is used.

'name'

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'

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 The 'color-profile' property

'color-profile'

Value:	auto | sRGB | <name> | <iri> | 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 ([CSS2], section D.2), 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.

<iri>

A IRI reference to the source color profile. The referenced color profile overrides an embedded profile inside the image.

12.4 DOM interfaces

12.4.1 Interface SVGColorProfileElement

The SVGColorProfileElement interface corresponds to the 'color-profile' element.

```
interface SVGColorProfileElement : SVGElement,
                                   SVGURIReference,
                                   SVGRenderingIntent {
  attribute DOMString local;
  attribute DOMString name;
  attribute unsigned short renderingIntent;
};
```

Attributes:

- **local** (DOMString)

  Corresponds to attribute 'local' on the given element.

- **name** (DOMString)

  Corresponds to attribute 'name' on the given element.

- **renderingIntent** (unsigned short)

  Corresponds to attribute 'rendering-intent' on the given element. The value of this attribute is the value of the the RENDERING_INTENT_* constant defined on SVGRenderingIntent that corresponds to the value of the 'rendering-intent' attribute.

12.4.2 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.

```
interface SVGColorProfileRule : SVGCSSRule,
                                SVGRenderingIntent {
  attribute DOMString src setraises(DOMException);
  attribute DOMString name setraises(DOMException);
  attribute unsigned short renderingIntent setraises(DOMException);
};
```

Attributes:

- **src** (DOMString)

  Corresponds to descriptor **src** within an @color-profile rule.

Exceptions on setting

- DOMException, code NO_MODIFICATION_ALLOWED_ERR
  Raised on an attempt to change the value of a read only attribute.

- **name** (DOMString)

  Corresponds to descriptor '**name**' within an @color-profile rule.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

- **renderingIntent** (unsigned short)

  The type of rendering intent, identified by one of the SVGRenderingIntent constants.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

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 (using <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 an IRI reference on a '**fill**' or '**stroke**' property.

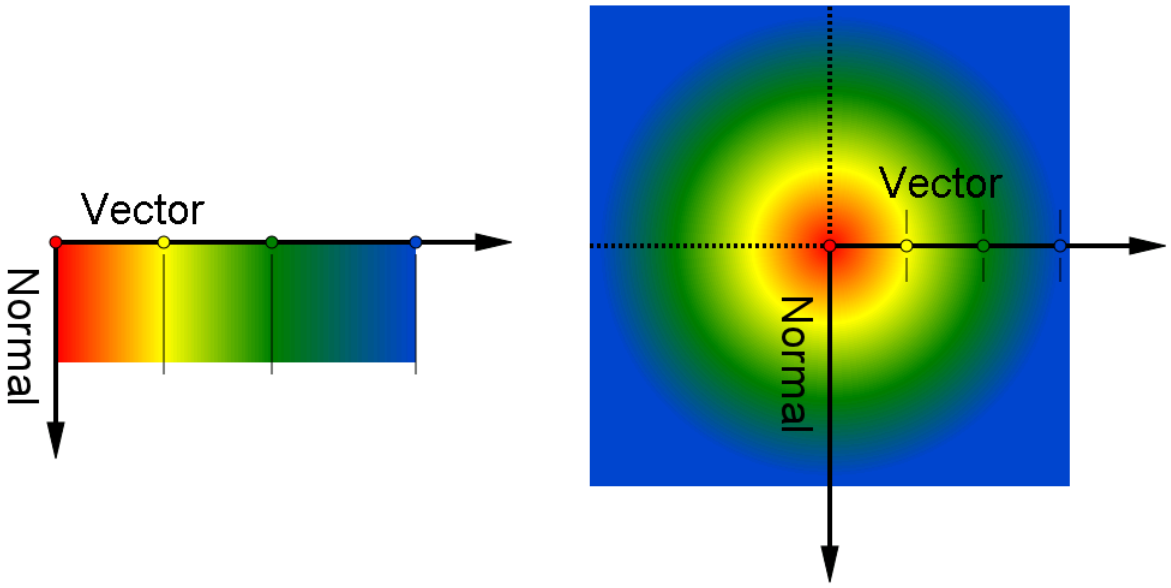
13.2 Gradients

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.

The angle of the color transitions along the gradient vector is defined by the gradient normal. Before any transforms are applied to the gradient or its referencing graphics element, the gradient normal is perpendicular with the gradient vector. If a graphics element references a gradient, conceptually the graphics element should take a copy of the gradient vector and gradient normal and treat it as part of its own geometry. Any transformations applied to the graphics element geometry also apply to the copied gradient vector and gradient normal. Any gradient transforms that are specified on the reference gradient are applied before any graphics element transformations are applied to the gradient.



13.2.2 Linear gradients

Linear gradients are defined by a 'linearGradient' element.

Categories: 'linearGradient'

Gradient element

Content model:

Any number of the following elements, in any order:

descriptive elements
'animate'
'animateTransform'
'set'
'stop'

Attributes:

core attributes
presentation attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'x1'
'y1'
'x2'
'y2'
'gradientUnits'
'gradientTransform'
'spreadMethod'
'xlink:href'

DOM Interfaces:

SVGLinearGradientElement

Attribute definitions:

`gradientUnits` = "***userSpaceOnUse*** | ***objectBoundingBox***"

Defines the coordinate system for attributes 'x1', 'y1', 'x2' and 'y2'.
If **gradientUnits="userSpaceOnUse"**, 'x1', 'y1', 'x2' and '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' and '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 normal of the linear gradient is 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 gradient normal which is initially perpendicular to the gradient vector within object bounding box space may render non-perpendicular relative to the gradient vector in user space. If the gradient vector is parallel to one of the axes of the bounding box, the gradient normal will remain perpendicular. This transformation is 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' and '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' and '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` = "**<iri>**"

An IRI 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 'xlink: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 'xlink:href' attribute, then the current element can inherit those attributes or gradient stops.
Animatable: yes.

Percentages are allowed for 'x1', 'y1', 'x2' and '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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400" version="1.1"
     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

13.2.3 Radial gradients

Radial gradients are defined by a 'radialGradient' element.

Categories: 'radialGradient'

Gradient element

Content model:

Any number of the following elements, in any order:

- descriptive elements
- 'animate'
- 'animateTransform'
- 'set'
- 'stop'

Attributes:

- core attributes
- presentation attributes
- xlink attributes
- 'class'
- 'style'
- 'externalResourcesRequired'
- 'cx'
- 'cy'
- 'r'
- 'fx'
- 'fy'
- 'gradientUnits'
- 'gradientTransform'
- 'spreadMethod'
- 'xlink:href'

DOM Interfaces:
SVGRadialGradientElement

Attribute definitions:

`gradientUnits` = "***userSpaceOnUse*** | ***objectBoundingBox***"
Defines the coordinate system for attributes 'cx', 'cy', 'r', 'fx' and 'fy'.
If **gradientUnits="userSpaceOnUse"**, 'cx', 'cy', 'r', 'fx' and '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' and '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' and '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' and '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 the presentational value of 'cx' for the element whether the value for 'cx' was inherited or not. If the element references an element that specifies a value for 'fx', then the value of 'fx' is inherited from the referenced element.
Animatable: yes.

`fy` = "**<coordinate>**"
See 'fx'.
If attribute 'fy' is not specified, 'fy' will coincide with the presentational vlaue of 'cy' for the element whether the value for 'cy' was inherited or not. If the element references an element that specifies a value for 'fy', then the value of 'fy' is inherited from the referenced element.
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` = "**<iri>**"
An IRI 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 'xlink: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 'xlink:href' attribute, then the current element can inherit those attributes or gradient stops.
Animatable: yes.

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'.

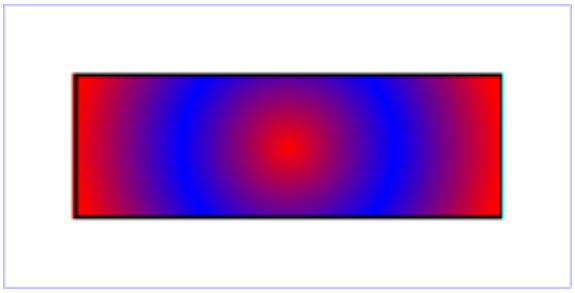
Properties inherit into the 'radialGradient' element from its ancestors; properties do *not* inherit from the element referencing the 'radialGradient' element.

'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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400" version="1.1"
     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



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.

Categories: 'stop'
None

Content model:
Any number of the following elements, in any order:
'animate'
'animateColor'
'set'

Attributes:
core attributes
presentation attributes
'class'
'style'
'offset'

DOM Interfaces:
SVGStopElement

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.

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> <icccolor> | 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:	<opacity-value> | 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*.

Categories: 'pattern'

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'
- 'cursor'
- 'filter'
- 'font'
- 'font-face'
- 'foreignObject'
- 'image'
- 'marker'
- 'mask'
- 'pattern'
- 'script'
- 'style'
- 'switch'
- 'text'
- 'view'

Attributes:

- conditional processing attributes
- core attributes
- presentation attributes
- xlink attributes
- 'class'
- 'style'
- 'externalResourcesRequired'

'viewBox'
'preserveAspectRatio'
'x'
'y'
'width'
'height'
'patternUnits'
'patternContentUnits'
'patternTransform'
'xlink:href'

DOM Interfaces:
SVGPatternElement

Attribute definitions:

`patternUnits` = "***userSpaceOnUse*** | ***objectBoundingBox***"
Defines the coordinate system for attributes 'x', 'y', 'width' and 'height'.
If **patternUnits="userSpaceOnUse"**, 'x', 'y', 'width' and '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' and '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 spe-

cified.
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' and '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 = "<iri>"`
An IRI 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 'xlink: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 'xlink:href' attribute, then the current element can inherit those attributes or children.
Animatable: yes.

preserveAspectRatio = "[defer] <align> [<meetOrSlice>]"
See 'preserveAspectRatio'.
If the attribute is not specified, then the effect is as if a value of **xMidYMid meet** were specified.
Animatable: yes.

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. Note that if the '**overflow**' property is set to **visible** the rendering behavior for the pattern is undefined. 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**.

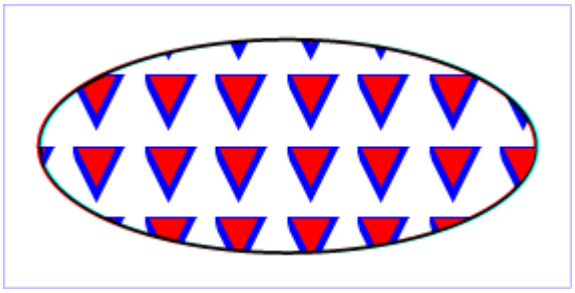
Event attributes and event listeners attached to the contents of a 'pattern' element are not processed; only the rendering aspects of 'pattern' elements are processed.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg width="8cm" height="4cm" viewBox="0 0 800 400" version="1.1"
     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



13.4 DOM interfaces

13.4.1 Interface SVGGradientElement

The SVGGradientElement interface is a base interface used by SVGLinearGradientElement and SVGRadialGradientElement.

```
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;
};
```

Constants in group "Spread Method Types":

- **SVG_SPREADMETHOD_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'pad'**.

- **SVG_SPREADMETHOD_REFLECT** (unsigned short)

  Corresponds to value **'reflect'**.

- **SVG_SPREADMETHOD_REPEAT** (unsigned short)

  Corresponds to value **'repeat'**.

Attributes:

- **gradientUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'gradientUnits'** on the given element. Takes one of the constants defined in SVGUnitTypes.

- **gradientTransform** (readonly SVGAnimatedTransformList)

  Corresponds to attribute **'gradientTransform'** on the given element.

- **spreadMethod** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'spreadMethod'** on the given element. One of the Spread Method Types defined on this interface.

13.4.2 Interface SVGLinearGradientElement

The SVGLinearGradientElement interface corresponds to the **'linearGradient'** element.

```
interface SVGLinearGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength x1;
  readonly attribute SVGAnimatedLength y1;
  readonly attribute SVGAnimatedLength x2;
  readonly attribute SVGAnimatedLength y2;
};
```

Attributes:

- **x1** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x1' on the given 'linearGradient' element.

- **y1** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y1' on the given 'linearGradient' element.

- **x2** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x2' on the given 'linearGradient' element.

- **y2** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y2' on the given 'linearGradient' element.

13.4.3 Interface SVGRadialGradientElement

The SVGRadialGradientElement interface corresponds to the 'radialGradient' element.

```
interface SVGRadialGradientElement : SVGGradientElement {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
  readonly attribute SVGAnimatedLength fx;
  readonly attribute SVGAnimatedLength fy;
};
```

Attributes:

- **cx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cx' on the given 'radialGradient' element.

- **cy** (readonly SVGAnimatedLength)

  Corresponds to attribute 'cy' on the given 'radialGradient' element.

- **r** (readonly SVGAnimatedLength)

  Corresponds to attribute 'r' on the given 'radialGradient' element.

- **fx** (readonly SVGAnimatedLength)

  Corresponds to attribute 'fx' on the given 'radialGradient' element.

- **fy** (readonly SVGAnimatedLength)

  Corresponds to attribute 'fy' on the given 'radialGradient' element.

13.4.4 Interface SVGStopElement

The SVGStopElement interface corresponds to the 'stop' element.

```
interface SVGStopElement : SVGElement,
                           SVGStylable {
  readonly attribute SVGAnimatedNumber offset;
};
```

Attributes:

- **offset** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'offset' on the given 'stop' element.

13.4.5 Interface SVGPatternElement

The SVGPatternElement interface corresponds to the 'pattern' element.

```
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:

- **patternUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'patternUnits' on the given 'pattern' element. Takes one of the constants defined in SVGUnitTypes.

- **patternContentUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'patternContentUnits' on the given 'pattern' element. Takes one of the constants defined in SVGUnitTypes.

- **patternTransform** (readonly SVGAnimatedTransformList)

  Corresponds to attribute 'patternTransform' on the given 'pattern' element.

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'pattern' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'pattern' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'pattern' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'pattern' element.

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 ([CSS2], section 11.1.1); 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], chapter 9) 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 ([CSS2], section 11.1.1).
- When an outermost svg element is stand-alone or embedded inline within a parent XML grammar which does not use CSS layout or XSL formatting, 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', and this applies also to the root 'svg' element; however, for child elements of an SVG document, SVG's user agent style sheet overrides this initial value and sets 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 ([CSS2], section 11.1.2). 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: the 'clipPath' element

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. If a child element is made invisible by '**display**' or '**visibility**' it does not contribute to the clipping path. 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'.

For a given graphics element, the actual clipping path used will be the intersection of the clipping path spe-

cified 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.
- An empty clipping path will completely clip away the element that had the '**clip-path**' property applied.

Categories: 'clipPath'

None

Content model:

Any number of the following elements, in any order:

descriptive elements
animation elements
shape elements
'text'
'use'

Attributes:

conditional processing attributes
core attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'transform'
'clipPathUnits'

DOM Interfaces:

SVGClipPathElement

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.

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:	<funciri> | none | inherit
Initial:	none
Applies to:	container elements, graphics elements and 'clipPath'
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

<funciri>

An IRI reference to another graphical object within the same SVG document fragment which will be used as the clipping path. If the IRI reference is not valid (e.g it points to an object that doesn't exist or the object is not a 'clipPath' element) the '**clip-path**' property must be treated as if it hadn't been specified.

'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>
```

14.3.6 Clipping paths, geometry, and pointer events

A clipping path is conceptually equivalent to a custom viewport for the referencing element. Thus, it affects the rendering of an element, but not the element's inherent geometry. The bounding box of a clipped element (that is, an element which references a 'clipPath' element via a '**clip-path**' property, or a child of the referencing element) must remain the same as if it were not clipped.

By default, pointer-events must not be dispatched on the clipped (non-visible) regions of a shape. For example, a circle with a radius of 10 which is clipped to a circle with a radius of 5 will not receive 'click' events outside the smaller radius. Later versions of SVG may define new properties to enable fine-grained control over the interactions between hit testing and clipping.

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. First a luminance value is computed from the color channel values:

- If the computed value of '**color-interpolation**' on the 'mask' element is **linearRGB**, first convert 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 computed value of '**color-interpolation**' on the 'mask' element is **sRGB** then the luminance value is calculated by taking the non-premultiplied RGB color values, applying the luminance-to-alpha coefficients (as defined in the 'feColorMatrix' filter primitive) to convert the RGB color values to luminance values.

Finally if the graphics object also includes an alpha channel, then the computed 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="3cm" viewBox="0 0 800 300" version="1.1"
     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

Categories: ‘mask’

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- ‘a’
- ‘altGlyphDef’
- ‘clipPath’

'color-profile'
'cursor'
'filter'
'font'
'font-face'
'foreignObject'
'image'
'marker'
'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:

conditional processing attributes
core attributes
presentation attributes
'class'
'style'
'externalResourcesRequired'
'x'
'y'
'width'
'height'
'maskUnits'
'maskContentUnits'

DOM Interfaces:

SVGMaskElement

Attribute definitions:

`maskUnits` = "***userSpaceOnUse*** | ***objectBoundingBox***"

Defines the coordinate system for attributes 'x', 'y', 'width' and 'height'.

If **maskUnits="userSpaceOnUse"**, 'x', 'y', 'width' and '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' and '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' and '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' and '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.

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 '**opacity**', '**filter**' and '**display**' properties do 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:	<funciri> | none | inherit
Initial:	none
Applies to:	container elements and graphics elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

<funciri>

A IRI 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**', which specifies the opacity of a fill operation;
- '**stroke-opacity**', which specifies the opacity of a stroking operation;
- '**stop-opacity**', which specifies the opacity of a gradient stop; and
- '**opacity**', which specifies object/group opacity and which is described in this section.

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:	<opacity-value> | inherit
Initial:	1
Applies to:	container elements (except 'mask') and graphics elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

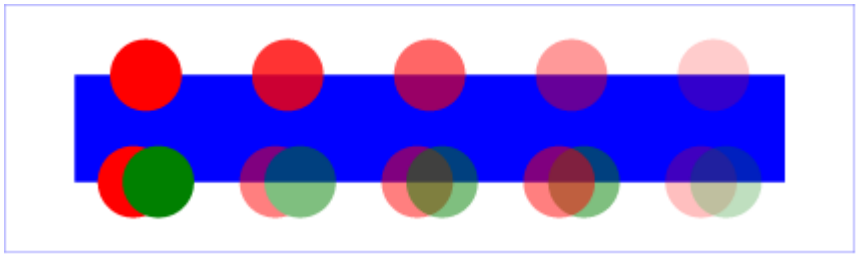
<opacity-value>

The uniform opacity setting to be applied across an entire object, as a <number>. 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 <opacity-value>. (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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="3.5cm" viewBox="0 0 1200 350"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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

14.6.1 Interface SVGClipPathElement

The SVGClipPathElement interface corresponds to the 'clipPath' element.

```
interface SVGClipPathElement : SVGElement,
                               SVGTests,
                               SVGLangSpace,
                               SVGExternalResourcesRequired,
                               SVGStylable,
                               SVGTransformable,
                               SVGUnitTypes {
  readonly attribute SVGAnimatedEnumeration clipPathUnits;
};
```

Attributes:

- **clipPathUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'clipPathUnits' on the given 'clipPath' element. Takes one of the constants defined in SVGUnitTypes.

14.6.2 Interface SVGMaskElement

The SVGMaskElement interface corresponds to the 'mask' element.

```
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:

- **maskUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'maskUnits' on the given 'mask' element. Takes one of the constants defined in SVGUnitTypes.

- **maskContentUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'maskContentUnits' on the given 'mask' element. Takes one of the constants defined in SVGUnitTypes.

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'mask' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'mask' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'mask' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'mask' element.

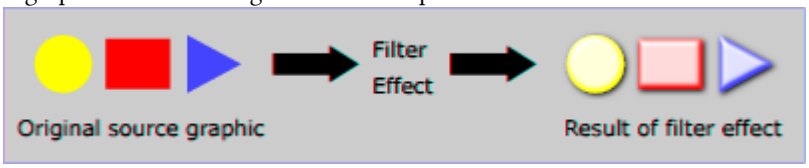
15 Filter Effects

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:



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 1.1//EN"
              "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="7.5cm" height="5cm" viewBox="0 0 200 120"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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>
```

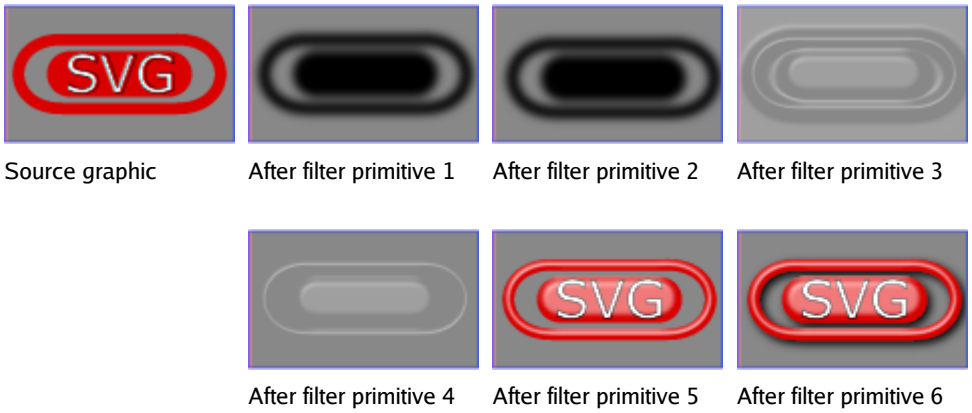
```
      <path fill="none" stroke="#D90000" stroke-width="10"
            d="M50,90 C0,90 0,30 50,30 L150,30 C200,30 200,90 150,90 z" />
      <path fill="#D90000"
            d="M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z" />
      <g fill="#FFFFFF" stroke="black" font-size="45" font-family="Verdana" >
        <text x="52" y="76">SVG</text>
      </g>
    </g>
  </g>
</svg>
```

Example filters01

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:



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:

Categories: 'filter'
None

Content model:
Any number of the following elements, in any order:
descriptive elements
filter primitive elements
'animate'
'set'

Attributes:
core attributes
presentation attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'x'
'y'
'width'
'height'
'filterRes'
'filterUnits'
'primitiveUnits'
'xlink:href'

DOM Interfaces:
SVGFilterElement

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). Note that if only one number was specified in a <number-optional-number> value this number is expanded out before the 'primitiveUnits' computation takes place.
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` = "<iri>"
: An IRI 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 'xlink: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 'xlink:href' attribute, then the current element can inherit those attributes or filter node specifications.
: *Animatable: yes.*

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:	<funciri> | none | inherit
Initial:	none
Applies to:	container elements (except 'mask') and graphics elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

<funciri>

An Functional IRI 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' · Defines the coordinate system for attributes 'x', 'y', 'width' and 'height'.

If **filterUnits="userSpaceOnUse"**, 'x', 'y', 'width' and 'height' represent values in the current user coordinate system in place at the time when the 'filter' is referenced (i.e., the user coordinate system for the element referencing the 'filter' via a **'filter'** property).

If **filterUnits="objectBoundingBox"**, then 'x', 'y', 'width' and '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 if a value of **'objectBoundingBox'** were specified.

Animatable: yes.

'x', 'y', 'width' **and** 'height' · These attributes 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' · This attribute takes the form `x-pixels [y-pixels]`, and indicates the width and height of the intermediate images in pixels. If not provided, then the user agent will use reasonable values to produce a high-quality result on the output device.

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.

Non-integer values are truncated, i.e rounded to the closest integer value towards zero.

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' and 'y', and percentages values greater than 100% for 'width' and '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**', defined below:

'**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 are <number> values that 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. 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="13.5cm" height="2.7cm" viewBox="0 0 1350 270"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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

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 in-

stead 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

With the exception of the 'in' attribute, all of the following attributes are available on all filter primitive elements:

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.

The 'in' attribute is available on all filter primitive elements that require an input.
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 as a special-case the percentages are relative to the dimensions of the filter region, thus making the the default filter primitive subregion equal to the filter region.

'x', 'y', 'width' and 'height' act as a hard clip clipping rectangle on both the filter primitive's input image(s) and the filter primitive result.

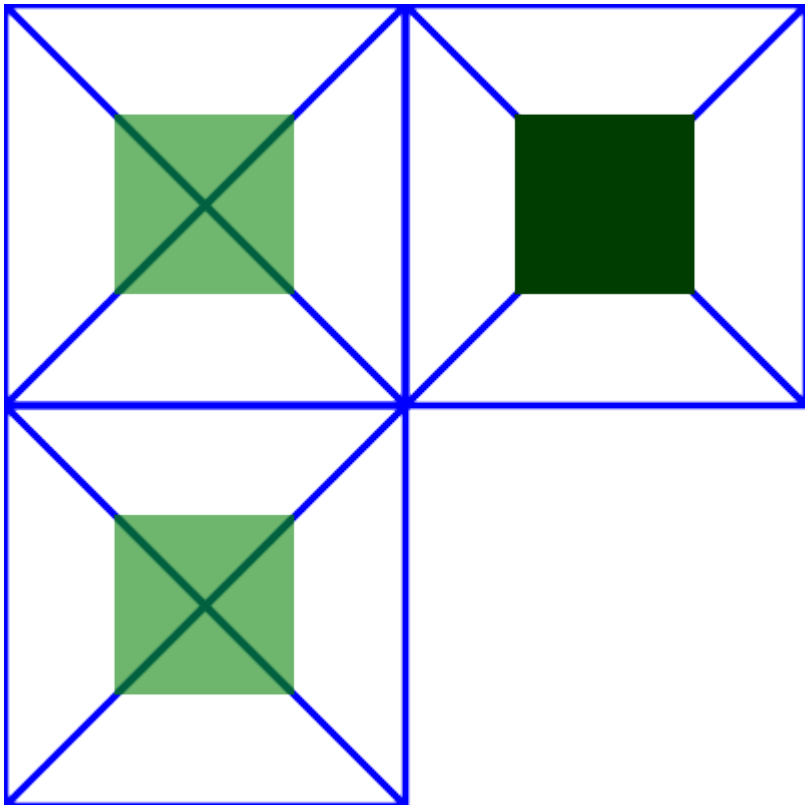
All intermediate offscreens are defined to not exceed the intersection of 'x', 'y', 'width' and 'height' with the filter region. The filter region and any of the 'x', 'y', 'width' and 'height' subregions are to be set up such that all offscreens are made big enough to accommodate any pixels which even 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.

Example primitive-subregion-01 demonstrates the effect of specifying a filter primitive subregion:

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <filter id="flood" x="0" y="0" width="100%" height="100%" primitiveUnits="objectBoundingBox">
       <feFlood x="25%" y="25%" width="50%" height="50%"
          flood-color="green" flood-opacity="0.75"/>
    </filter>
    <filter id="blend" primitiveUnits="objectBoundingBox">
       <feBlend x="25%" y="25%" width="50%" height="50%"
          in2="SourceGraphic" mode="multiply"/>
    </filter>
    <filter id="merge" primitiveUnits="objectBoundingBox">
       <feMerge x="25%" y="25%" width="50%" height="50%">
        <feMergeNode in="SourceGraphic"/>
        <feMergeNode in="FillPaint"/>
       </feMerge>
    </filter>
  </defs>

  <g fill="none" stroke="blue" stroke-width="4">
     <rect width="200" height="200"/>
     <line x2="200" y2="200"/>
     <line x1="200" y2="200"/>
  </g>
  <circle fill="green" filter="url(#flood)" cx="100" cy="100" r="90"/>

  <g transform="translate(200 0)">
    <g fill="none" stroke="blue" stroke-width="4">
       <rect width="200" height="200"/>
       <line x2="200" y2="200"/>
       <line x1="200" y2="200"/>
    </g>
    <circle fill="green" filter="url(#blend)" cx="100" cy="100" r="90"/>
  </g>

  <g transform="translate(0 200)">
    <g fill="none" stroke="blue" stroke-width="4">
       <rect width="200" height="200"/>
       <line x2="200" y2="200"/>
       <line x1="200" y2="200"/>
    </g>
    <circle fill="green" fill-opacity="0.5" filter="url(#merge)" cx="100" cy="100" r="90"/>
  </g>
</svg>
```

Example primitive-subregion-01



In the example above there are three rects that each have a cross and a circle in them. The circle element in each one has a different filter applied, but with the same filter primitive subregion. The filter output should be limited to the filter primitive subregion, so you should never see the circles themselves, just the rects that make up the filter primitive subregion.

- The upper left rect shows an 'feFlood' with **flood-opacity="75%"** so the cross should be visible through the green rect in the middle.
- The lower left rect shows an 'feMerge' that merges **SourceGraphic** with **FillPaint**. Since the circle has **fill-opacity="0.5"** it will also be transparent so that the cross is visible through the green rect in the middle.
- The upper right rect shows an 'feBlend' that has **mode="multiply"**. Since the circle in this case isn't transparent the result is totally opaque. The rect should be dark green and the cross should not be visible through it.

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'

Categories: 'feDistantLight'
Light source element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
'azimuth'
'elevation'

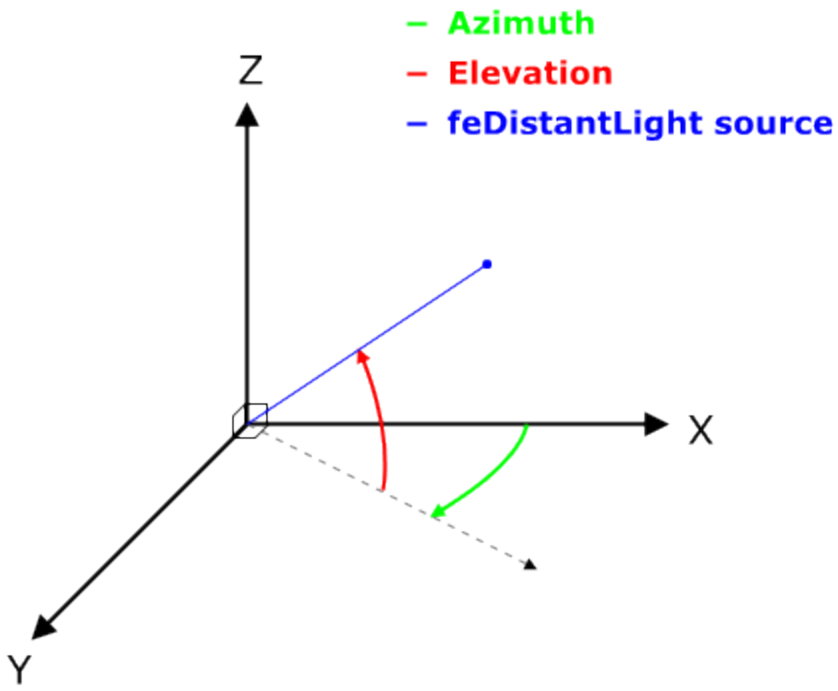
DOM Interfaces:
SVGFEDistantLightElement

Attribute definitions:

`azimuth` = "***<number>***"
Direction angle for the light source on the XY plane (clockwise), in degrees from the x axis.
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 from the XY plane towards the z axis, in degrees. Note the positive Z-axis points towards the viewer of the content.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

The following diagram illustrates the angles which 'azimuth' and 'elevation' represent in an XYZ coordinate system.



15.8.3 Light source 'fePointLight'

Categories: 'fePointLight'

Light source element

Content model:

Any number of the following elements, in any order:

'animate'

'set'

Attributes:

core attributes

'x'

'y'

'z'

DOM Interfaces:

SVGFEPointLightElement

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 and Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

15.8.4 Light source 'feSpotLight'

Categories: 'feSpotLight'
: Light source element

Content model:
: Any number of the following elements, in any order:
: 'animate'
: 'set'

Attributes:
: core attributes
: 'x'
: 'y'
: 'z'
: 'pointsAtX'
: 'pointsAtY'
: 'pointsAtZ'
: 'specularExponent'
: 'limitingConeAngle'

DOM Interfaces:
SVGFESpotLightElement

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 and 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 in the coordinate system established by attribute 'primitiveUnits' on the 'filter' element 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 and 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 in degrees 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.

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> [<icccolor>] | 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.

Categories: 'feBlend'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'in2'
'mode'

DOM Interfaces:
SVGFEBlendElement

Attribute definitions:

`mode` = "***normal | multiply | screen | darken | lighten***"
One of the image blending modes (see table below). If attribute 'mode' is not specified, then the effect is as if a value of **normal** were specified.
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.

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:

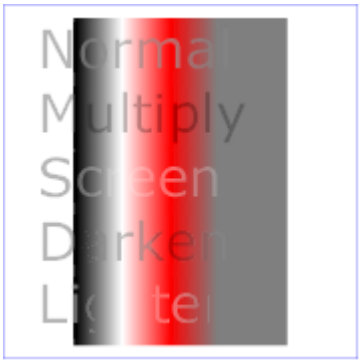
Image Blending Mode	Formula for computing result color
normal	cr = (1 - qa) * cb + ca
multiply	cr = (1-qa)*cb + (1-qb)*ca + ca*cb
screen	cr = cb + ca - ca * cb
darken	cr = Min ((1 - qa) * cb + ca, (1 - qb) * ca + cb)
lighten	cr = Max ((1 - qa) * cb + ca, (1 - qb) * ca + cb)

'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 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="5cm" viewBox="0 0 500 500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



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.

Categories: 'feColorMatrix'

Filter primitive element

Content model:

Any number of the following elements, in any order:

'animate'

'set'

Attributes:

core attributes

presentation attributes

filter primitive attributes

'class'

'style'

'in'
'type'
'values'

DOM Interfaces:
SVGFEColorMatrixElement

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. If attribute 'type' is not specified, then the effect is as if a value of **matrix** were specified.
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 identity matrix. If **type="hueRotate"**, then this attribute defaults to the value **0**, which results in the identity matrix.
Animatable: yes.

Example feColorMatrix shows examples of the four types of feColorMatrix operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="5cm" viewBox="0 0 800 500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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="0.4"/>
    </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

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.)

Categories: ‘feComponentTransfer’
Filter primitive element

Content model:
Any number of the following elements, in any order:
'feFuncA'
'feFuncB'
'feFuncG'
'feFuncR'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'

DOM Interfaces:
SVGFEComponentTransferElement

The child elements of a 'feComponentTransfer' element specify the transfer functions for the four channels:

- 'feFuncR' — transfer function for the red component of the input graphic
- 'feFuncG' — transfer function for the green component of the input graphic
- 'feFuncB' — transfer function for the blue component of the input graphic
- 'feFuncA' — transfer function for the alpha component of the input graphic

The following rules apply to the processing of the 'feComponentTransfer' element:

- If more than one transfer function element of the same kind is specified, the last occurrence is to be used.
- If any of the transfer function elements are unspecified, the 'feComponentTransfer' must be processed as if those transfer function elements were specified with their 'type' attributes set to **'identity'**.

Categories: 'feFuncR'
None

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes

transfer function element attributes

DOM Interfaces:
SVGFEFuncRElement

Categories: 'feFuncG'
None

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
transfer function element attributes

DOM Interfaces:
SVGFEFuncGElement

Categories: 'feFuncB'
None

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
transfer function element attributes

DOM Interfaces:
SVGFEFuncBElement

Categories: 'feFuncA'
None

Content model:
Any number of the following elements, in any order:
'animate'

'set'

Attributes:
core attributes
transfer function element attributes

DOM Interfaces:
SVGFEFuncAElement

The attributes below are the **transfer function element attributes**, which apply to sub-elements 'feFuncR', 'feFuncG', 'feFuncB' and 'feFuncA' that 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.

In the following, C is the initial component (e.g., 'feFuncR'), C' is the remapped component; both in the closed interval [0,1].

- For **identity**:

  ```
  C' = C
  ```

- For **table**, the function is defined by linear interpolation between values given in the attribute 'tableValues'. The table has *n+1* values (i.e., v_0 to v_n) specifying the start and end values for *n* evenly sized interpolation regions. Interpolations use the following formula:
  For a value `C < 1` find `k` such that:
  ```
  k/n <= C < (k+1)/n
  ```
  The result `C'` is given by:
  $$C' = v_k + (C - k/n)*n * (v_{k+1} - v_k)$$
  If `C = 1` then:
  $$C' = v_n.$$
- For **discrete**, the function is defined by the step function given in the 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 < 1` find `k` such that:
  ```
  k/n <= C < (k+1)/n
  ```
  The result `C'` is given by:
  $$C' = v_k$$

If `C = 1` then:

`C' = v`$_{n-1}$.

- 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.

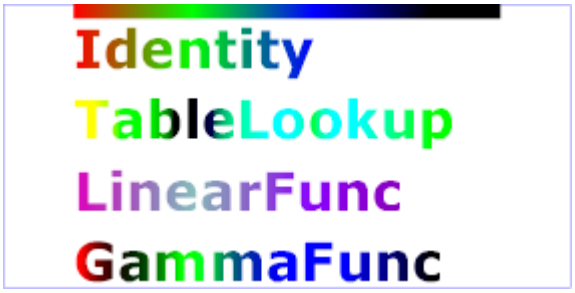
Animatable: yes.

Example feComponentTransfer shows examples of the four types of feComponentTransfer operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
```

```
  <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



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* [SVG-COMPOSITING]. 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
```

where:

- `i1` and `i2` indicate the corresponding pixel channel values of the input image, which map to in and in2 respectively
- `k1, k2, k3` and `k4` indicate the values of the attributes with the same name

For this filter primitive, the extent of the resulting image might grow as described in the section that describes the filter primitive subregion.

Categories: 'feComposite'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes

'class'
'style'
'in'
'in2'
'operator'
'k1'
'k2'
'k3'
'k4'

DOM Interfaces:
SVGFECompositeElement

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 corresponding operation as described in [PORTERDUFF]. The **arithmetic** operator is described above. If attribute 'operator' is not specified, then the effect is as if a value of **over** were specified.
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.

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.)

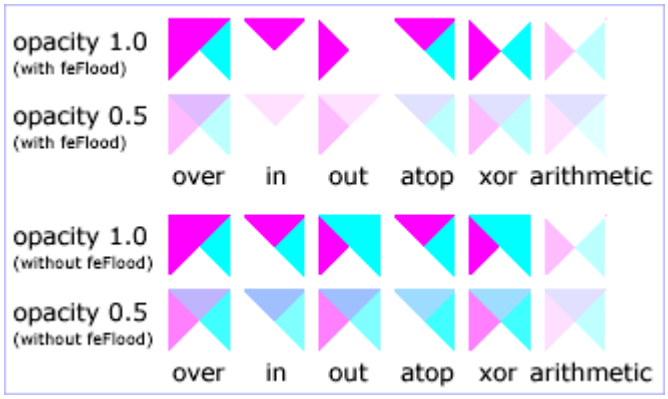
The last two rows apply the same compositing operations of reddish triangles into bluish triangles. However, the compositing result is directly blended into the canvas (the opaque white temporary surface technique is not used). In some cases, the results are different than when a temporary opaque white 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 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="330" height="195" viewBox="0 0 1100 650" version="1.1"
     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



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:

```
COLOR_X,Y = (
              SUM I=0 to [orderY-1] {
                SUM J=0 to [orderX-1] {
                  SOURCE X-targetX+J, Y-targetY+I *  kernelMatrix_orderX-J-1,  orderY-I-1
                }
              }
            ) /  divisor +  bias * ALPHA_X,Y
```

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.

'feConvolveMatrix'

Categories:

Filter primitive element

Content model:

Any number of the following elements, in any order:

'animate'
'set'

Attributes:

core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'order'
'kernelMatrix'
'divisor'
'bias'
'targetX'
'targetY'
'edgeMode'
'kernelUnitLength'
'preserveAlpha'

DOM Interfaces:
SVGFEConvolveMatrixElement

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. The bias property shifts the range of the filter. This allows representation of values that would otherwise be clamped to 0 or 1. If '**bias**' is not specified, then the effect is as if a value of **0** 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.

If attribute 'edgeMode' is not specified, then the effect is as if a value of **duplicate** 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') 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. In this case the $ALPHA_{X,Y}$ of the convolution formula for a given pixel is:

```
ALPHA_X,Y = (
                SUM I=0 to [orderY-1] {
                  SUM J=0 to [orderX-1] {
                    SOURCE X-targetX+J, Y-targetY+I *  kernelMatrix_orderX-J-1,  orderY-I-1
                  }
                }
              ) /  divisor +  bias
```

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. In this case the $ALPHA_{X,Y}$ of the convolution formula for a given pixel is:

$ALPHA_{X,Y} = SOURCE_{X,Y}$

If 'preserveAlpha' is not specified, then the effect is as if a value of **false** were specified.

Animatable: yes.

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 `Norm(Ax,Ay,Az)` function is defined as:

$$Norm(A_x,A_y,A_z) = sqrt(A_x^2+A_y^2+A_z^2)$$

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 I(x,y) is:

$$Z(x,y) = surfaceScale * I(x,y)$$

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'**.

<table>
<tr>
<td>Top/left corner:

$FACTOR_x=2/(3*dx)$
K_x =
| 0 0 0 |
| 0 -2 2 |
| 0 -1 1 |

$FACTOR_y=2/(3*dy)$
K_y =
| 0 0 0 |
| 0 -2 -1 |
| 0 2 1 |</td>
<td>Top row:

$FACTOR_x=1/(3*dx)$
K_x =
| 0 0 0 |
| -2 0 2 |
| -1 0 1 |

$FACTOR_y=1/(2*dy)$
K_y =
| 0 0 0 |
| -1 -2 -1 |
| 1 2 1 |</td>
<td>Top/right corner:

$FACTOR_x=2/(3*dx)$
K_x =
| 0 0 0 |
| -2 2 0 |
| -1 1 0 |

$FACTOR_y=2/(3*dy)$
K_y =
| 0 0 0 |
| -1 -2 0 |
| 1 2 0 |</td>
</tr>
<tr>
<td>Left column:

$FACTOR_x=1/(2*dx)$
K_x =
| 0 -1 1 |
| 0 -2 2 |
| 0 -1 1 |

$FACTOR_y=1/(3*dy)$
K_y =
| 0 -2 -1 |
| 0 0 0 |
| 0 2 1 |</td>
<td>Interior pixels:

$FACTOR_x=1/(4*dx)$
K_x =
| -1 0 1 |
| -2 0 2 |
| -1 0 1 |

$FACTOR_y=1/(4*dy)$
K_y =
| -1 -2 -1 |
| 0 0 0 |
| 1 2 1 |</td>
<td>Right column:

$FACTOR_x=1/(2*dx)$
K_x =
| -1 1 0|
| -2 2 0|
| -1 1 0|

$FACTOR_y=1/(3*dy)$
K_y =
| -1 -2 0 |
| 0 0 0 |
| 1 2 0 |</td>
</tr>
<tr>
<td>Bottom/left corner:

$FACTOR_x=2/(3*dx)$
K_x =
| 0 -1 1 |
| 0 -2 2 |
| 0 0 0 |

$FACTOR_y=2/(3*dy)$
K_y =
| 0 -2 -1 |
| 0 2 1 |
| 0 0 0 |</td>
<td>Bottom row:

$FACTOR_x=1/(3*dx)$
K_x =
| -1 0 1 |
| -2 0 2 |
| 0 0 0 |

$FACTOR_y=1/(2*dy)$
K_y =
| -1 -2 -1 |
| 1 2 1 |
| 0 0 0 |</td>
<td>Bottom/right corner:

$FACTOR_x=2/(3*dx)$
K_x =
| -1 1 0 |
| -2 2 0 |
| 0 0 0 |

$FACTOR_y=2/(3*dy)$
K_y =
| -1 -2 0 |
| 1 2 0 |
| 0 0 0 |</td>
</tr>
</table>

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 = (Lx, Ly, Lz) / Norm(Lx, Ly, Lz)
```

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:

```
Lr = Lightr*pow((-L.S),specularExponent)
Lg = Lightg*pow((-L.S),specularExponent)
Lb = Lightb*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:

```
Sx = pointsAtX - Lightx
Sy = pointsAtY - Lighty
Sz = pointsAtZ - Lightz

S = (Sx, Sy, Sz) / Norm(Sx, Sy, Sz)
```

If L.S is positive, no light is present. ($L_r = L_g = L_b = 0$). If **'limitingConeAngle'** is specified, -L.S < cos(limitingConeAngle) also indicates that no light is present.

Categories: **'feDiffuseLighting'**

Filter primitive element

Content model:

Any number of descriptive elements and exactly one light source element, in any order.

Attributes:

core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'surfaceScale'
'diffuseConstant'
'kernelUnitLength'

DOM Interfaces:

SVGFEDiffuseLightingElement

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.

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 channel 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 input image in is to remain premultiplied for this filter primitive. 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. The 'in' source image must remain in its current color space.

Categories: 'feDisplacementMap'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'in2'
'scale'
'xChannelSelector'
'yChannelSelector'

DOM Interfaces:
SVGFEDisplacementMapElement

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. If attribute 'xChannelSelector' is not specified, then the effect is as if a value of **A** were specified.
Animatable: yes.

`yChannelSelector` = "***R* | *G* | *B* | *A***"

Indicates which channel from 'in2' to use to displace the pixels in 'in' along the y-axis. If attribute 'yChannelSelector' is not specified, then the effect is as if a value of A were specified.
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.

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.

Categories: 'feFlood'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'animateColor'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'

DOM Interfaces:
SVGFEFloodElement

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> [<icccolor>] | 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:	<opacity-value> | 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:

```
G(x,y) = H(x)I(y)
```

where

$$H(x) = \exp(-x^2/(2s^2)) / \sqrt{2\pi s^2}$$

and

$$I(y) = \exp(-y^2/(2t^2)) / \sqrt{2\pi t^2}$$

with 's' being the standard deviation in the x direction and 't' being the standard deviation in the y direction, as 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.

Note: the approximation formula also applies correspondingly to 't'.

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 (e.g **FillPaint** where the fill is a solid color), this operation has no effect. If the input has infinite extent and the filter result is the input to an 'feTile', the filter is evaluated with periodic boundary conditions.

Categories: 'feGaussianBlur'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'stdDeviation'

DOM Interfaces:
SVGFEGaussianBlurElement

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 the filter input image). If 'stdDeviation' is **0** in only one of X or Y, then the effect is that the blur is only applied in the direction that has a non-zero value.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

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. The processing of the 'preserveAspectRatio' attribute on the 'feImage' element is identical to that of the 'image' 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.

Categories: 'feImage'

Filter primitive element

Content model:

Any number of the following elements, in any order:

'animate'
'animateTransform'
'set'

Attributes:

core attributes
presentation attributes
filter primitive attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'preserveAspectRatio'
'xlink:href'

DOM Interfaces:

SVGFEImageElement

Attribute definitions:

`xlink:href` = "**<iri>**"

An IRI reference to the image source.

Animatable: yes.

`preserveAspectRatio` = "**[defer] <align> [<meetOrSlice>]**"

See '**preserveAspectRatio**'.

If attribute '**preserveAspectRatio**' is not specified, then the effect is as if a value of **xMidYMid meet** were specified.

Animatable: yes.

Example feImage illustrates how images are placed relative to an object. From left to right:

- The default placement of an image. Note that the image is centered in the filter region and has the maximum size that will fit in the region consistent with preserving the aspect ratio.
- The image stretched to fit the bounding box of an object.
- The image placed using user coordinates. Note that the image is first centered in a box the size of the filter region and has the maximum size that will fit in the box consistent with preserving the aspect ratio. This box is then shifted by the given 'x' and 'y' values relative to the viewport the object is in.

```
<svg width="600" height="250" viewBox="0 0 600 250"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Example feImage - Examples of feImage use</title>
  <desc>Three examples of using feImage, the first showing the
        default rendering, the second showing the image fit
        to a box and the third showing the image
        shifted and clipped.</desc>
  <defs>
    <filter id="Default">
      <feImage xlink:href="smiley.png" />
    </filter>
    <filter id="Fitted" primitiveUnits="objectBoundingBox">
      <feImage xlink:href="smiley.png"
               x="0" y="0" width="100%" height="100%"
               preserveAspectRatio="none"/>
    </filter>
    <filter id="Shifted">
      <feImage xlink:href="smiley.png"
               x="500" y="5"/>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="598" height="248"/>
  <g>
    <rect x="50"  y="25" width="100" height="200" filter="url(#Default)"/>
    <rect x="50"  y="25" width="100" height="200" fill="none" stroke="green"/>
    <rect x="250" y="25" width="100" height="200" filter="url(#Fitted)"/>
    <rect x="250" y="25" width="100" height="200" fill="none" stroke="green"/>
    <rect x="450" y="25" width="100" height="200" filter="url(#Shifted)"/>
    <rect x="450" y="25" width="100" height="200" fill="none" stroke="green"/>
  </g>
</svg>
```

Example feImage

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.

Categories: 'feMerge'

Filter primitive element

Content model:

Any number of the following elements, in any order:

'feMergeNode'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'

DOM Interfaces:
SVGFEMergeElement

Categories: 'feMergeNode'
None

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
'in'

DOM Interfaces:
SVGFEMergeNodeElement

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 (e.g **FillPaint** where the fill is a solid color), this operation has no effect. If the input has infinite extent and the filter result is the input to an 'feTile', 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.

Categories: ‘feMorphology’
Filter primitive element

Content model:
Any number of the following elements, in any order:
‘animate’
‘set’

Attributes:
core attributes
presentation attributes
filter primitive attributes
‘class’
‘style’
‘in’
‘operator’
‘radius’

DOM Interfaces:
SVGFEMorphologyElement

Attribute definitions:

`operator` = "***erode* | *dilate***"
A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic. If attribute ‘operator’ is not specified, then the effect is as if a value of **erode** were specified.
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.

Example feMorphology shows examples of the four types of feMorphology operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg width="5cm" height="7cm" viewBox="0 0 700 500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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

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.

Categories: 'feOffset'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'dx'
'dy'

DOM Interfaces:
SVGFEOffsetElement

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.

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:

$$S_r = k_s * pow(N.H, specularExponent) * L_r$$
$$S_g = k_s * pow(N.H, specularExponent) * L_g$$
$$S_b = k_s * pow(N.H, specularExponent) * L_b$$
$$S_a = max(S_r, S_g, S_b)$$

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.

Categories: 'feSpecularLighting'
Filter primitive element

Content model:
Any number of descriptive elements and exactly one light source element, in any order.

Attributes:

core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'
'surfaceScale'
'specularConstant'
'specularExponent'
'kernelUnitLength'

DOM Interfaces:

SVGFESpecularLightingElement

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.

`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.

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.

Categories: 'feTile'
Filter primitive element

Content model:
Any number of the following elements, in any order:
'animate'
'set'

Attributes:
core attributes
presentation attributes
filter primitive attributes
'class'
'style'
'in'

DOM Interfaces:
SVGFETileElement

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;
}
```

Categories: ‘feTurbulence’
Filter primitive element

Content model:
Any number of the following elements, in any order:
‘animate’
‘set’

Attributes:
core attributes
presentation attributes
filter primitive attributes
‘class’
‘style’
‘baseFrequency’
‘numOctaves’
‘seed’
‘stitchTiles’
‘type’

DOM Interfaces:
SVGFETurbulenceElement

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. When the seed number is handed over to the algorithm above it must first be truncated, i.e. rounded to the closest integer value to-

wards zero.
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 sub-region) 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.
If attribute 'stitchTiles' is not specified, then the effect is as if a value of **noStitch** were specified.

Animatable: yes.

`type` = "***fractalNoise | turbulence***"

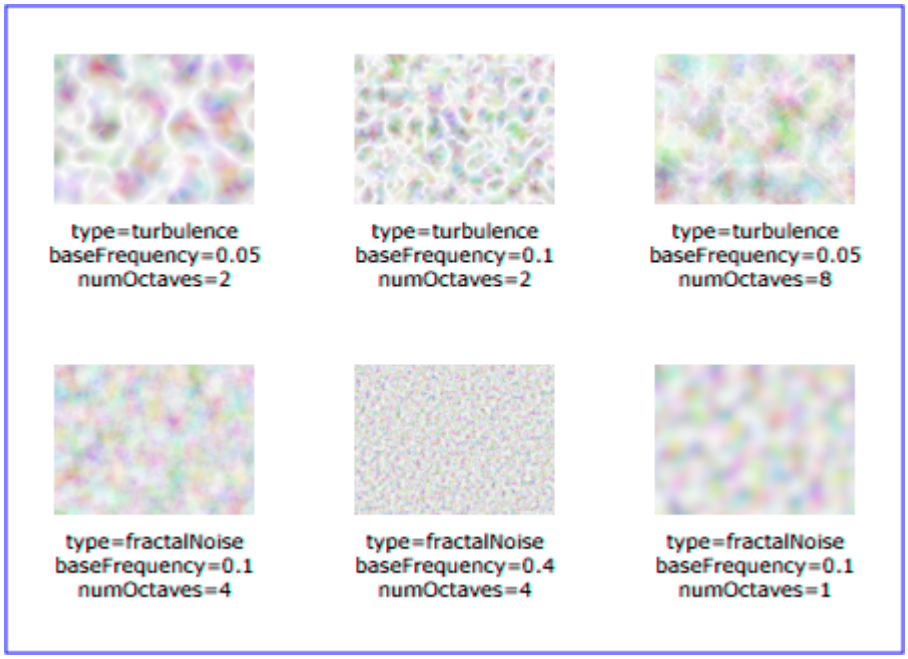
Indicates whether the filter primitive should perform a noise or turbulence function. If attribute 'type' is not specified, then the effect is as if a value of **turbulence** were specified.
Animatable: yes.

Example feTurbulence shows the effects of various parameter settings for feTurbulence.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="450px" height="325px" viewBox="0 0 450 325"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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



15.25 DOM interfaces

15.25.1 Interface SVGFilterElement

The SVGFilterElement interface corresponds to the 'filter' element.

```
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) raises(DOMException);
};
```

Attributes:

- **filterUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'filterUnits' on the given 'filter' element. Takes one of the constants defined in SVGUnitTypes.

- **primitiveUnits** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'primitiveUnits' on the given 'filter' element. Takes one of the constants defined in SVGUnitTypes.

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'filter' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'filter' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'filter' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'filter' element.

- **filterResX** (readonly SVGAnimatedInteger)

  Corresponds to attribute 'filterRes' on the given 'filter' element. Contains the X component of attribute 'filterRes'.

- **filterResY** (readonly SVGAnimatedInteger)

  Corresponds to attribute 'filterRes' on the given 'filter' element. Contains the Y component (possibly computed automatically) of attribute 'filterRes'.

Operations:

- void **setFilterRes**(in unsigned long *filterResX*, in unsigned long *filterResY*)

  Sets the values for attribute 'filterRes'.

  Parameters

  - unsigned long *filterResX*
    The X component of attribute 'filterRes'.
  - unsigned long *filterResY*
    The Y component of attribute 'filterRes'.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

15.25.2 Interface SVGFilterPrimitiveStandardAttributes

This interface defines the set of DOM attributes that are common across the filter primitive interfaces.

```
interface SVGFilterPrimitiveStandardAttributes : SVGStylable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};
```

Attributes:

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given element.

- **result** (readonly SVGAnimatedString)

  Corresponds to attribute 'result' on the given element.

15.25.3 Interface SVGFEBlendElement

The SVGFEBlendElement interface corresponds to the 'feBlend' element.

```
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;
};
```

Constants in group "Blend Mode Types":

- **SVG_FEBLEND_MODE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'normal'**.

- **SVG_FEBLEND_MODE_MULTIPLY** (unsigned short)

  Corresponds to value **'multiply'**.

- **SVG_FEBLEND_MODE_SCREEN** (unsigned short)

  Corresponds to value **'screen'**.

- **SVG_FEBLEND_MODE_DARKEN** (unsigned short)

  Corresponds to value **'darken'**.

- **SVG_FEBLEND_MODE_LIGHTEN** (unsigned short)

  Corresponds to value **'lighten'**.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute **'in'** on the given **'feBlend'** element.

- **in2** (readonly SVGAnimatedString)

  Corresponds to attribute **'in2'** on the given **'feBlend'** element.

- **mode** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'mode'** on the given **'feBlend'** element. Takes one of the SVG_FEBLEND_MODE_* constants defined on this interface.

15.25.4 Interface SVGFEColorMatrixElement

The SVGFEColorMatrixElement interface corresponds to the **'feColorMatrix'** element.

```
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;
};
```

Constants in group "Color Matrix Types":

- **SVG_FECOLORMATRIX_TYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'matrix'**.

- **SVG_FECOLORMATRIX_TYPE_SATURATE** (unsigned short)

  Corresponds to value **'saturate'**.

- **SVG_FECOLORMATRIX_TYPE_HUEROTATE** (unsigned short)

  Corresponds to value **'hueRotate'**.

- **SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA** (unsigned short)

  Corresponds to value **'luminanceToAlpha'**.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feColorMatrix' element.

- **type** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'type' on the given 'feColorMatrix' element. Takes one of the SVG_FECOLORMATRIX_TYPE_* constants defined on this interface.

- **values** (readonly SVGAnimatedNumberList)

  Corresponds to attribute 'values' on the given 'feColorMatrix' element.

15.25.5 Interface SVGFEComponentTransferElement

The SVGFEComponentTransferElement interface corresponds to the 'feComponentTransfer' element.

```
interface SVGFEComponentTransferElement : SVGElement,
                                          SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feComponentTransfer' element.

15.25.6 Interface SVGComponentTransferFunctionElement

This interface defines a base interface used by the component transfer function interfaces.

```
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;
};
```

Constants in group "Component Transfer Types":

- **SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'identity'**.

- **SVG_FECOMPONENTTRANSFER_TYPE_TABLE** (unsigned short)

  Corresponds to value **'table'**.

- **SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE** (unsigned short)

  Corresponds to value **'discrete'**.

- **SVG_FECOMPONENTTRANSFER_TYPE_LINEAR** (unsigned short)

  Corresponds to value **'linear'**.

- **SVG_FECOMPONENTTRANSFER_TYPE_GAMMA** (unsigned short)

  Corresponds to value **'gamma'**.

Attributes:

- **type** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'type'** on the given element. Takes one of the SVG_FECOMPONENTTRANSFER_TYPE_* constants defined on this interface.

- **tableValues** (readonly SVGAnimatedNumberList)

  Corresponds to attribute **'tableValues'** on the given element.

- **slope** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'slope'** on the given element.

- **intercept** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'intercept'** on the given element.

- **amplitude** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'amplitude'** on the given element.

- **exponent** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'exponent'** on the given element.

- **offset** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'offset'** on the given element.

15.25.7 Interface SVGFEFuncRElement

The SVGFEFuncRElement interface corresponds to the 'feFuncR' element.

```
interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {
};
```

15.25.8 Interface SVGFEFuncGElement

The SVGFEFuncRElement interface corresponds to the 'feFuncG' element.

```
interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {
};
```

15.25.9 Interface SVGFEFuncBElement

The SVGFEFuncBElement interface corresponds to the 'feFuncB' element.

```
interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {
};
```

15.25.10 Interface SVGFEFuncAElement

The SVGFEFuncAElement interface corresponds to the 'feFuncA' element.

```
interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {
};
```

15.25.11 Interface SVGFECompositeElement

The SVGFECompositeElement interface corresponds to the 'feComposite' element.

```
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;
};
```

Constants in group "Composite Operators":

- **SVG_FECOMPOSITE_OPERATOR_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'over'**.

- **SVG_FECOMPOSITE_OPERATOR_IN** (unsigned short)

  Corresponds to value **'in'**.

- **SVG_FECOMPOSITE_OPERATOR_OUT** (unsigned short)

  Corresponds to value **'out'**.

- **SVG_FECOMPOSITE_OPERATOR_ATOP** (unsigned short)

  Corresponds to value **'atop'**.

- **SVG_FECOMPOSITE_OPERATOR_XOR** (unsigned short)

  Corresponds to value **'xor'**.

- **SVG_FECOMPOSITE_OPERATOR_ARITHMETIC** (unsigned short)

  Corresponds to value **'arithmetic'**.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute **'in'** on the given **'feComposite'** element.

- **in2** (readonly SVGAnimatedString)

  Corresponds to attribute **'in2'** on the given **'feComposite'** element.

- **operator** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'operator' on the given 'feComposite' element. Takes one of the SVG_FECOMPOSITE_OPERATOR_* constants defined on this interface.

- **k1** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'k1' on the given 'feComposite' element.

- **k2** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'k2' on the given 'feComposite' element.

- **k3** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'k3' on the given 'feComposite' element.

- **k4** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'k4' on the given 'feComposite' element.

15.25.12 Interface SVGFEConvolveMatrixElement

The SVGFEConvolveMatrixElement interface corresponds to the 'feConvolveMatrix' element.

```
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 SVGAnimatedString in1;
  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 SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
  readonly attribute SVGAnimatedBoolean preserveAlpha;
};
```

Constants in group "Edge Mode Values":

- **SVG_EDGEMODE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'duplicate'**.

- **SVG_EDGEMODE_WRAP** (unsigned short)

  Corresponds to value **'wrap'**.

- **SVG_EDGEMODE_NONE** (unsigned short)

  Corresponds to value **'none'**.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute **'in'** on the given **'feConvolveMatrix'** element.

- **orderX** (readonly SVGAnimatedInteger)

  Corresponds to attribute **'order'** on the given **'feConvolveMatrix'** element.

- **orderY** (readonly SVGAnimatedInteger)

  Corresponds to attribute **'order'** on the given **'feConvolveMatrix'** element.

- **kernelMatrix** (readonly SVGAnimatedNumberList)

  Corresponds to attribute **'kernelMatrix'** on the given **'feConvolveMatrix'** element.

- **divisor** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'divisor'** on the given **'feConvolveMatrix'** element.

- **bias** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'bias'** on the given **'feConvolveMatrix'** element.

- **targetX** (readonly SVGAnimatedInteger)

  Corresponds to attribute 'targetX' on the given 'feConvolveMatrix' element.

- **targetY** (readonly SVGAnimatedInteger)

  Corresponds to attribute 'targetY' on the given 'feConvolveMatrix' element.

- **edgeMode** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'edgeMode' on the given 'feConvolveMatrix' element. Takes one of the SVG_EDGEMODE_* constants defined on this interface.

- **kernelUnitLengthX** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feConvolveMatrix' element.

- **kernelUnitLengthY** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feConvolveMatrix' element.

- **preserveAlpha** (readonly SVGAnimatedBoolean)

  Corresponds to attribute 'preserveAlpha' on the given 'feConvolveMatrix' element.

15.25.13 Interface SVGFEDiffuseLightingElement

The SVGFEDiffuseLightingElement interface corresponds to the 'feDiffuseLighting' element.

```
interface SVGFEDiffuseLightingElement : SVGElement,
                                        SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feDiffuseLighting' element.

- **surfaceScale** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'surfaceScale' on the given 'feDiffuseLighting' element.

- **diffuseConstant** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'diffuseConstant' on the given 'feDiffuseLighting' element.

- **kernelUnitLengthX** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feDiffuseLighting' element.

- **kernelUnitLengthY** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feDiffuseLighting' element.

15.25.14 Interface SVGFEDistantLightElement

The SVGFEDistantLightElement interface corresponds to the 'feDistantLight' element.

```
interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};
```

Attributes:

- **azimuth** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'azimuth' on the given 'feDistantLight' element.

- **elevation** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'elevation' on the given 'feDistantLight' element.

15.25.15 Interface SVGFEPointLightElement

The SVGFEPointLightElement interface corresponds to the 'fePointLight' element.

```
interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};
```

Attributes:

- **x** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'x' on the given 'fePointLight' element.

- **y** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'y' on the given 'fePointLight' element.

- **z** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'z' on the given 'fePointLight' element.

15.25.16 Interface SVGFESpotLightElement

The SVGFESpotLightElement interface corresponds to the 'feSpotLight' element.

```
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:

- **x** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'x' on the given 'feSpotLight' element.

- **y** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'y' on the given 'feSpotLight' element.

- **z** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'z' on the given 'feSpotLight' element.

- **pointsAtX** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'pointsAtX' on the given 'feSpotLight' element.

- **pointsAtY** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'pointsAtY' on the given 'feSpotLight' element.

- **pointsAtZ** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'pointsAtZ' on the given 'feSpotLight' element.

- **specularExponent** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'specularExponent' on the given 'feSpotLight' element.

- **limitingConeAngle** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'limitingConeAngle' on the given 'feSpotLight' element.

15.25.17 Interface SVGFEDisplacementMapElement

The SVGFEDisplacementMapElement interface corresponds to the 'feDisplacementMap' element.

```
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;
};
```

Constants in group "Channel Selectors":

- **SVG_CHANNEL_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'R'**.

- **SVG_CHANNEL_G** (unsigned short)

  Corresponds to value **'G'**.

- **SVG_CHANNEL_B** (unsigned short)

  Corresponds to value **'B'**.

- **SVG_CHANNEL_A** (unsigned short)

  Corresponds to value 'A'.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feDisplacementMap' element.

- **in2** (readonly SVGAnimatedString)

  Corresponds to attribute 'in2' on the given 'feDisplacementMap' element.

- **scale** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'scale' on the given 'feDisplacementMap' element.

- **xChannelSelector** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'xChannelSelector' on the given 'feDisplacementMap' element. Takes one of the SVG_CHANNEL_* constants defined on this interface.

- **yChannelSelector** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute 'yChannelSelector' on the given 'feDisplacementMap' element. Takes one of the SVG_CHANNEL_* constants defined on this interface.

15.25.18 Interface SVGFEFloodElement

The SVGFEFloodElement interface corresponds to the 'feFlood' element.

```
interface SVGFEFloodElement : SVGElement,
                              SVGFilterPrimitiveStandardAttributes {
};
```

15.25.19 Interface SVGFEGaussianBlurElement

The SVGFEGaussianBlurElement interface corresponds to the 'feGaussianBlur' element.

```
interface SVGFEGaussianBlurElement : SVGElement,
                                     SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;
```

```
  void setStdDeviation(in float stdDeviationX, in float stdDeviationY) raises(DOMException);
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feGaussianBlur' element.

- **stdDeviationX** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'stdDeviation' on the given 'feGaussianBlur' element. Contains the X component of attribute 'stdDeviation'.

- **stdDeviationY** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'stdDeviation' on the given 'feGaussianBlur' element. Contains the Y component (possibly computed automatically) of attribute 'stdDeviation'.

Operations:

- void **setStdDeviation**(in float *stdDeviationX*, in float *stdDeviationY*)

  Sets the values for attribute 'stdDeviation'.

  Parameters

  - float *stdDeviationX*
    The X component of attribute 'stdDeviation'.
  - float *stdDeviationY*
    The Y component of attribute 'stdDeviation'.

  Exceptions

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

15.25.20 Interface SVGFEImageElement

The SVGFEImageElement interface corresponds to the 'feImage' element.

```
interface SVGFEImageElement : SVGElement,
                              SVGURIReference,
```

```
                              SVGLangSpace,
                              SVGExternalResourcesRequired,
                              SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

Attributes:

- **preserveAspectRatio** (readonly SVGAnimatedPreserveAspectRatio)

  Corresponds to attribute 'preserveAspectRatio' on the given 'feImage' element.

15.25.21 Interface SVGFEMergeElement

The SVGFEMergeElement interface corresponds to the 'feMerge' element.

```
interface SVGFEMergeElement : SVGElement,
                              SVGFilterPrimitiveStandardAttributes {
};
```

15.25.22 Interface SVGFEMergeNodeElement

The SVGFEMergeNodeElement interface corresponds to the 'feMergeNode' element.

```
interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feMergeNode' element.

15.25.23 Interface SVGFEMorphologyElement

The SVGFEMorphologyElement interface corresponds to the 'feMorphology' element.

```
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 SVGAnimatedNumber radiusX;
  readonly attribute SVGAnimatedNumber radiusY;
};
```

Constants in group "Morphology Operators":

- **SVG_MORPHOLOGY_OPERATOR_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'erode'**.

- **SVG_MORPHOLOGY_OPERATOR_DILATE** (unsigned short)

  Corresponds to value **'dilate'**.

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute **'in'** on the given **'feMorphology'** element.

- **operator** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'operator'** on the given **'feMorphology'** element. Takes one of the SVG_MORPHOLOGY_OPERATOR_* constants defined on this interface.

- **radiusX** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'radius'** on the given **'feMorphology'** element.

- **radiusY** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'radius'** on the given **'feMorphology'** element.

15.25.24 Interface SVGFEOffsetElement

The SVGFEOffsetElement interface corresponds to the **'feOffset'** element.

```
interface SVGFEOffsetElement : SVGElement,
                               SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feOffset' element.

- **dx** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'dx' on the given 'feOffset' element.

- **dy** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'dy' on the given 'feOffset' element.

15.25.25 Interface SVGFESpecularLightingElement

The SVGFESpecularLightingElement interface corresponds to the 'feSpecularLighting' element.

```
interface SVGFESpecularLightingElement : SVGElement,
                                         SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feSpecularLighting' element.

- **surfaceScale** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'surfaceScale' on the given 'feSpecularLighting' element.

- **specularConstant** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'specularConstant' on the given 'feSpecularLighting' element.

- **specularExponent** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'specularExponent' on the given 'feSpecularLighting' element.

- **kernelUnitLengthX** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feSpecularLighting' element.

- **kernelUnitLengthY** (readonly SVGAnimatedNumber)

  Corresponds to attribute 'kernelUnitLength' on the given 'feSpecularLighting' element.

15.25.26 Interface SVGFETileElement

The SVGFETileElement interface corresponds to the 'feTile' element.

```
interface SVGFETileElement : SVGElement,
                             SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes:

- **in1** (readonly SVGAnimatedString)

  Corresponds to attribute 'in' on the given 'feTile' element.

15.25.27 Interface SVGFETurbulenceElement

The SVGFETurbulenceElement interface corresponds to the 'feTurbulence' element.

```
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;
};
```

Constants in group "Turbulence Types":

- **SVG_TURBULENCE_TYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'fractalNoise'**.

- **SVG_TURBULENCE_TYPE_TURBULENCE** (unsigned short)

  Corresponds to value **'turbulence'**.

Constants in group "Stitch Options":

- **SVG_STITCHTYPE_UNKNOWN** (unsigned short)

  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** (unsigned short)

  Corresponds to value **'stitch'**.

- **SVG_STITCHTYPE_NOSTITCH** (unsigned short)

  Corresponds to value **'noStitch'**.

Attributes:

- **baseFrequencyX** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'baseFrequency'** on the given **'feTurbulence'** element. Contains the X component of the **'baseFrequency'** attribute.

- **baseFrequencyY** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'baseFrequency'** on the given **'feTurbulence'** element. Contains the Y component of the (possibly computed automatically) **'baseFrequency'** attribute.

- **numOctaves** (readonly SVGAnimatedInteger)

  Corresponds to attribute **'numOctaves'** on the given **'feTurbulence'** element.

- **seed** (readonly SVGAnimatedNumber)

  Corresponds to attribute **'seed'** on the given **'feTurbulence'** element.

- **stitchTiles** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'stitchTiles'** on the given **'feTurbulence'** element. Takes one of the SVG_STITCHTYPE_* constants defined on this interface.

- **type** (readonly SVGAnimatedEnumeration)

  Corresponds to attribute **'type'** on the given **'feTurbulence'** element. Takes one of the SVG_TURBULENCE_TYPE_* constants defined on this interface.

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 DOM 2 event listeners ([DOM2EVENTS], section 1.3) 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 second column is the name to use when defining DOM 2 event listeners ([DOM2EVENTS], section 1.3). The *Event attribute name* in the fourth column contains the corresponding name of the event attributes that can be attached to elements in the SVG language.

Requirements in the table on whether an event of a given type bubbles or is cancelable apply only to events that are created and dispatched by the user agent. Events of those types created from script using the createEvent method on the DocumentEvent interface can be made to bubble or be cancelable with the initEvent method.

Event name and 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, through 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	DOMActivate	UIEvent	onactivate

hyperactivation (for instance a double click or Shift Enter).			
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 repeats with the `detail` attribute incrementing with each repetition.	(same)	MouseEvent	onclick
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. SVGLoad events do not bubble and are not cancelable.	(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. SVGUnload events do not bubble and are not cancelable.	(same)	none	onunload
SVGAbort The abort event occurs when page loading is stopped before an element has been allowed to load completely. SVGAbort events bubble but are not cancelable.	(same)	none	onabort
SVGError The error event occurs when an element does not load properly or when an error occurs during script execution. SVGError events bubble but are not cancelable.	(same)	none	onerror
SVGResize Occurs when a document view is being resized. This event is only applicable to outermost svg elements and is dispatched after the resize operation has taken place. The target of the event is the **'svg'** element. SVGResize events bubble but are not cancelable.	(same)	none	onresize
SVGScroll Occurs when a document view is being shifted along the X or Y or both axis, either through a direct user interaction or any change on the currentTranslate property available on SVGSVGElement interface. This event is only applicable to outermost svg elements and is dispatched after the shift modification has taken place. The target of the event is the **'svg'** element. SVGScroll events bubble but are not cancelable.	(same)	none	onscroll
SVGZoom	none	none	onzoom

Occurs when the zoom level of a document view is being changed, either through a direct user interaction or any change to the currentScale property available on SVGSVGElement interface. This event is only applicable to outermost svg elements and is dispatched after the zoom level modification has taken place. The target of the event is the **'svg'** element. SVGZoom events bubble but are not cancelable.			
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 DOM 2 Key events ([DOM2EVENTS], section 1.6.3), 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.

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.

Event listener attributes can be specified on some elements to listen to a given event. The script in such attributes is run only in response to "bubbling" and "at target" phase events dispatched to the element.

Likewise, event-value timing specifiers used in animation element 'begin' and 'end' attributes are resolved to concrete times only in response to "bubbling" and "at target" phase events dispatched to the relevant element.

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.

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.

If a target element for the pointer event exists, then the event is dispatched to that element according to the normal event flow ([DOM2EVENTS], section 1.2). Note, however, that if the target element is in a 'use' element shadow tree, that the event flow will include SVGElementInstance objects. See The 'use' element for details.

If a target element for the pointer event does not exist, then the event is ignored.

16.5 Hit-testing and processing order for user interface events

There are two distinct aspects of pointer-device interaction with an element or area:

1. hit-testing, to determine if a pointer event (such as a mouse movement or mouse click) occurred within the interaction area of an element, and the subsequent DOM event flow;
2. functional processing of actions associated with any relevant element.

16.5.1 Hit-testing

Determining whether a pointer event results in a positive hit-test depends upon the position of the pointer, the size and shape of the graphics element, and the computed value of the '**pointer-events**' property on the element. The definition of the '**pointer-events**' property below describes the exact region that is sensitive to pointer events for a given type of graphics element.

Note that the 'svg' element is not a graphics element, and in a Conforming SVG Stand-Alone File a rootmost 'svg' element will never be the target of pointer events, though events can bubble to this element. If a pointer event

does not result in a positive hit-test on a graphics element, then it should evoke any user-agent-specific window behavior, such as a presenting a context menu or controls to allow zooming and panning of an SVG document fragment.

This specification does not define the behavior of pointer events on the rootmost 'svg' element for SVG images which are embedded by reference or inclusion within another document, e.g., whether the rootmost 'svg' element embedded in an HTML document intercepts mouse click events; future specifications may define this behavior, but for the purpose of this specification, the behavior is implementation-specific.

16.5.2 Event processing

An element which is the target of a user interface event may have particular interaction behaviors, depending upon the type of element and whether it has explicit associated interactions, such as scripted event listeners, CSS pseudo-classes matches, or declarative animations with event-based timing. The algorithm and order for processing user interface events for a given target element, after dispatching the DOM event, is as follows:

1. If an event handler registered on this element invokes the `preventDefault()` DOM method, then no further processing for this element is performed, and the event follows the event flow processing as described in DOM Level 2 Events [DOM2EVENTS] (or its successor);
2. If the element has an associated title or description, such as a 'title' element or an 'xlink:title' attribute, and the user agent supports the display of such information (e.g. via a tooltip or status-bar message), that information should be displayed, as appropriate to the type of pointer event;
3. If the element matches any relevant dynamic pseudo-class selectors appropriate to the type of pointer event, such as `:hover`, `:active`, or `:focus` as described in [CSS2], section 5.11, then the relevant class properties are applied;
4. If the element and the event type are associated with the activation or cancelation of declarative animation though the use of event-value timing specifiers, any corresponding instance times must be resolved, and any conseqential actions of this instance time resolution (such as immediately starting or stopping the animation) must be performed;
5. If the element is a hyperlink (e.g., it is a descendant element of an 'a' element), and the pointer event is of a type that activates that hyperlink (e.g. via a mouse click), and if the hyperlink traversal changes the context of the content (e.g. opens a different document, or moves the pointer away from this element by moving to another part of the same document), then no further processing for this element is performed;
6. If the element is a text content element, and the event type is one which the user agent recognizes as part of a text-selection operation (e.g., a mouse click and drag, or a double-click), then the text selection algorithm is performed;
7. If the event type is one which the user agent associates with the evocation of special user-interface controls (e.g., a right-click or command-click evoking a context menu), the user agent should evoke such user-agent-specific behavior, such as presenting a context menu or controls 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 conditions 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.

The effects of masking and clipping differ with respect to pointer-events. A clip path is a geometric boundary, and a given point is clearly either inside or outside that boundary; thus, pointer events must be captured normally over the rendered areas of a clipped element, but must not be captured over the clipped areas, as described in the definition of clipping paths. By contrast, a mask is not a binary transition, but a pixel operation, and different behavior for fully transparent and almost-but-not-fully-transparent may be confusingly arbitrary; as a consequence, for elements with a mask applied, pointer-events must still be captured even in areas where the mask goes to zero opacity. If an author wishes to achieve an effect where the transparent parts of a mask allow pointer-events to pass to an element below, a combination of masking and clipping may be used.

The '**filter**' property has no effect on pointer-events processing, and must in this context be treated as if the '**filter**' wasn't specified.

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], section 5.11)
- 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 has an actual 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 affect 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 affect 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 affect 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 has an actual value other than **none** or it is over the perimeter (i.e., stroke) of the element and the '**stroke**' property has an actual 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 affect 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 affect 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 affect event processing.

none

The given element does not receive pointer events.

For text elements, hit-testing 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 affect 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 affect 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 affect event processing.
- The value **none** indicates that the given text does not receive pointer events.

For raster images, hit-testing 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 basis (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 affect 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 affect 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 affect 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 <funciri> 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:	[[<funciri> ,]* [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.

<funciri>
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.svg#curs"), text; }
```

The '**cursor**' property for SVG is identical to the '**cursor**' property defined in CSS2 ([CSS2], section 18.1), with the additional requirement that SVG user agents must support cursors defined by the SVG '**cursor**' element. This gives a single, cross-platform, interoperable cursor format, with PNG as the raster component.

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 image [PNG] 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.

'**cursor**'

Categories:
None

Content model:
Any number of the following elements, in any order:
descriptive elements

Attributes:
core attributes
conditional processing attributes
xlink attributes
'externalResourcesRequired'
'x'
'y'
'xlink:href'

DOM Interfaces:
SVGCursorElement

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` = "**<funciri>**"
A Functional IRI reference to the file or element which provides the image of the cursor.
Animatable: yes.

SVG user agents are required to support PNG format images as targets of the 'xlink:href' attribute.

16.9 DOM interfaces

16.9.1 Interface SVGCursorElement

The SVGCursorElement interface corresponds to the 'cursor' element.

```
interface SVGCursorElement : SVGElement,
                             SVGURIReference,
                             SVGTests,
                             SVGExternalResourcesRequired {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
};
```

Attributes:

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'cursor' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'cursor' element.

17 Linking

Contents

17.1 References

17.1.1 Overview

On the Internet, resources are identified using IRIs (Internationalized Resource Identifiers). For example, an SVG file called someDrawing.svg located at http://example.com might have the following IRI:

```
http://example.com/someDrawing.svg
```

An IRI can also address a particular element within an XML document by including an IRI fragment identifier as part of the IRI. An IRI which includes an IRI fragment identifier consists of an optional base IRI, followed by a "#" character, followed by the IRI fragment identifier. For example, the following IRI can be used to specify the element whose ID is "Lamppost" within file someDrawing.svg:

```
http://example.com/someDrawing.svg#Lamppost
```

17.1.2 IRIs and URIs

Internationalized Resource Identifiers (IRIs) are a more generalized complement to Uniform Resource Identifiers (URIs). An IRI is a sequence of characters from the Universal Character Set [UNICODE]. A URI is constructed from a much more restricted set of characters. All URIs are already conformant IRIs. A mapping from IRIs to

URIs is defined by the IRI specification, which means that IRIs can be used instead of URIs in XML documents, to identify resources. IRIs can be converted to URIs for resolution on a network, if the protocol does not support IRIs directly.

Previous versions of SVG, following XLink, defined an IRI reference type as a URI *or as a sequence of characters which must result in an IRI after a particular escaping procedure was applied.* The escaping procedure was repeated in the XLink 1.0 specification [XLINK], and in the W3C XML Schema Part 2: Datatypes specification [SCHEMA2]. This copying introduced the possibility of error and divergence, but was done because the IRI specification was not yet standardized.

In this specification, the correct term IRI is used for this "URI or sequence of characters plus an algorithm" and the escaping method, which turns IRIs into URIs, is defined by reference to the IRI specification [RFC3987], which has since become an IETF Proposed Standard. Other W3C specifications are expected to be revised over time to remove these duplicate descriptions of the escaping procedure and to refer to IRI directly.

17.1.3 Syntactic forms: IRI and FuncIRI

IRIs are used in the 'xlink:href' attribute. Some attributes allow both IRIs and text strings as content. To disambiguate a text string from a relative IRI, the functional notation <FuncIRI> is used. This is simply an IRI delimited with a functional notation. **Note:** For historical reasons, the delimiters are "url(" and ")", for compatibility with the CSS specifications. The FuncIRI form is used in presentation attributes .

SVG makes extensive use of IRI references, both absolute and relative, 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 xml:id="MyGradient">...</linearGradient>
```

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

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

SVG supports two types of IRI references:

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

17.1.4 Processing of IRI references

The following rules apply to the processing of IRI references:

- IRI references to nodes that do not exist shall be treated as invalid references.
- IRI references to elements which are inappropriate targets for the given reference shall be treated as invalid references (see list below for appropriate targets). 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'.

- IRI references that directly or indirectly reference themselves are treated as invalid circular references.

The following list describes the elements and properties that allow IRI references and the valid target types for those references:

- the 'a' element can reference any local or non-local resource
- the 'altGlyph' element must reference either an 'altGlyphDef' element or a 'glyph' element
- the 'animate' element (see Identifying the target element for an animation for reference rules)
- the 'animateColor' element (see Identifying the target element for an animation for reference rules)
- the 'animateMotion' element (see Identifying the target element for an animation for reference rules)
- the 'animateTransform' element (see Identifying the target element for an animation for reference rules)
- the '**clip-path**' property must reference a 'clipPath' element
- the 'color-profile' element must reference an ICC profile resource
- the '**color-profile**' property must reference an ICC profile resource or a 'color-profile' element
- the 'src' descriptor on an @color-profile definition must reference an ICC profile resource or a 'color-profile' element
- the 'cursor' element must reference a resource that can provide an image for the cursor graphic
- the '**cursor**' property must reference a resource that can provide an image for the cursor graphic
- the 'feImage' element must reference any local or non-local resource
- the '**fill**' property (see Specifying paint for reference rules)
- the 'filter' element must reference a 'filter' element
- the '**filter**' property must reference a 'filter' element
- the 'image' element must reference any local or non-local resource
- the 'linearGradient' element must reference a 'linearGradient' or 'radialGradient' element
- the '**marker**', '**marker-start**', '**marker-mid**' and '**marker-end**' properties must reference a 'marker' element.
- the '**mask**' property must reference a 'mask' element
- the 'pattern' element must reference a 'pattern' element
- the 'radialGradient' element must reference a 'linearGradient' or 'radialGradient' element
- the 'script' element must reference an external resource that provides the script content
- the '**stroke**' property (see Specifying paint for reference rules)
- the 'textPath' element must reference a 'path' element
- the 'tref' element can reference any SVG element
- the 'set' element (see Identifying the target element for an animation for reference rules)
- the 'use' element can reference any local or non-local resource

The following rules apply to the processing of invalid IRI references:

- An invalid local IRI reference (i.e., an invalid references to a node within the current document) represents an error (see Error processing), apart from the 'xlink:href' attribute on the 'a' element and the properties that allow for backup values in the case where the IRI reference is invalid (see '**fill**' and '**stroke**').
- An invalid circular IRI reference represents an error (see Error processing).
- When attribute 'externalResourcesRequired' has been set to '**true**' on the referencing element or one of its an-

cestors, then an unresolved external IRI reference (i.e., a resource that cannot be located) represents an error (see Error processing).

17.1.5 IRI reference attributes

IRI references are normally specified with an 'href' attribute in the XLink [XLink] namespace. For example, if the prefix of 'xlink' is used for attributes in the XLink namespace, then the attribute is be specified as 'xlink:href'. The value of this attribute forms a reference for the desired resource (or secondary resource, if there is a fragment identifier).

The value of the 'href' attribute must be an Internationalized Resource Identifier.

If the protocol, such as HTTP, does not support IRIs directly, the IRI is converted to a URI by the SVG implementation, as described in section 3.1 of the IRI specification [RFC3987.

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

If the IRI reference is relative, its absolute version must be computed by the method described in XML Base before use [XML-BASE].

`xlink:type` = **"simple"**

Identifies the type of XLink being used. In SVG 1.1, only simple links are available. Links are simple links by default, so the attribute **xlink:type="simple"** is optional and may be omitted on simple links. Refer to the XML Linking Language (XLink) [XLINK].

Animatable: no.

`xlink:role` = "**<IRI>**"

An optional IRI reference that identifies some resource that describes the intended property. The value must be an IRI reference as defined in [RFC3987], except that if the IRI scheme used is allowed to have absolute and relative forms, the IRI portion must be absolute. When no value is supplied, no particular role value shall be inferred. Refer to the XML Linking Language (XLink) [XLINK].

Animatable: no.

`xlink:arcrole` = "**<IRI>**"

An optional IRI reference that identifies some resource that describes the intended property. The value must be an IRI reference as defined in [RFC3987], except that if the IRI scheme used is allowed to have absolute and relative forms, the IRI portion must be absolute. When no value is supplied, no particular role value shall be inferred. The arcrole attribute corresponds to the [RDF-PRIMER] 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` = "**<anything>**"

The title attribute shall be 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 shall contain a string that describes the resource. In general it is preferable to use a 'title' child element rather than a 'title' attribute. 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` = **'new' | 'replace' | 'embed' | 'other' | 'none'**

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors. In case of a conflict, the target attribute has priority, since it can express a wider range of values. Refer to the XML Linking Language (XLink) [XLINK].

Animatable: no.

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

This attribute is provided for backwards compatibility with SVG 1.1. It provides documentation to XLink-aware processors. Refer to the XML Linking Language (XLink) [XLINK].

Animatable: no.

In all cases, for compliance with either the "Namespaces in XML 1.0" or the "Namespaces in XML 1.1" Recommendation [XML-NS10][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 '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>
```

17.2 Links out of SVG content: the 'a' element

SVG provides an 'a' element, to indicate links (also known as *hyperlinks* or *Web links*). The 'a' element may contain any element that its parent may contain, except itself.

SVG uses XLink ([XLink]) for all link definitions. SVG 1.1 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).

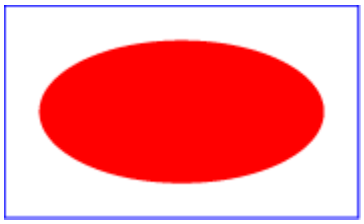
The remote resource (the destination for the link) is defined by a IRI specified by the XLink 'xlink:href' attribute on the 'a' element. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, an element within the current 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="3cm" viewBox="0 0 5 3" version="1.1"
     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



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.

Categories: ‘a’

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- ‘a’
- ‘altGlyphDef’
- ‘clipPath’
- ‘color-profile’
- ‘cursor’
- ‘filter’
- ‘font’
- ‘font-face’

'foreignObject'
'image'
'marker'
'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:
conditional processing attributes
core attributes
graphical event attributes
presentation attributes
xlink attributes
'class'
'style'
'externalResourcesRequired'
'transform'
'xlink:href'
'xlink:show'
'xlink:actuate'
'target'

DOM Interfaces:
SVGAElement

Attribute definitions:

`xlink:show` = **"new" | "replace"**
This attribute provides documentation to XLink-aware processors. If **target="_blank"** then use **xlink:show="new"** else use **'replace'**. In case of a conflict, the target attribute has priority, since it can express a wider range of values. Refer to the XML Linking Language (XLink) [XLINK].
Animatable: no.

`xlink:actuate` = **"onRequest"**
This attribute provides documentation to XLink-aware processors that 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 = "<IRI>"`

The location of the referenced object, expressed as an IRI reference.

Animatable: yes.

`target = "_replace" | "_self" | "_parent" | "_top" | "_blank" | "<XML-Name>"`

This attribute should be used 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 or portion of the target window, frame, pane, tab, or other relevant presentation context (e.g., an HTML or XHTML frame, iframe, or object element) into which a document is to be opened when the link is activated:

_replace · The current SVG image is replaced by the linked content in the same rectangular area in the same frame as the current SVG image.

_self · The current SVG image is replaced by the linked content in the same frame as the current SVG image. If the attribute is not specified, '**_self**' is assumed.

_parent · The immediate frameset parent of the SVG image is replaced by the linked content.

_top · The content of the full window or tab, including any frames, is replaced by the linked content

_blank · A new un-named window or tab is requested for the display of the linked content. If this fails, the result is the same as **_top**

<XML-Name> · Specifies the name of the frame, pane, or other relevant presentation context for display of the linked content. If this already exists, it is re-used, replacing the existing content. If it does not exist, it is created (the same as '**_blank**', except that it now has a name).

Note: The value '**_new**' is *not* a legal value for target (use '**_blank**').

Animatable: yes.

17.3 Linking into SVG content: IRI fragments and SVG views

17.3.1 Introduction: IRI fragments and SVG views

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.

17.3.2 SVG fragment identifiers

To link into a particular view of an SVG document, the IRI 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 IRIs that locate resources of MIME media type "image/svg+xml".

An SVG fragment identifier can come in two 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.
- **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 |
                          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:

- 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))**).

Semicolons used to separate 'SVGViewAttribute' in SVG fragments may be url-escaped (as %3B); this is useful when animating a (semi-colon separated) list of IRIs because otherwise the semicolon would be interpreted as a list separator.

The five types of **SVGViewAttribute** may occur in any order, but each type may only occur at most one time in a correctly formed **SVGViewSpec**.

When a source document performs a link into an SVG document, for example via an HTML anchor element ([HTML4], section 12.2; 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 IRI 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.

17.3.3 Predefined views: the 'view' element

The 'view' element is defined as follows:

Categories: 'view'
None

Content model:
Any number of the following elements, in any order:
descriptive elements

Attributes:
core attributes
'externalResourcesRequired'
'viewBox'

'preserveAspectRatio'
'zoomAndPan'
'viewTarget'

DOM Interfaces:
SVGViewElement

Attribute definitions:

`viewTarget` = "***XML_Name [XML_NAME]****"
Indicates the target object associated with the view.
Animatable: no.

17.3.4 Highlighting views

It is helpful to users if the target element(s) are highlighted. The visual styling of this highlight should be decided by the document author, because the SVG User Agent has no way to determine what changes would make the elements more visible.

The CSS :target selector ([SELECTORS], section 6.2.2) may be used in a stylesheet to provide alternate styling for elements which are the target of links. For example:

```
<style type="text/css">
#foo:target {filter: url(#glow)}
/* when the element with id foo is linked to, use a glow filter */

.bar :target {stroke: green; fill-opacity: 0.5}
/* when any descendants of elements with class bar are linked
   to, make the fill partly transparent and use a green stroke */

:target {stroke: red }
/* for everything else, just use a red stroke */
</style>
```

17.4 DOM interfaces

17.4.1 Interface SVGAElement

The SVGAElement interface corresponds to the 'a' element.

```
interface SVGAElement : SVGElement,
                        SVGURIReference,
                        SVGTests,
                        SVGLangSpace,
                        SVGExternalResourcesRequired,
                        SVGStylable,
                        SVGTransformable {
  readonly attribute SVGAnimatedString target;
};
```

Attributes:

- **target** (readonly SVGAnimatedString)

  Corresponds to attribute 'target' on the given 'a' element.

17.4.2 Interface SVGViewElement

The SVGAElement interface corresponds to the 'view' element.

```
interface SVGViewElement : SVGElement,
                           SVGExternalResourcesRequired,
                           SVGFitToViewBox,
                           SVGZoomAndPan {
  readonly attribute SVGStringList viewTarget;
};
```

Attributes:

- **viewTarget** (readonly SVGStringList)

  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.

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 SVG document fragment. This attribute sets the default scripting language used to process the value strings in event attributes. This language must be used for all instances of script that do not specify their own scripting language. The value ***content-type*** specifies a media type, per MIME Part Two: Media Types [RFC2046]. The default value is **'application/ecmascript'** [RFC4329]. *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' 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.

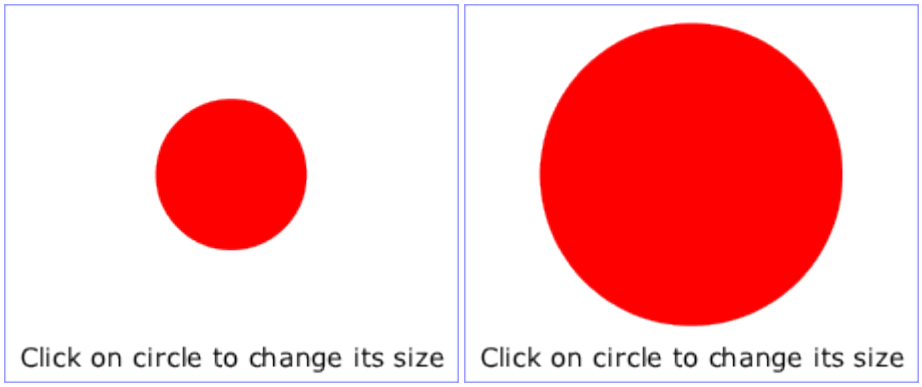
Example script01 defines a function `circle_click` which is called by the 'onclick' event attribute on the 'circle' element. The drawing below on the left is the initial image. The drawing below on the right shows the result after clicking on the circle.

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.

Before attempting to execute the 'script' element the resolved media type value for 'type' must be inspected. If the SVG user agent does not support the scripting language then the 'script' element must not be executed.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="6cm" height="5cm" viewBox="0 0 600 500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example script01 - invoke an ECMAScript function from an onclick event
  </desc>
  <!-- ECMAScript to change the radius with each click -->
  <script type="application/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



Categories: ‘script’
None

Content model:
Any elements or character data.

Attributes:
core attributes
xlink attributes
‘externalResourcesRequired’
‘type’
‘xlink:href’

DOM Interfaces:
SVGScriptElement

Attribute definitions:

`type` = "***content-type***"
Identifies the scripting language for the given ‘script’ element. The value ***content-type*** specifies a media type, per Multipurpose Internet Mail Extensions (MIME) Part Two [RFC2046]. If a ‘type’ is not provided, the value of ‘contentScriptType’ on the ‘svg’ element shall be used, which in turn defaults to "application/ecmascript" [RFC4329]. If a ‘script’ element falls outside of the outermost svg element and the ‘type’ is not provided, the ‘type’ must default to "application/ecmascript" [RFC4329].
Animatable: no.

`xlink:href` = "<iri>"
An IRI reference to an external resource containing the script code.
Animatable: no.

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 values of the event attributes are script which is executed when the given event occurs.
- Event listeners as described in DOM Level 2 Events [DOM2EVENTS] 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.

18.4.1 Event attribute for the SVGLoad event

Below is the definition for the 'onload' event attribute. It can be specified on all of the animation elements and most of the graphics elements and container elements. The 'onload' event attribute is classified as both a graphical event attribute and an animation event attribute. (See the definition for each element to determine whether it can have a graphical event attribute specified on it.)

Attribute definitions:

`onload = "<anything>"`

Specifies some script to execute when "bubbling" or "at target" phase listeners for the SVGLoad event are fired on the element the attribute is specified on.
Animatable: no.

18.4.2 Event attributes on graphics and container elements

Below are the definitions for the graphical event attributes. These can be specified on most graphics elements and container elements. (See the definition for each element to determine whether it can have a graphical event attribute specified on it.)

Note that 'onload', defined above, is also classified as a graphical event attribute.

Attribute definitions:

`onfocusin` = "<anything>"
`onfocusout` = "<anything>"
`onactivate` = "<anything>"
`onclick` = "<anything>"
`onmousedown` = "<anything>"
`onmouseup` = "<anything>"
`onmouseover` = "<anything>"
`onmousemove` = "<anything>"
`onmouseout` = "<anything>"

Specifies some script to execute when "bubbling" or "at target" phase listeners for the corresponding event are fired on the element the attribute is specified on. See the Complete list of support events to determine which event each of these event attributes corresponds to.
Animatable: no.

18.4.3 Document-level event attributes

Below are the definitions for the document event attributes. These can be specified only on 'svg' elements.

Attribute definitions:

`onunload` = "<anything>"
`onabort` = "<anything>"
`onerror` = "<anything>"
`onresize` = "<anything>"
`onscroll` = "<anything>"
`onzoom` = "<anything>"

Specifies some script to execute when "bubbling" or "at target" phase listeners for the corresponding event are fired on the element the attribute is specified on. See the Complete list of support events to determine which event each of these event attributes corresponds to.
Animatable: no.

18.4.4 Animation event attributes

Below are the definitions for the animation event attributes. These can be specified on the animation elements.

Note that 'onload', defined above, is also classified as an animation event attribute.

Attribute definitions:

`onbegin` = "**<anything>**"
`onend` = "**<anything>**"
`onrepeat` = "**<anything>**"

Specifies some script to execute when "bubbling" or "at target" phase listeners for the corresponding event are fired on the element the attribute is specified on. See the Complete list of support events to determine which event each of these event attributes corresponds to.
Animatable: no.

18.5 DOM interfaces

18.5.1 Interface SVGScriptElement

The SVGScriptElement interface corresponds to the 'script' element.

```
interface SVGScriptElement : SVGElement,
                             SVGURIReference,
                             SVGExternalResourcesRequired {
  attribute DOMString type setraises(DOMException);
};
```

Attributes:

- **type** (DOMString)

  Corresponds to attribute 'type' on the given 'script' element.

  Exceptions on setting

  - DOMException, code NO_MODIFICATION_ALLOWED_ERR
    Raised on an attempt to change the value of a read only attribute.

18.5.2 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.)

```
interface SVGZoomEvent : UIEvent {
  readonly attribute SVGRect zoomRectScreen;
  readonly attribute float previousScale;
  readonly attribute SVGPoint previousTranslate;
  readonly attribute float newScale;
  readonly attribute SVGPoint newTranslate;
};
```

Attributes:

- **zoomRectScreen** (readonly SVGRect)

  The specified zoom rectangle in screen units.
  The SVGRect object is read only.

- **previousScale** (readonly float)

  The scale factor from previous zoom operations that was in place before the zoom operation occurred.

- **previousTranslate** (readonly SVGPoint)

  The translation values from previous zoom operations that were in place before the zoom operation occurred.
  The SVGPoint object is read only.

- **newScale** (readonly float)

  The scale factor that will be in place after the zoom operation has been processed.

- **newTranslate** (readonly SVGPoint)

  The translation values that will be in place after the zoom operation has been processed.
  The SVGPoint object is read only.

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 [ECMA-262]. (See example below.)
- SVG has been designed to allow SMIL [SMIL] to use animated or static SVG content as media components.

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) 3.0 Specification* [SMIL].

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 ([SMILANIM], sections 2 and 3). 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* specification [SMILANIM].

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

Although SVG defines 'animateColor', its use is deprecated in favor of simply using the 'animate' element to target properties that can take color values.

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 IRI references via an 'xlink:href' attribute to identify the elements which are to be targets of the animations, as allowed in SMIL 3.0.

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 SVGLoad 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. However, nested 'svg' elements within an SVG document do not constitute document fragments in this sense, and do not define a separate document begin; all times within the nested SVG fragment are relative to the document time defined for the root 'svg' element.

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* specification [SMILANIM]. 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).

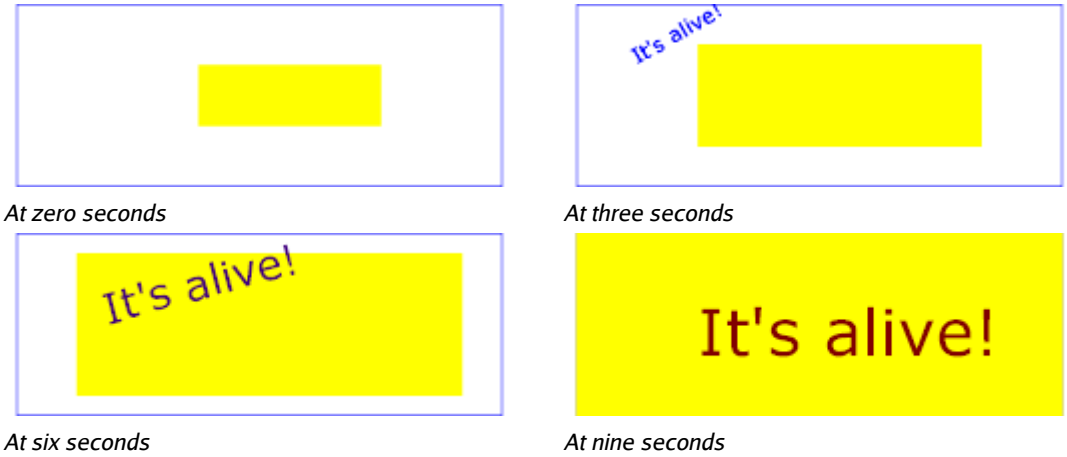
19.2.3 Animation elements example

Example anim01 below demonstrates each of SVG's five animation elements.

```
<?xml version="1.0" standalone="no"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="3cm"  viewBox="0 0 800 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <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',
         'animate' 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" />
      <animate 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>
```

Example anim01



At zero seconds *At three seconds*

At six seconds *At nine seconds*

The sections below describe the various animation attributes and elements.

19.2.4 Attributes to identify the target element for an animation

The following attribute is common to all animation elements and identifies the target element for the animation.

Attribute definitions:

`xlink:href = "<iri>"`

An IRI 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.

<iri> must point to exactly one target element which is capable of being the target of the given animation. If <iri> 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* specification. In particular, see SMIL Animation: Specifying the animation target ([SMILANIM], section 3.1).

19.2.5 Attributes to identify the target attribute or property for an animation

The following attributes are the **animation attribute target attributes**, which identify the target attribute or property for the given target element whose value changes over time.

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* specification. In particular, see SMIL Animation: Specifying the animation target ([SMILANIM], section 3.1).

`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* specification. In particular, see SMIL Animation: Specifying the animation target ([SMILANIM], section 3.1).

19.2.6 Animation with namespaces

Example animns01 below shows a namespace prefix being resolved to a namespace name in the scope of the ref-

erencing element, and that namespace name being used (regardless of the prefix which happens to be used in the target scope) to identify the attribute being animated.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Demonstration of the resolution of namespaces for animation</title>
  <!-- at the point of definition, the QName a:href resolves to the namespace
       name "http://www.w3.org/1999/xlink" and the local name "href" -->
  <g xmlns:a="http://www.w3.org/1999/xlink">
    <animate attributeName="a:href" xlink:href="#foo" dur="2s" to="two.png" fill="freeze"/>
  </g>
  <!-- at the point of use, the namespace name "http://www.w3.org/1999/xlink"
       happens to be bound to the namespace prefix 'b' while the prefix
       'xlink' is bound to a different namespace name -->
  <g xmlns:b="http://www.w3.org/1999/xlink" xmlns:xlink="http://example.net/bar">
    <image xml:id="foo" b:href="one.png" x="35" y="50" width="410" height="160"/>
  </g>
</svg>
```

19.2.7 Paced animation and complex types

Paced animations assume a notion of distance between the various animation values defined by the 'to', 'from', 'by' and 'values' attributes. Distance is defined only for scalar types (such as <length>), colors and the subset of transformation types that are supported by 'animateTransform'. In the list of distance functions below, V_a and V_b represent the two values the distance between which is being calculated.

Since paced animation is intended to produce an animation with an even pace of change, it does not make sense to define distance functions for all data types. Distance can be usefully defined for types whose values are *n*-dimensional vectors (including scalars, which are 1-dimensional vectors). For example, a <length> value is a scalar value, and a <color> value is a 3-dimensional vector. Thus attributes of these types can have paced animation applied to them. On the other hand, a <list-of-length> (as used by '**stroke-dasharray**') is a list of scalars (1-dimensional vectors), and <list-of-points> (as used by the 'points' attribute on a 'polygon') is a list of 2-dimensional vectors. Therefore, these types do not have a distance function defined and cannot have paced animation applied to them.

The distance functions for types that support paced animation are as follows:

<coordinate>, <integer>, <length> and <number>

distance(V_a, V_b) = $|V_a - V_b|$

Examples: animating the 'x' attribute on a 'rect', or the '**stroke-width**' property on a 'circle'.

<color>

distance(V_a, V_b) = $\text{sqrt}((V_a.\text{red} - V_b.\text{red})^2 + (V_a.\text{green} - V_b.\text{green})^2 + (V_a.\text{blue} - V_b.\text{blue})^2)$, where:

V_i.red is the red component of the V_i color value,

V_i.green is the green component of the V_i color value, and

V_i.blue is the blue component of the V_i color value.

Each of the color component values is usually in the range [0, 1], where 0 represents none of that color component, and 1 represents the maximum amount of that color component, in the sRGB gamut [SRGB]. Since <color> values may specify colors outside of the sRGB gamut, these component values may lie outside the range [0, 1].

Example: animating the '**fill**' property on an 'ellipse'.

Transform definitions of type 'translate'

distance(V_a, V_b) = $\text{sqrt}((V_a.\text{tx} - V_b.\text{tx})^2 + (V_a.\text{ty} - V_b.\text{ty})^2)$, where:

V_i.tx is the *x* component of the V_i translation transform value, and
V_i.ty is the *y* component of the V_i translation transform value.

Example (for all transform definition types): animating the 'transform' attribute on a 'g' using 'animateTransform'.

Transform definitions of type 'scale'

distance(V_a, V_b) = sqrt((V_a.sx – V_b.sx)^2 + (V_a.sy – V_b.sy)^2), where:
V_i.sx is the *x* component of the V_i scale transform value, and
V_i.sy is the *y* component of the V_i scale transform value.

Note that, as when specifying scale transformations in a <transform-list>, if the *y* component of the scale is omitted it is implicitly equal to the *x* component.

Transform definitions of type 'rotate', 'skewX' and 'skewY'

distance(V_a, V_b) = sqrt((V_a.angle – V_b.angle)^2), where:
V_i.angle is the angle component of the V_i rotation or skew transform value.

Since the distance function for rotations is not in terms of the rotation center point components, a paced animation that changes the rotation center point may not appear to have a paced movement when the animation is applied.

Distance functions for all other data types are not defined. If **calcMode="paced"** is used on an animation of an attribute or property whose type is not one of those listed above, the animation effect is undefined. SVG user agents may choose to perform the animation as if **calcMode="linear"**, but this is not required. Authors are recommended not to specify paced animation on types not listed above.

19.2.8 Attributes to control the timing of the animation

The following attributes are the **animation timing attributes**. They 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.

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. Negative begin times are entirely valid and easy to compute, as long as there is a resolved document begin time.

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 *Representation of dates and times* [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* specification. In particular, see SMIL Animation: 'begin' attribute ([SMILANIM], section 3.2.1).

`dur` = **Clock-value** | **"media"** | **"indefinite"**

Specifies the simple duration.

The attribute value can be one 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* specification. In particular, see SMIL Animation: 'dur' attribute ([SMILANIM], section 3.2.1).

`end` = "**end-value-list**"

Defines an end value for the animation that can constrain the active duration. The attribute value is a semi-colon 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 an 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* specification. In particular, see SMIL Animation: 'end' attribute ([SMILANIM], section 3.3.2).

`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* specification. In particular, see SMIL Animation: 'min' attribute ([SMILANIM], section 3.3.3).

`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* specification. In particular, see SMIL Animation: 'max' attribute ([SMILANIM], section 3.3.3).

`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* specification. In particular, see SMIL Animation: 'restart' attribute ([SMILANIM], section 3.3.7).

`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* specification. In particular, see SMIL Animation: 'repeatCount' attribute ([SMILANIM], section 3.3.1).

`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* specification. In particular, see SMIL Animation: 'repeatDur' attribute ([SMILANIM], section 3.3.1).

`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* specification. In particular, see SMIL Animation: 'fill' attribute ([SMILANIM], section 3.3.5).

The *SMIL Animation* specification [SMILANIM] defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the SMIL Animation specification is the normative definition of the processing rules for the above attributes.

19.2.8.1 Clock values

Clock values have the same syntax as in *SMIL Animation* specification [SMILANIM]. The grammar for clock values 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
```

19.2.9 Attributes that define animation values over time

The following attributes are the **animation value attributes**. They are common to elements 'animate', 'animateColor', 'animateMotion' 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.

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 the data types for which there is an appropriate distance function defined, which includes only scalar numeric types plus the types listed in Paced animation and complex types. If '**paced**' is specified, any 'keyTimes' or 'keySplines' will be ignored. For 'animateMotion', this is the default 'calcMode'. Authors are discouraged from using paced animation on types that do not have a distance function defined, due to its unpredictable behavior in some user agents.

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* specification. In particular, see SMIL Animation: 'calcMode' attribute ([SMILANIM], section 3.2.3).

`values = "<list>"`

A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the 'attributeType' domain. Per the SMIL specification, leading and trailing white space, and white space before and after semicolon separators, is allowed and will be ignored. Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for this attribute is the *SMIL Animation* specification. In particular, see SMIL Animation: 'values' attribute ([SMILANIM], section 3.2.2).

`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.

For animations specified with a 'values' list, the 'keyTimes' attribute if specified must have exactly as many values as there are in the 'values' attribute. For from/to/by animations, the 'keyTimes' attribute if specified must have two values.

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 key time associated with each value defines when the value is set; values are interpolated between the key times.
- 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.

Because paced animation interpolation is unspecified for some value types, authors are encouraged to use **'linear'** animation interpolation with calculated 'keyTimes' to achieve particular interpolation behavior for these types.

Except for any SVG-specific rules explicitly mentioned in this specification, the normative definition for

this attribute is the *SMIL Animation* specification. In particular, see SMIL Animation: 'keyTimes' attribute ([SMILANIM], section 3.2.3).

`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. Note: SMIL allows these values to be separated either by commas with optional whitespace, or by whitespace alone. 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* specification. In particular, see SMIL Animation: 'keySplines' attribute ([SMILANIM], section 3.2.3).

`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* specification. In particular, see SMIL Animation: 'from' attribute ([SMILANIM], section 3.2.2).

`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* specification. In particular, see SMIL Animation: 'to' attribute ([SMILANIM], section 3.2.2).

`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* specification. In particular, see SMIL Animation: 'by' attribute ([SMILANIM], section 3.2.2).

The *SMIL Animation* specification [SMILANIM] defines the detailed processing rules associated with the above attributes. Except for any SVG-specific rules explicitly mentioned in this specification, the SMIL Animation 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).

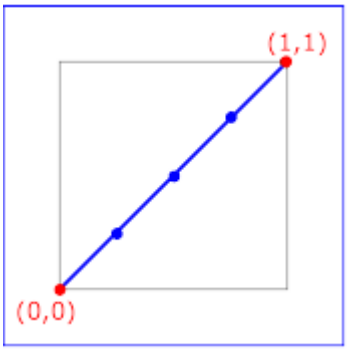
If a list of values is used, the animation will apply the values in order over the course of the animation. If a list of 'values' is specified, any 'from', 'to' and 'by' attribute values are ignored.

The processing rules for the variants of *from/by/to* animations are described in Animation function values with the following exception.

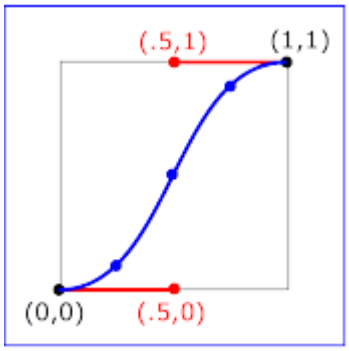
In order to provide behavior that is intuitive and consistent between discrete animations with an explicitly specified 'from' attribute (e.g. "from-to animation") and those where the underlying value is used (e.g. "to animation"), the behavior of discrete to-animation in SVG deviates from the definition in SMIL Animation. As with a discrete from-to animation, a discrete to animation will set the underlying value for the first half of the simple duration (or, if a 'keyTimes' list is provided, until the simple duration specified by the second value in the 'keyTimes' list) and the 'to' value for the remainder of the simple duration.

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 function that the 'keySplines' attribute defines. 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.

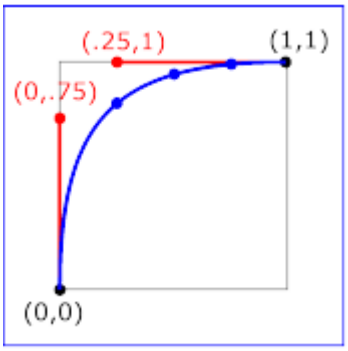
Examples of keySplines



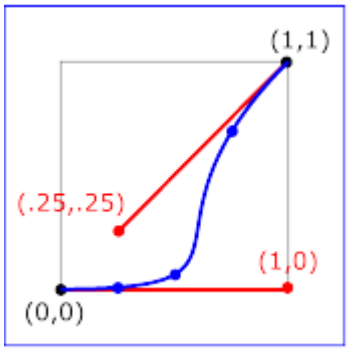
keySplines="0 0 1 1" (the default)



keySplines=".5 0 .5 1"



keySplines="0 .75 .25 1"



keySplines="1 0 .25 .25"

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:

Value of 'keySplines'	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], pp. 488-491.

19.2.10 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 the **animation addition attributes**, which are common to elements 'animate', 'animateColor', 'animateMotion' and 'animateTransform'.

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* specification. In particular, see SMIL Animation: 'additive' attribute ([SMILANIM], section 3.3.6).

`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* specification. In particular, see SMIL Animation: 'accumulate' attribute ([SMILANIM], section 3.3.1).

19.2.11 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.

19.2.12 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* specification. In particular, see SMIL Animation: 'animate' element ([SMILANIM], section 4.1).

Categories: 'animate'

Animation element

Content model:

Any number of the following elements, in any order:

descriptive elements

Attributes:
conditional processing attributes
core attributes
animation event attributes
xlink attributes
animation attribute target attributes
animation timing attributes
animation value attributes
animation addition attributes
presentation attributes
'externalResourcesRequired'

DOM Interfaces:
SVGAnimateElement

The '**color-interpolation**' property applies to color interpolations that result from animations using the 'animate' element.

For a list of attributes and properties that can be animated using the 'animate' element, see Elements, attributes and properties that can be animated.

19.2.13 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* specification. In particular, see SMIL Animation: 'set' element ([SMILANIM], section 4.2).

Categories: 'set'
Animation element

Content model:
Any number of the following elements, in any order:
descriptive elements

Attributes:
conditional processing attributes
core attributes

animation event attributes
xlink attributes
animation attribute target attributes
animation timing attributes
'externalResourcesRequired'
'to'

DOM Interfaces:
SVGSetElement

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.

For a list of attributes and properties that can be animated using the 'set' element, see Elements, attributes and properties that can be animated.

19.2.14 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* specification. In particular, see SMIL Animation: 'animateMotion' element ([SMILANIM], section 4.3).

Categories: 'animateMotion'
Animation element

Content model:
Any number of descriptive elements and at most one 'mpath' element, in any order.

Attributes:
conditional processing attributes
core attributes
animation event attributes
xlink attributes
animation timing attributes
animation value attributes
animation addition attributes
'externalResourcesRequired'

'path'
'keyPoints'
'rotate'
'origin'

DOM Interfaces:
SVGAnimateMotionElement

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` = "**<number> | auto | auto-reverse**"
The 'rotate' attribute post-multiplies a supplemental transformation matrix onto the CTM of the target element 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.

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.

<number>
: Indicates that the target element has a constant rotation transformation applied to it, where the rotation angle is the specified number of degrees.

The default value is **'0'**.

`origin` = **"default"**
: The 'origin' attribute is defined in the SMIL Animation specification ([SMILANIM], section 4.3). It has no effect in SVG.

Categories: **'mpath'**
: None

Content model:
: Any number of the following elements, in any order:
descriptive elements

Attributes:
: core attributes
xlink attributes
'externalResourcesRequired'
'xlink:href'

DOM Interfaces:
: SVGMPathElement

Attribute definitions:

`xlink:href` = "**<iri>**"
: An IRI reference to the 'path' element which defines the motion path.
Animatable: no.

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' and 'to'. Regarding determining the points which correspond to the 'keyTimes' attributes, the 'keyPoints' attribute overrides 'path', which overrides 'values', which overrides 'from', 'by' and 'to'.

At any time *t* within a motion path animation of duration *dur*, the computed coordinate (x,y) along the mo-

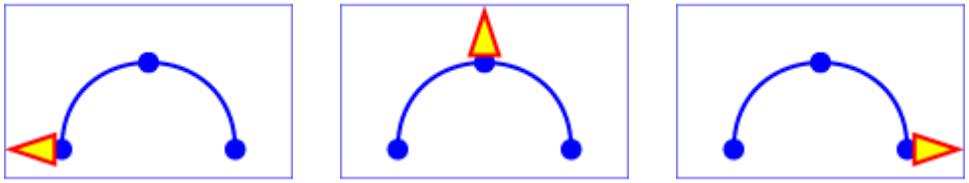
tion path is determined by finding the point (x,y) which is *t/dur* distance along the motion path using the user agent's distance along the path algorithm.

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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="3cm"  viewBox="0 0 500 300"
     xmlns="http://www.w3.org/2000/svg" version="1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink" >
  <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 id="path1" 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" rotate="auto" >
       <mpath xlink:href="#path1"/>
    </animateMotion>
  </path>
</svg>
```

Example animMotion01



At zero seconds *At three seconds* *At six seconds*

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.

19.2.15 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* specification. In particular, see SMIL Animation: 'animateColor' element ([SMILANIM], section 4.4).

Categories: 'animateColor'
Animation element

Content model:
Any number of the following elements, in any order:
descriptive elements

Attributes:
conditional processing attributes
core attributes
animation event attributes
xlink attributes
animation attribute target attributes
animation timing attributes
animation value attributes
animation addition attributes
presentation attributes
'externalResourcesRequired'

DOM Interfaces:
SVGAnimateColorElement

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> <icccolor>?
```

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.

The use of 'animateColor' is deprecated, since all of its functionality can be achieved simply by using 'animate' to target properties that can take color values. The 'animateColor' element may be dropped from a future version of the SVG specification.

For a list of attributes and properties that can be animated using the 'animateColor' element, see Elements, attributes and properties that can be animated.

19.2.16 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.

Categories: 'animateTransform'

Animation element

Content model:

Any number of the following elements, in any order:

descriptive elements

Attributes:

conditional processing attributes
core attributes
animation event attributes
xlink attributes
animation attribute target attributes
animation timing attributes
animation value attributes
animation addition attributes
'externalResourcesRequired'
'type'

DOM Interfaces:

SVGAnimateTransformElement

Attribute definitions:

`type` = **"translate | scale | rotate | skewX | skewY"**
Indicates the type of transformation which is to have its values change over time. If the attribute is not specified, then the effect is as if a value of **'translate'** were specified.

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'.

The animation effect for 'animateTransform' is post-multiplied to the underlying value for additive 'animateTransform' animations (see below) instead of added to the underlying value, due to the specific behavior of 'animateTransform'.

From-to, *from-by* and *by animations* are defined in SMIL to be equivalent to a corresponding *values animation*. See the Animation function values section of SMIL Animation ([SMILANIM], section 3.2.2). However, *to animations* are a mixture of additive and non-additive behavior, as described in the How from, to and by attributes affect additive behavior section of SMIL Animation ([SMILANIM], section 3.3.6). *To animations* provide specific functionality to get a smooth change from the underlying value to the 'to' attribute value, which conflicts mathematically with the requirement for additive transform animations to be post-multiplied. As a consequence, in SVG 1.1 the behavior of *to animations* for 'animateTransform' is undefined. Authors are suggested to use *from-to*, *from-by*, *by* or *values animations* to achieve any desired transform animation.

If 'calcMode' has the value **'paced'**, then the "distance" for the transformation is calculated as further described in Paced animations and complex types.

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)" ... />
```

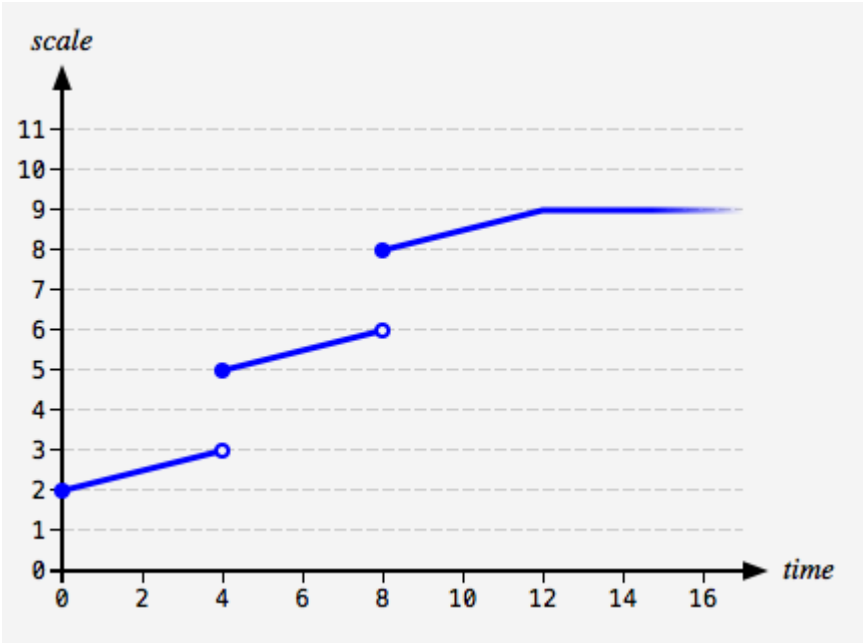
Note that the zero value used when performing a *by animation* with **type="scale"** is indeed 0. Thus, performing the following animation causes the rectangle to be invisible at time 0s (since the animated transform list value is '**scale(0)**'), and be scaled back to its original size at time 5s (since the animated transform list value is '**scale(1)**'):

```
<rect width="100" height="100">
  <animateTransform attributeName="transform" attributeType="XML"
                    type="scale" by="1" dur="5s" fill="freeze"/>
</rect>
```

When a transform animation has **accumulate='sum'**, the accumulation that occurs for each completed repetition of the animation is computed on the values specified in the 'animateTransform' element's animation value attributes (i.e., 'values', 'from', 'to' and 'by') and not on the transformation matrix that these values represent. For example, in the following code snippet, 3 is added to the scale value at the start of each repetition:

```
<rect width="100" height="100">
  <animateTransform attributeName="transform" attributeType="XML"
                    type="scale" from="2" to="3" repeatCount="3" dur="4s"
                    fill="freeze"/>
</rect>
```

The following graph and table shows the animated 'transform' value on the 'rect' over the course of the animation:



Time	Value
0s	scale(2)
1s	scale(2.25)
2s	scale(2.5)
3s	scale(2.75)
4s	scale(5)
5s	scale(5.25)
6s	scale(5.5)
7s	scale(5.75)
8s	scale(8)
9s	scale(8.25)
10s	scale(8.5)
11s	scale(8.75)
12s	scale(9)

Transform item types that can have multiple values – **'translate'**, **'scale'** and **'rotate'** – are treated as vectors and accumulation is performed with vector addition. Optional values that are omitted are taken to have their usual implied value: **1** for the **<sy>** component of a **'scale'** and **0** for the **<tx>** component of a **'translate'** and the **<cx cy>** components of a **'rotate'**.

For example, consider the following code snippet, which has a cumulative transform animation of type **'rotate'**:

```
<rect width="100" height="100">
  <animateTransform attributeName="transform" attributeType="XML"
                    type="rotate" from="0 30 40" to="10 30 40"
                    repeatCount="2" dur="1s" fill="freeze"/>
</rect>
```

At time 1 second, the animated value of 'transform' on the 'rect' will jump from **'rotate(10 30 40)'** to **'rotate(10 60 80)'**, because the effect of the accumulation is to take the value at the end of the first repetition, **'10 30 40'**, and add to it the value at simple duration t = 0s, which is **'0 30 40'**.

For a list of attributes and properties that can be animated using the 'animateTransform' element, see Elements, attributes and properties that can be animated.

19.2.17 Elements, attributes and properties that can be animated

The following lists all of the elements which can be animated by an 'animateMotion' element:

- 'svg'
- '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.

Some properties are defined as being animatable but only for non-additive animations:

Animatable: yes (non-additive).

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 additive if each value can be converted to an RGB color.
<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-*T*s>	no	yes	yes	no	no	
<number>	yes	yes	yes	no	no	
<paint>	yes	yes	yes	yes	no	Only additive if each value can be converted to an RGB color.
<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.
<iri>	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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4cm" height="2cm" viewBox="0 0 400 200"
     xmlns="http://www.w3.org/2000/svg"
     onload="StartAnimation(evt)" version="1.1">
  <script type="application/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>
```

Example dom01

At zero seconds *At 2.5 seconds* *At five seconds*

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

Below are the DOM interfaces for the elements defined in this chapter. In addition, ElementTimeControl and TimeEvent, which are from SMIL Animation, are included here for easy reference.

19.4.1 Interface ElementTimeControl

SMIL Animation supports several methods for controlling the behavior of animation: `beginElement()`, `beginElementAt()`, `endElement()` and `endElementAt()`. These methods are used to begin and end the active duration of an element. Authors can (but are not required to) declare the timing to respond to the DOM using the following syntax:

```
<animate begin="indefinite" end="indefinite" .../>
```

If a DOM method call is made to begin or end the element (using `beginElement()`, `beginElementAt()`, `endElement()` or `endElementAt()`), each method call creates a single instance time (in the appropriate instance times list). These times are then interpreted as part of the semantics of lists of times, as described in Evaluation of begin and end time lists.

- The instance time associated with a `beginElement()` or `endElement()` call is the current presentation time at the time of the DOM method call.
- The instance time associated with a `beginElementAt()` or `endElementAt()` call is the current presentation time at the time of the DOM method call, plus or minus the specified offset.

- Note that `beginElement()` is subject to the 'restart' attribute in the same manner that event-based begin timing is. Refer also to SMIL Animation: Restarting animation ([SMILANIM], section 3.3.7).

The expectation of the following interface is that an instance of the ElementTimeControl interface can be obtained by using binding-specific casting methods on an instance of an animation element. A DOM application can use the hasFeature method of the DOMImplementation interface to determine whether the ElementTimeControl interface is supported or not. The feature string for this interface is "TimeControl".

```
interface ElementTimeControl {
  void beginElement();
  void beginElementAt(in float offset);
  void endElement();
  void endElementAt(in float offset);
};
```

Operations:

- void **beginElement**()

  Creates a begin instance time for the current time. The new instance time is added to the *begin instance times list*. The behavior of this method is equivalent to `beginElementAt(0)`.

- void **beginElementAt**(in float *offset*)

  Creates a begin instance time for the current time plus the specified offset. The new instance time is added to the *begin instance times list*.

  Parameters

  - float *offset*
    The offset from the current document time, in seconds, at which to begin the element.

- void **endElement**()

  Creates an end instance time for the current time. The new instance time is added to the *end instance times list*. The behavior of this method is equivalent to `endElementAt(0)`.

- void **endElementAt**(in float *offset*)

  Creates a end instance time for the current time plus the specified offset. The new instance time is added to the *end instance times list*.

  Parameters

  - float *offset*
    offset from the current document time, in seconds, at which to end the element.

For the corresponding Java binding, see section 6.4 of SMIL Animation [SMILANIM].

19.4.2 Interface TimeEvent

The TimeEvent interface, 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)

```
interface TimeEvent : Event {
```

```
  readonly attribute AbstractView view;
  readonly attribute long detail;

  void initTimeEvent(in DOMString typeArg, in AbstractView viewArg, in long detailArg);
};
```

Attributes:

- **view** (readonly AbstractView)

  The view attribute identifies the AbstractView [DOM2VIEWS] from which the event was generated.

- **detail** (readonly long)

  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.

Operations:

- void **initTimeEvent**(in DOMString *typeArg*, in AbstractView *viewArg*, in long *detailArg*)

  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

  - DOMString *typeArg*
    Specifies the event type.
  - AbstractView *viewArg*
    Specifies the Event's AbstractView.
  - long *detailArg*
    Specifies the Event's detail.

For the corresponding Java binding, see section 6.4 of SMIL Animation [SMILANIM].

19.4.3 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 correspond-

ing 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 DOM Level 2 Core [DOM2].

```
interface SVGAnimationElement : SVGElement,
                                SVGTests,
                                SVGExternalResourcesRequired,
                                ElementTimeControl {

  readonly attribute SVGElement targetElement;

  float getStartTime() raises(DOMException);
  float getCurrentTime();
  float getSimpleDuration() raises(DOMException);
};
```

Attributes:

- **targetElement** (readonly SVGElement)

  The element which is being animated.

Operations:

- float **getStartTime**()

  Returns the begin time, in seconds, for this animation element's current interval, if it exists, regardless of whether the interval has begun yet. If there is no current interval, then a DOMException with code INVALID_STATE_ERR is thrown.

  Returns
  The start time, in seconds, of this animation element's current interval.

  Exceptions

  - DOMException, code INVALID_STATE_ERR
    The animation element does not have a current interval.

- float **getCurrentTime**()

  Returns the current time in seconds relative to time zero for the given time container.

  Returns
  The current time in seconds relative to time zero for the given time container.

- float **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.

  Returns

  number of seconds for the simple duration for this animation.

  Exceptions

  - DOMException, code NOT_SUPPORTED_ERR
    The simple duration is not determined on the given element.

19.4.4 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.

```
interface SVGAnimateElement : SVGAnimationElement,
                              SVGStylable {
};
```

19.4.5 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.

```
interface SVGSetElement : SVGAnimationElement {
};
```

19.4.6 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.

```
interface SVGAnimateMotionElement : SVGAnimationElement {
};
```

19.4.7 Interface SVGMPathElement

The SVGMPathElement interface corresponds to the 'mpath' element.

```
interface SVGMPathElement : SVGElement,
                            SVGURIReference,
```

```
                          SVGExternalResourcesRequired {
};
```

19.4.8 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.

```
interface SVGAnimateColorElement : SVGAnimationElement,
                                   SVGStylable {
};
```

19.4.9 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.

```
interface SVGAnimateTransformElement : SVGAnimationElement {
};
```

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 CSS2 ([CSS2], section 15.1) 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], chapter 15). 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-face' 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" version="1.1"
  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 ([CSS2], section 15.3.1) 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%" version="1.1"
 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" version="1.1"
 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 referenced font</text>
</svg>
```

20.3 The 'font' element

The 'font' element defines an SVG font.

Categories: 'font'

None

Content model:

Any number of the following elements, in any order:

- descriptive elements
- 'font-face'
- 'glyph'
- 'hkern'
- 'missing-glyph'
- 'vkern'

Attributes:

core attributes

presentation attributes

'class'
'style'
'externalResourcesRequired'
'horiz-origin-x'
'horiz-origin-y'
'horiz-adv-x'
'vert-origin-x'
'vert-origin-y'
'vert-adv-y'

DOM Interfaces:
SVGFontElement

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.

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 a single path data specification within the 'd' attribute, arbitrary SVG as content within the 'glyph', or both. These two alternatives are processed differently (see below).

Categories: 'glyph'
Container element

Content model:
Any number of the following elements, in any order:
animation elements
descriptive elements
shape elements
structural elements
gradient elements
'a'
'altGlyphDef'
'clipPath'
'color-profile'
'cursor'
'filter'
'font'
'font-face'
'foreignObject'
'image'
'marker'
'mask'

'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:

core attributes
presentation attributes
'class'
'style'
'd'
'horiz-adv-x'
'vert-origin-x'
'vert-origin-y'
'vert-adv-y'
'unicode'
'glyph-name'
'orientation'
'arabic-form'
'lang'

DOM Interfaces:

SVGGlyphElement

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 BCP 47 [BCP47]. 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.

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], chapter 6) 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 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 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.

Categories: 'missing-glyph'

Container element

Content model:

Any number of the following elements, in any order:

- animation elements
- descriptive elements
- shape elements
- structural elements
- gradient elements
- 'a'
- 'altGlyphDef'
- 'clipPath'
- 'color-profile'
- 'cursor'
- 'filter'
- 'font'

'font-face'
'foreignObject'
'image'
'marker'
'mask'
'pattern'
'script'
'style'
'switch'
'text'
'view'

Attributes:

core attributes
presentation attributes
'class'
'style'
'd'
'horiz-adv-x'
'vert-origin-x'
'vert-origin-y'
'vert-adv-y'

DOM Interfaces:

SVGMissingGlyphElement

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.)

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.

Categories: 'hkern'

None

Content model:

Empty.

Attributes:

core attributes
'u1'
'g1'
'u2'
'g2'
'k'

DOM Interfaces:

SVGHKernElement

'vkern'

Categories:
None

Content model:
Empty.

Attributes:
core attributes
'u1'
'g1'
'u2'
'g2'
'k'

DOM Interfaces:
SVGVKernElement

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; [CSS2], section 15.3.3) 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 'glyph-name' 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 'glyph-name' 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` = "[<character> | <urange>] [, [<character> | <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. This attribute is required.
Animatable: no.

At least one each of 'u1' or 'g1' and at least one of 'u2' or 'g2' must be provided.

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 Fonts chapter in the CSS2 specification ([CSS2], chapter 15).

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 ([CSS2], section 15.3.1) 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 ([CSS2], section 15.3.1). 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.

Categories: ‘font-face’

None

Content model:

Any number of descriptive elements and at most one ‘font-face-src’ element, in any order.

Attributes:

core attributes
‘font-family’
‘font-style’
‘font-variant’
‘font-weight’
‘font-stretch’
‘font-size’
‘unicode-range’
‘units-per-em’
‘panose-1’
‘stemv’
‘stemh’
‘slope’
‘cap-height’
‘x-height’
‘accent-height’
‘ascent’
‘descent’
‘widths’
‘bbox’
‘ideographic’
‘alphabetic’
‘mathematical’
‘hanging’
‘v-ideographic’
‘v-alphabetic’
‘v-mathematical’
‘v-hanging’
‘underline-position’
‘underline-thickness’
‘strikethrough-position’
‘strikethrough-thickness’
‘overline-position’
‘overline-thickness’

DOM Interfaces:
SVGFontFaceElement

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 con-

densed 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 ([CSS2], section 15.3.3).
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 ([CSS2], section 15.3.6).
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 ([CSS2], section 15.3.6).

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.

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.

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.

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.

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.

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.

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.

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.

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.
Animatable: no.

`underline-position` = "**<number>**"
The ideal position of an underline within the font coordinate system.
Animatable: no.

`underline-thickness` = "**<number>**"
The ideal thickness of an underline, expressed as a length within the font coordinate system.
Animatable: no.

`strikethrough-position` = "**<number>**"
The ideal position of a strike-through within the font coordinate system.
Animatable: no.

`strikethrough-thickness` = "**<number>**"
The ideal thickness of a strike-through, expressed as a length within the font coordinate system.
Animatable: no.

`overline-position` = "**<number>**"
The ideal position of an overline within the font coordinate system.
Animatable: no.

`overline-thickness` = "**<number>**"
The ideal thickness of an overline, expressed as a length within the font coordinate system.
Animatable: no.

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 ([CSS2], sections 15.3.1 and 15.3.5.)

20.8.4 The 'font-face-src' element

The 'font-face-src' element, together with the 'font-face-uri' and 'font-face-format' elements described in the following sections, 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 ([CSS2], sections 15.3.1 and 15.3.5).

A 'font-face-src' element contains 'font-face-uri' and 'font-face-name' elements, which are used for referencing external and local fonts, respectively.

Categories: 'font-face-src'
None

Content model:
One or more of the following elements, in any order:

'font-face-name'
'font-face-uri'

Attributes:
core attributes

DOM Interfaces:
SVGFontFaceSrcElement

20.8.5 The 'font-face-uri' and 'font-face-format' elements

The 'font-face-uri' element is used within a 'font-face-src' element to reference a font defined inside or outside of the current SVG document.

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 [RFC3986]. 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).

Categories: 'font-face-uri'
None

Content model:
Any number of the following elements, in any order:
'font-face-format'

Attributes:
core attributes
xlink attributes
'xlink:href'

DOM Interfaces:
SVGFontFaceUriElement

Attribute definitions:

`xlink:href = "<IRI>"`
The 'xlink:href' attribute specifies the location of the referenced font.
Animatable: no.

Child 'font-face-format' elements of a 'font-face-uri' element are used to specify the supported formats of the font referenced by that 'font-face-uri' element. They correspond to entries in a **format(...)** clause of the '**src**' descriptor in an @font-face rule.

Categories: ‘font-face-format’
None

Content model:
Empty.

Attributes:
core attributes
‘string’

DOM Interfaces:
SVGFontFaceFormatElement

Attribute definitions:

`string = "<anything>"`
The ‘string’ attribute is a hint to the user agent, and specifies a list of formats that the font referenced by the parent ‘font-face-uri’ element supports. The syntax of the attribute value is a format string as defined in CSS2, such as **'truetype'**. Refer to the description of the 'src' descriptor in CSS2 for details on how the format hint is interpreted ([CSS2], section 15.3.5).
Animatable: no.

20.8.6 The ‘font-face-name’ element

The ‘font-face-name’ element is used within a ‘font-face-src’ element to reference a local font by name. It corresponds to a **local(...)** clause in an @font-face rule ‘**src**’ descriptor.

Categories: ‘font-face-name’
None

Content model:
Empty.

Attributes:
core attributes
‘name’

DOM Interfaces:
SVGFontFaceNameElement

Attribute definitions:

`name = "<anything>"`

The 'name' attribute specifies the name of a local font. Unlike the syntax allowed between the parentheses of the **local(...)** clause in an @font-face rule '**src**' descriptor, the font name specified in this attribute is not surrounded in single or double quotes. Refer to the description of the 'src' descriptor in CSS2 for details on how the font name is interpreted ([CSS2], section 15.3.5).
Animatable: no.

20.9 DOM interfaces

20.9.1 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.

```
interface SVGFontElement : SVGElement,
                           SVGExternalResourcesRequired,
                           SVGStylable {
};
```

20.9.2 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.

```
interface SVGGlyphElement : SVGElement,
                            SVGStylable {
};
```

20.9.3 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.

```
interface SVGMissingGlyphElement : SVGElement,
                                   SVGStylable {
};
```

20.9.4 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.

```
interface SVGHKernElement : SVGElement {
};
```

20.9.5 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.

```
interface SVGVKernElement : SVGElement {
};
```

20.9.6 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.

```
interface SVGFontFaceElement : SVGElement {
};
```

20.9.7 Interface SVGFontFaceSrcElement

The SVGFontFaceSrcElement interface corresponds to the 'font-face-src' element.

Object-oriented access to the attributes of the 'font-face-src' element via the SVG DOM is not available.

```
interface SVGFontFaceSrcElement : SVGElement {
};
```

20.9.8 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.

```
interface SVGFontFaceUriElement : SVGElement {
};
```

20.9.9 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.

```
interface SVGFontFaceFormatElement : SVGElement {
};
```

20.9.10 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.

```
interface SVGFontFaceNameElement : SVGElement {
};
```

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 *RDF Primer* is the first in a set of six documents that define the Resource Description Framework [RDF-PRIMER].

Another activity relevant to most applications of metadata is the Dublin Core [DCORE], 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 ([XML10], section 3.2.2) 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 it should choose the first one.

Categories: 'metadata'
Descriptive element

Content model:
Any elements or character data.

Attributes:
core attributes

DOM Interfaces:
SVGMetadataElement

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" version="1.1"
    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

21.4.1 Interface SVGMetadataElement

The SVGMetadataElement interface corresponds to the 'metadata' element.

```
interface SVGMetadataElement : SVGElement {
};
```

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 3.0 specification [SMIL]) 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, 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>
```

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" version="1.1"
     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 [MATHML].

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

Categories: 'foreignObject'

None

Content model:

Any elements or character data.

Attributes:

core attributes
conditional processing attributes
graphical event attributes
presentation attributes
'class'

'style'
'externalResourcesRequired'
'transform'
'x'
'y'
'width'
'height'

DOM Interfaces:
SVGForeignObjectElement

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.

23.4 An example

Here is an example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in" version="1.1"
 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 -->
      <body xmlns="http://www.w3.org/1999/xhtml">
        <p>Here is a paragraph that requires word wrap</p>
      </body>
    </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

Using foreign namespaces as an extension mechanism adds flexibility, is readily handled by validation technologies like NVDL and RelaxNG, but typically breaks DTD validation unless the DTD has explicit extensibility hooks.

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. This facility may be used if DTD validation is desired.

The DTD defines an extension entity for most of SVG elements. For example, the 'view' element is defined in the DTD as follows:

```
<!ENTITY % SVG.view.extra.content "" >

<!ENTITY % SVG.view.element "INCLUDE" >
<![%SVG.view.element;[
<!ENTITY % SVG.view.content
    "( %SVG.Description.class; %SVG.view.extra.content; )*"
>
<!ELEMENT %SVG.view.qname; %SVG.view.content; >
<!-- end of SVG.view.element -->]]>

<!ENTITY % SVG.view.attlist "INCLUDE" >
```

```
<![%SVG.view.attlist;[
<!ATTLIST %SVG.view.qname;
    %SVG.Core.attrib;
    %SVG.External.attrib;
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
    zoomAndPan ( disable | magnify ) 'magnify'
    viewTarget CDATA #IMPLIED
>
<!-- end of SVG.view.attlist -->]]>
```

The entity `SVG.view.extra.content` can be defined in the internal DTD subset to add custom sub-elements attributes to the 'view' element within a given document, and an `<!ATTLIST>` can be used to add custom attributes. 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 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" [
<!ENTITY % SVG.view.extra.content  "| customNS:customElement" >
<!ATTLIST %SVG.view.qname;
  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 xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="8cm" height="4cm">
  <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

23.6.1 Interface SVGForeignObjectElement

The SVGForeignObjectElement interface corresponds to the 'foreignObject' element.

```
interface SVGForeignObjectElement : SVGElement,
                                    SVGTests,
                                    SVGLangSpace,
                                    SVGExternalResourcesRequired,
                                    SVGStylable,
                                    SVGTransformable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
};
```

Attributes:

- **x** (readonly SVGAnimatedLength)

  Corresponds to attribute 'x' on the given 'foreignObject' element.

- **y** (readonly SVGAnimatedLength)

  Corresponds to attribute 'y' on the given 'foreignObject' element.

- **width** (readonly SVGAnimatedLength)

  Corresponds to attribute 'width' on the given 'foreignObject' element.

- **height** (readonly SVGAnimatedLength)

  Corresponds to attribute 'height' on the given 'foreignObject' element.

Appendix A: Document Type Definition

Contents

This appendix is normative.

A.1 Introduction

This appendix defines a DTD for SVG 1.1, which is used as part of determining whether a given document or document fragment is conforming. See Conformance Criteria for details on how the DTD is to be used in this regard. Note in particular that simply validating a given XML document against this DTD cannot definitively, by itself, determine conformance to this specification.

If errors are found in this DTD, then they will be listed in the **SVG 1.1 Second Edition errata**. A dated version of the flattened DTD will always be available from **a URI to be announced**.

A.2 Modularization

The modularization of SVG included here is a decomposition of SVG 1.0 [SVG10] and errata into a collection of abstract modules that provide specific units of functionality. These modules may be combined with each other and with modules defined in other specifications (such as XHTML) to create SVG subset and extension document types that qualify as members of the SVG family of document types. See Conformance for a description of SVG family documents, and *An XHTML + MathML + SVG Profile* [XHTMLplusMathMLplusSVG] for a profile that combines XHTML, MathML and SVG.

Each major section of the SVG specification corresponds to a module named after that section, e.g. "Text Module" or "Basic Structure Module". A module without the "Basic" prefix implies that the module includes the complete set of elements and attributes, with no restrictions, from the corresponding section of the specification. If there is a need to provide a subset of the functionality of the complete module, then a Basic module is created with the "Basic" prefix added to the name of the complete module. For example, the "Basic Text Module" is a subset of the "Text Module".

It is an error for a profile of SVG 1.1 to include both the complete module and its basic subset (e.g. the "Text Module" and the "Basic Text Module").

A.2.1 Element and attribute collections

Most modules define a named collection of elements or attributes. These collections are used as a shorthand when describing the set of attributes allowed on a particular element (e.g. the "Style" attribute collection) or the set of elements allowed as children of a particular element (e.g. the "Shape" element collection). All collections have names that begin with an uppercase character.

When defining a profile, it is assumed that all the element and attribute collections are defined to be empty. That way, a module can redefine the collection as it is included in the profile, adding elements or attributes to make them available within the profile. Therefore, it is not a mistake to refer to an element or attribute collection from a module that is not included in the profile, it simply means that collection is empty.

The exception to this is the collection Presentation.attrib, which is the union of all the presentation attribute collections (i.e. all the attribute collections with the string "Presentation" in their name). Presentation.attrib is not defined in any module, but it exists in every profile.

A subset module (i.e. a Basic module) may define a different named collection from a superset module. Since it is an error to include a subset and superset module of the same group in a profile, all attribute and element collections will either be defined once by the module that includes them, or will have their default empty value (again, with the exception of Presentation.attrib which is not defined by any module).

A.2.2 Profiling the SVG specification

The modularization of SVG 1.1 allows profiles to be described by listing the SVG modules they allow and possibly a small number of restrictions or extensions on the elements provided by those modules.

The "Full" profile of SVG 1.1 is the collection of all the complete modules listed in this specification (i.e., every module that is not a subset module).

When applied to conformance, the unqualified term "SVG" implies the "Full" profile of SVG 1.1 defined by this specification. If an implementation does not implement the Full profile, it must state either the profile to which it conforms, or that it implements a subset of SVG.

A.2.3 Practical considerations

DTD-based modularization has proven to be an unwieldy method of defining composable XML languages, due to the inherent inability to describe certain complex content models in DTDs as well as their being agnostic with respect to XML namespaces. While the SVG 1.1 DTD is provided in a modularized form, it is recommended that alternate technologies such as Namespace-based Validation Dispatch Language [NVDL] be used to accomplish XML language composition instead.

A.3 SVG 1.1 module definitions and DTD implementations

This section contains the formal definition of each of the SVG abstract modules as a DTD module. Any element and attribute collections defined by the module are also listed.

A.3.1 Modular Framework Module

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Modular Framework Module ..................................... -->
<!-- file: svg-framework.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-framework.mod,v 1.2 2009/07/28 06:37:42 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Modular Framework//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-framework.mod"

     ..................................................................... -->

<!-- Modular Framework

     This module instantiates the modules needed to support the SVG
     modularization model, including:

        + Datatypes
        + Qualified Name
        + Document Model
        + Attribute Collection
-->

<!ENTITY % svg-datatypes.module "INCLUDE" >
<![%svg-datatypes.module;[
<!ENTITY % svg-datatypes.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Datatypes//EN"
           "svg-datatypes.mod" >
%svg-datatypes.mod;]]>

<!ENTITY % svg-qname.module "INCLUDE" >
<![%svg-qname.module;[
<!ENTITY % svg-qname.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Qualified Name//EN"
           "svg-qname.mod" >
%svg-qname.mod;]]>

<!ENTITY % svg-model.module "INCLUDE" >
<![%svg-model.module;[
<!-- instantiate the Document Model declared in the DTD driver -->
%svg-model.mod;]]>

<!ENTITY % svg-attribs.module "INCLUDE" >
<![%svg-attribs.module;[
<!-- instantiate the Attribute Collection declared in the DTD driver -->
%svg-attribs.mod;]]>

<!-- end of svg-framework.mod -->
```

A.3.2 Datatypes Module

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Datatypes Module ............................................. -->
<!-- file: svg-datatypes.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
```

```
     Revision: $Id: svg-datatypes.mod,v 1.2 2009/05/23 12:55:40 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Datatypes//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-datatypes.mod"

     ..................................................................... -->

<!-- Datatypes

     This module declares common data types for properties and attributes.
-->

<!-- feature specification -->
<!ENTITY % Boolean.datatype "( false | true )" >

<!-- 'clip-rule' or 'fill-rule' property/attribute value -->
<!ENTITY % ClipFillRule.datatype "( nonzero | evenodd | inherit )" >

<!-- media type, as per [RFC2045] -->
<!ENTITY % ContentType.datatype "CDATA" >

<!-- a <coordinate> -->
<!ENTITY % Coordinate.datatype "CDATA" >

<!-- a list of <coordinate>s -->
<!ENTITY % Coordinates.datatype "CDATA" >

<!-- a <color> value -->
<!ENTITY % Color.datatype "CDATA" >

<!-- a <integer> -->
<!ENTITY % Integer.datatype "CDATA" >

<!-- a language code, as per [BCP47] -->
<!ENTITY % LanguageCode.datatype "NMTOKEN" >

<!-- comma-separated list of language codes, as per [BCP47] -->
<!ENTITY % LanguageCodes.datatype "CDATA" >

<!-- a <length> -->
<!ENTITY % Length.datatype "CDATA" >

<!-- a list of <length>s -->
<!ENTITY % Lengths.datatype "CDATA" >

<!-- a <number> -->
<!ENTITY % Number.datatype "CDATA" >

<!-- a list of <number>s -->
<!ENTITY % Numbers.datatype "CDATA" >

<!-- opacity value (e.g., <number>) -->
<!ENTITY % OpacityValue.datatype "CDATA" >

<!-- a path data specification -->
<!ENTITY % PathData.datatype "CDATA" >

<!-- 'preserveAspectRatio' attribute specification -->
<!ENTITY % PreserveAspectRatioSpec.datatype "CDATA" >

<!-- script expression -->
<!ENTITY % Script.datatype "CDATA" >

<!-- An SVG color value (RGB plus optional ICC) -->
<!ENTITY % SVGColor.datatype "CDATA" >

<!-- arbitrary text string -->
<!ENTITY % Text.datatype "CDATA" >

<!-- list of transforms -->
<!ENTITY % TransformList.datatype "CDATA" >

<!-- a Uniform Resource Identifier, see [URI] -->
<!ENTITY % URI.datatype "CDATA" >
```

```
<!-- 'viewBox' attribute specification -->
<!ENTITY % ViewBoxSpec.datatype "CDATA" >

<!-- end of svg-datatypes.mod -->
```

A.3.3 Qualified Name Module

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Qualified Name Module ....................................... -->
<!-- file: svg-qname.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-qname.mod,v 1.2 2009/05/06 05:34:40 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Qualified Name//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-qname.mod"

     ..................................................................... -->

<!-- Qualified Name

     This module is contained in two parts, labeled Section 'A' and 'B':

        Section A declares parameter entities to support namespace-
        qualified names, namespace declarations, and name prefixing
        for SVG and extensions.

        Section B declares parameter entities used to provide
        namespace-qualified names for all SVG element types:
-->

<!-- Section A: SVG XML Namespace Framework :::::::::::::::::::::: -->

<!-- 1. Declare a %SVG.prefixed; conditional section keyword, used
        to activate namespace prefixing. The default value should
        inherit '%NS.prefixed;' from the DTD driver, so that unless
        overridden, the default behaviour follows the overall DTD
        prefixing scheme.
-->
<!ENTITY % NS.prefixed "IGNORE" >
<!ENTITY % SVG.prefixed "%NS.prefixed;" >

<!-- 2. Declare a parameter entity (eg., %SVG.xmlns;) containing
        the URI reference used to identify the SVG namespace:
-->
<!ENTITY % SVG.xmlns "http://www.w3.org/2000/svg" >
<!ENTITY % XLINK.xmlns "http://www.w3.org/1999/xlink" >

<!-- 3. Declare parameter entities (eg., %SVG.prefix;) containing
        the default namespace prefix string(s) to use when prefixing
        is enabled. This may be overridden in the DTD driver or the
        internal subset of an document instance. If no default prefix
        is desired, this may be declared as an empty string.
-->
<!ENTITY % SVG.prefix "" >
<!ENTITY % XLINK.prefix "xlink" >

<!-- 4. Declare parameter entities (eg., %SVG.pfx;) containing the
        colonized prefix(es) (eg., '%SVG.prefix;:') used when
        prefixing is active, an empty string when it is not.
-->
<![%SVG.prefixed;[
<!ENTITY % SVG.pfx "%SVG.prefix;:" >
]]>
<!ENTITY % SVG.pfx "" >
<!ENTITY % XLINK.pfx "%XLINK.prefix;:" >

<!-- 5. The parameter entity %SVG.xmlns.extra.attrib; may be
        redeclared to contain any non-SVG namespace declaration
        attributes for namespaces embedded in SVG. The default
        is an empty string.
```

```
-->
<!ENTITY % SVG.xmlns.extra.attrib "" >

<!-- Declare a parameter entity XLINK.xmlns.attrib containing
     the XML Namespace declarations for XLink.
-->
<!ENTITY % XLINK.xmlns.attrib
     "xmlns:%XLINK.prefix; %URI.datatype; #FIXED '%XLINK.xmlns;'"
>

<!-- Declare a parameter entity %NS.decl.attrib; containing
     all XML Namespace declarations used in the DTD, plus the
     xmlns declaration for SVG, its form dependent on whether
     prefixing is active.
-->
<![%SVG.prefixed;[
<!ENTITY % NS.decl.attrib
    "xmlns:%SVG.prefix; %URI.datatype; #FIXED '%SVG.xmlns;'
     %XLINK.xmlns.attrib;
     %SVG.xmlns.extra.attrib;"
>
]]>
<!ENTITY % NS.decl.attrib
    "%XLINK.xmlns.attrib;
     %SVG.xmlns.extra.attrib;"
>

<!-- Declare a parameter entity %SVG.xmlns.attrib; containing
     all XML namespace declaration attributes used by SVG,
     including a default xmlns attribute when prefixing is
     inactive.
-->
<![%SVG.prefixed;[
<!ENTITY % SVG.xmlns.attrib
     "%NS.decl.attrib;"
>
]]>
<!ENTITY % SVG.xmlns.attrib
     "xmlns %URI.datatype; #FIXED '%SVG.xmlns;'
      %XLINK.xmlns.attrib;"
>

<!-- Section B: SVG Qualified Names :::::::::::::::::::::::::::::: -->

<!-- 6. This section declares parameter entities used to provide
        namespace-qualified names for all SVG element types.
-->

<!-- module: svg-structure.mod ........................ -->

<!ENTITY % SVG.svg.qname "%SVG.pfx;svg" >
<!ENTITY % SVG.g.qname "%SVG.pfx;g" >
<!ENTITY % SVG.defs.qname "%SVG.pfx;defs" >
<!ENTITY % SVG.desc.qname "%SVG.pfx;desc" >
<!ENTITY % SVG.title.qname "%SVG.pfx;title" >
<!ENTITY % SVG.metadata.qname "%SVG.pfx;metadata" >
<!ENTITY % SVG.symbol.qname "%SVG.pfx;symbol" >
<!ENTITY % SVG.use.qname "%SVG.pfx;use" >

<!-- module: svg-conditional.mod ...................... -->

<!ENTITY % SVG.switch.qname "%SVG.pfx;switch" >

<!-- module: svg-image.mod ............................ -->

<!ENTITY % SVG.image.qname "%SVG.pfx;image" >

<!-- module: svg-style.mod ............................ -->

<!ENTITY % SVG.style.qname "%SVG.pfx;style" >

<!-- module: svg-shape.mod ............................ -->

<!ENTITY % SVG.path.qname "%SVG.pfx;path" >
<!ENTITY % SVG.rect.qname "%SVG.pfx;rect" >
<!ENTITY % SVG.circle.qname "%SVG.pfx;circle" >
<!ENTITY % SVG.line.qname "%SVG.pfx;line" >
```

```
<!ENTITY % SVG.ellipse.qname "%SVG.pfx;ellipse" >
<!ENTITY % SVG.polyline.qname "%SVG.pfx;polyline" >
<!ENTITY % SVG.polygon.qname "%SVG.pfx;polygon" >

<!-- module: svg-text.mod ............................. -->

<!ENTITY % SVG.text.qname "%SVG.pfx;text" >
<!ENTITY % SVG.tspan.qname "%SVG.pfx;tspan" >
<!ENTITY % SVG.tref.qname "%SVG.pfx;tref" >
<!ENTITY % SVG.textPath.qname "%SVG.pfx;textPath" >
<!ENTITY % SVG.altGlyph.qname "%SVG.pfx;altGlyph" >
<!ENTITY % SVG.altGlyphDef.qname "%SVG.pfx;altGlyphDef" >
<!ENTITY % SVG.altGlyphItem.qname "%SVG.pfx;altGlyphItem" >
<!ENTITY % SVG.glyphRef.qname "%SVG.pfx;glyphRef" >

<!-- module: svg-marker.mod ........................... -->

<!ENTITY % SVG.marker.qname "%SVG.pfx;marker" >

<!-- module: svg-profile.mod .......................... -->

<!ENTITY % SVG.color-profile.qname "%SVG.pfx;color-profile" >

<!-- module: svg-gradient.mod ......................... -->

<!ENTITY % SVG.linearGradient.qname "%SVG.pfx;linearGradient" >
<!ENTITY % SVG.radialGradient.qname "%SVG.pfx;radialGradient" >
<!ENTITY % SVG.stop.qname "%SVG.pfx;stop" >

<!-- module: svg-pattern.mod .......................... -->

<!ENTITY % SVG.pattern.qname "%SVG.pfx;pattern" >

<!-- module: svg-clip.mod ............................. -->

<!ENTITY % SVG.clipPath.qname "%SVG.pfx;clipPath" >

<!-- module: svg-mask.mod ............................. -->

<!ENTITY % SVG.mask.qname "%SVG.pfx;mask" >

<!-- module: svg-filter.mod ........................... -->

<!ENTITY % SVG.filter.qname "%SVG.pfx;filter" >
<!ENTITY % SVG.feBlend.qname "%SVG.pfx;feBlend" >
<!ENTITY % SVG.feColorMatrix.qname "%SVG.pfx;feColorMatrix" >
<!ENTITY % SVG.feComponentTransfer.qname "%SVG.pfx;feComponentTransfer" >
<!ENTITY % SVG.feComposite.qname "%SVG.pfx;feComposite" >
<!ENTITY % SVG.feConvolveMatrix.qname "%SVG.pfx;feConvolveMatrix" >
<!ENTITY % SVG.feDiffuseLighting.qname "%SVG.pfx;feDiffuseLighting" >
<!ENTITY % SVG.feDisplacementMap.qname "%SVG.pfx;feDisplacementMap" >
<!ENTITY % SVG.feFlood.qname "%SVG.pfx;feFlood" >
<!ENTITY % SVG.feGaussianBlur.qname "%SVG.pfx;feGaussianBlur" >
<!ENTITY % SVG.feImage.qname "%SVG.pfx;feImage" >
<!ENTITY % SVG.feMerge.qname "%SVG.pfx;feMerge" >
<!ENTITY % SVG.feMergeNode.qname "%SVG.pfx;feMergeNode" >
<!ENTITY % SVG.feMorphology.qname "%SVG.pfx;feMorphology" >
<!ENTITY % SVG.feOffset.qname "%SVG.pfx;feOffset" >
<!ENTITY % SVG.feSpecularLighting.qname "%SVG.pfx;feSpecularLighting" >
<!ENTITY % SVG.feTile.qname "%SVG.pfx;feTile" >
<!ENTITY % SVG.feTurbulence.qname "%SVG.pfx;feTurbulence" >
<!ENTITY % SVG.feDistantLight.qname "%SVG.pfx;feDistantLight" >
<!ENTITY % SVG.fePointLight.qname "%SVG.pfx;fePointLight" >
<!ENTITY % SVG.feSpotLight.qname "%SVG.pfx;feSpotLight" >
<!ENTITY % SVG.feFuncR.qname "%SVG.pfx;feFuncR" >
<!ENTITY % SVG.feFuncG.qname "%SVG.pfx;feFuncG" >
<!ENTITY % SVG.feFuncB.qname "%SVG.pfx;feFuncB" >
<!ENTITY % SVG.feFuncA.qname "%SVG.pfx;feFuncA" >

<!-- module: svg-cursor.mod ........................... -->

<!ENTITY % SVG.cursor.qname "%SVG.pfx;cursor" >

<!-- module: svg-hyperlink.mod ........................ -->

<!ENTITY % SVG.a.qname "%SVG.pfx;a" >
```

```
<!-- module: svg-view.mod ............................ -->

<!ENTITY % SVG.view.qname "%SVG.pfx;view" >

<!-- module: svg-script.mod ........................... -->

<!ENTITY % SVG.script.qname "%SVG.pfx;script" >

<!-- module: svg-animation.mod ........................ -->

<!ENTITY % SVG.animate.qname "%SVG.pfx;animate" >
<!ENTITY % SVG.set.qname "%SVG.pfx;set" >
<!ENTITY % SVG.animateMotion.qname "%SVG.pfx;animateMotion" >
<!ENTITY % SVG.animateColor.qname "%SVG.pfx;animateColor" >
<!ENTITY % SVG.animateTransform.qname "%SVG.pfx;animateTransform" >
<!ENTITY % SVG.mpath.qname "%SVG.pfx;mpath" >

<!-- module: svg-font.mod ............................. -->

<!ENTITY % SVG.font.qname "%SVG.pfx;font" >
<!ENTITY % SVG.font-face.qname "%SVG.pfx;font-face" >
<!ENTITY % SVG.glyph.qname "%SVG.pfx;glyph" >
<!ENTITY % SVG.missing-glyph.qname "%SVG.pfx;missing-glyph" >
<!ENTITY % SVG.hkern.qname "%SVG.pfx;hkern" >
<!ENTITY % SVG.vkern.qname "%SVG.pfx;vkern" >
<!ENTITY % SVG.font-face-src.qname "%SVG.pfx;font-face-src" >
<!ENTITY % SVG.font-face-uri.qname "%SVG.pfx;font-face-uri" >
<!ENTITY % SVG.font-face-format.qname "%SVG.pfx;font-face-format" >
<!ENTITY % SVG.font-face-name.qname "%SVG.pfx;font-face-name" >

<!-- module: svg-extensibility.mod ..................... -->

<!ENTITY % SVG.foreignObject.qname "%SVG.pfx;foreignObject" >

<!-- end of svg-qname.mod -->
```

A.3.4 Core Attribute Module

The Core Attribute Module defines the attribute collection Core.attrib that is the core set of attributes that can be present on any element.

Collection name	Attributes in collection
Core.attrib	id, xml:base, xml:lang, xml:space

```
<!-- ................................................................... -->
<!-- SVG 1.1 Core Attribute Module ..................................... -->
<!-- file: svg-core-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-core-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Core Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-core-attrib.mod"

     ................................................................... -->

<!-- Core Attribute

        id, xml:base, xml:lang, xml:space

     This module defines the core set of attributes that can be present on
     any element.
-->

<!ENTITY % SVG.id.attrib
```

```
    "id ID #IMPLIED"
>

<!ENTITY % SVG.base.attrib
    "xml:base %URI.datatype; #IMPLIED"
>

<!ENTITY % SVG.lang.attrib
    "xml:lang %LanguageCode.datatype; #IMPLIED"
>

<!ENTITY % SVG.space.attrib
    "xml:space ( default | preserve ) #IMPLIED"
>

<!ENTITY % SVG.Core.extra.attrib "" >

<!ENTITY % SVG.Core.attrib
    "%SVG.id.attrib;
     %SVG.base.attrib;
     %SVG.lang.attrib;
     %SVG.space.attrib;
     %SVG.Core.extra.attrib;"
>

<!-- end of svg-core-attrib.mod -->
```

A.3.5 Container Attribute Module

The Container Attribute Module defines the Container.attrib attribute collection.

Collection name	Attributes in collection
Container.attrib	enable-background

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Container Attribute Module ................................. -->
<!-- file: svg-container-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-container-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Container Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-container-attrib.mod"

     ..................................................................... -->

<!-- Container Attribute

        enable-background

     This module defines the Container attribute set.
-->

<!-- 'enable-background' property/attribute value (e.g., 'new', 'accumulate') -->
<!ENTITY % EnableBackgroundValue.datatype "CDATA" >

<!ENTITY % SVG.enable-background.attrib
    "enable-background %EnableBackgroundValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.Container.extra.attrib "" >

<!ENTITY % SVG.Container.attrib
    "%SVG.enable-background.attrib;
     %SVG.Container.extra.attrib;"
>
```

```
<!-- end of svg-container-attrib.mod -->
```

A.3.6 Viewport Attribute Module

The Container Attribute Module defines the Container.attrib attribute collection.

Collection name	Attributes in collection
Viewport.attrib	clip, overflow

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Viewport Attribute Module ..................................... -->
<!-- file: svg-viewport-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-viewport-attrib.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Viewport Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-viewport-attrib.mod"

     ..................................................................... -->

<!-- Viewport Attribute

        clip, overflow

     This module defines the Viewport attribute set.
-->

<!-- 'clip' property/attribute value (e.g., 'auto', rect(...)) -->
<!ENTITY % ClipValue.datatype "CDATA" >

<!ENTITY % SVG.clip.attrib
    "clip %ClipValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.overflow.attrib
    "overflow ( visible | hidden | scroll | auto | inherit ) #IMPLIED"
>

<!ENTITY % SVG.Viewport.extra.attrib "" >

<!ENTITY % SVG.Viewport.attrib
    "%SVG.clip.attrib;
     %SVG.overflow.attrib;
     %SVG.Viewport.extra.attrib;"
>

<!-- end of svg-viewport-attrib.mod -->
```

A.3.7 Paint Attribute Module

The Paint Attribute Module defines the Paint.attrib attribute collection.

Collection name	Attributes in collection
Paint.attrib	color, fill, fill-rule, stroke, stroke-dasharray, stroke-dashoffset, stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-width, color-interpolation, color-rendering

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Paint Attribute Module ..................................... -->
<!-- file: svg-paint-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-paint-attrib.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Paint Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-paint-attrib.mod"

     ..................................................................... -->

<!-- Paint Attribute

        fill, fill-rule, stroke, stroke-dasharray, stroke-dashoffset,
        stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-width, color,
        color-interpolation, color-rendering

     This module defines the Paint and Color attribute sets.
-->

<!-- a 'fill' or 'stroke' property/attribute value: <paint> -->
<!ENTITY % Paint.datatype "CDATA" >

<!-- 'stroke-dasharray' property/attribute value (e.g., 'none', list of <number>s) -->
<!ENTITY % StrokeDashArrayValue.datatype "CDATA" >

<!-- 'stroke-dashoffset' property/attribute value (e.g., 'none', <legnth>) -->
<!ENTITY % StrokeDashOffsetValue.datatype "CDATA" >

<!-- 'stroke-miterlimit' property/attribute value (e.g., <number>) -->
<!ENTITY % StrokeMiterLimitValue.datatype "CDATA" >

<!-- 'stroke-width' property/attribute value (e.g., <length>) -->
<!ENTITY % StrokeWidthValue.datatype "CDATA" >

<!ENTITY % SVG.fill.attrib
    "fill %Paint.datatype; #IMPLIED"
>

<!ENTITY % SVG.fill-rule.attrib
    "fill-rule %ClipFillRule.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke.attrib
    "stroke %Paint.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-dasharray.attrib
    "stroke-dasharray %StrokeDashArrayValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-dashoffset.attrib
    "stroke-dashoffset %StrokeDashOffsetValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-linecap.attrib
    "stroke-linecap ( butt | round | square | inherit ) #IMPLIED"
>

<!ENTITY % SVG.stroke-linejoin.attrib
    "stroke-linejoin ( miter | round | bevel | inherit ) #IMPLIED"
>
```

```
<!ENTITY % SVG.stroke-miterlimit.attrib
    "stroke-miterlimit %StrokeMiterLimitValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-width.attrib
    "stroke-width %StrokeWidthValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.Paint.extra.attrib "" >

<!ENTITY % SVG.Paint.attrib
    "%SVG.fill.attrib;
     %SVG.fill-rule.attrib;
     %SVG.stroke.attrib;
     %SVG.stroke-dasharray.attrib;
     %SVG.stroke-dashoffset.attrib;
     %SVG.stroke-linecap.attrib;
     %SVG.stroke-linejoin.attrib;
     %SVG.stroke-miterlimit.attrib;
     %SVG.stroke-width.attrib;
     %SVG.Paint.extra.attrib;"
>

<!ENTITY % SVG.color.attrib
    "color %Color.datatype; #IMPLIED"
>

<!ENTITY % SVG.color-interpolation.attrib
    "color-interpolation ( auto | sRGB | linearRGB | inherit ) #IMPLIED"
>

<!ENTITY % SVG.color-rendering.attrib
    "color-rendering ( auto | optimizeSpeed | optimizeQuality | inherit )
                     #IMPLIED"
>

<!ENTITY % SVG.Color.extra.attrib "" >

<!ENTITY % SVG.Color.attrib
    "%SVG.color.attrib;
     %SVG.color-interpolation.attrib;
     %SVG.color-rendering.attrib;
     %SVG.Color.extra.attrib;"
>

<!-- end of svg-paint-attrib.mod -->
```

A.3.8 Basic Paint Attribute Module

The Basic Paint Attribute Module defines the Paint.attrib attribute collection.

Collection name	Attributes in collection
Paint.attrib	color, fill, fill-rule, stroke, stroke-dasharray, stroke-dashoffset, stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-width, color-rendering

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Basic Paint Attribute Module ................................ -->
<!-- file: svg-basic-paint-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-paint-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Basic Paint Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-paint-attrib.mod"
```

```
     ..... -->

<!-- Basic Paint Attribute

        fill, fill-rule, stroke, stroke-dasharray, stroke-dashoffset,
        stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-width, color,
        color-rendering

     This module defines the Paint and Color attribute sets.
-->

<!-- a 'fill' or 'stroke' property/attribute value: <paint> -->
<!ENTITY % Paint.datatype "CDATA" >

<!-- 'stroke-dasharray' property/attribute value (e.g., 'none', list of <number>s) -->
<!ENTITY % StrokeDashArrayValue.datatype "CDATA" >

<!-- 'stroke-dashoffset' property/attribute value (e.g., 'none', <legnth>) -->
<!ENTITY % StrokeDashOffsetValue.datatype "CDATA" >

<!-- 'stroke-miterlimit' property/attribute value (e.g., <number>) -->
<!ENTITY % StrokeMiterLimitValue.datatype "CDATA" >

<!-- 'stroke-width' property/attribute value (e.g., <length>) -->
<!ENTITY % StrokeWidthValue.datatype "CDATA" >

<!ENTITY % SVG.fill.attrib
    "fill %Paint.datatype; #IMPLIED"
>

<!ENTITY % SVG.fill-rule.attrib
    "fill-rule %ClipFillRule.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke.attrib
    "stroke %Paint.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-dasharray.attrib
    "stroke-dasharray %StrokeDashArrayValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-dashoffset.attrib
    "stroke-dashoffset %StrokeDashOffsetValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-linecap.attrib
    "stroke-linecap ( butt | round | square | inherit ) #IMPLIED"
>

<!ENTITY % SVG.stroke-linejoin.attrib
    "stroke-linejoin ( miter | round | bevel | inherit ) #IMPLIED"
>

<!ENTITY % SVG.stroke-miterlimit.attrib
    "stroke-miterlimit %StrokeMiterLimitValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-width.attrib
    "stroke-width %StrokeWidthValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.Paint.extra.attrib "" >

<!ENTITY % SVG.Paint.attrib
    "%SVG.fill.attrib;
     %SVG.fill-rule.attrib;
     %SVG.stroke.attrib;
     %SVG.stroke-dasharray.attrib;
     %SVG.stroke-dashoffset.attrib;
     %SVG.stroke-linecap.attrib;
     %SVG.stroke-linejoin.attrib;
     %SVG.stroke-miterlimit.attrib;
     %SVG.stroke-width.attrib;
     %SVG.Paint.extra.attrib;"
>
```

```
<!ENTITY % SVG.color.attrib
    "color %Color.datatype; #IMPLIED"
>

<!ENTITY % SVG.color-rendering.attrib
    "color-rendering ( auto | optimizeSpeed | optimizeQuality | inherit )
                     #IMPLIED"
>

<!ENTITY % SVG.Color.extra.attrib "" >

<!ENTITY % SVG.Color.attrib
    "%SVG.color.attrib;
     %SVG.color-rendering.attrib;
     %SVG.Color.extra.attrib;"
>

<!-- end of svg-basic-paint-attrib.mod -->
```

A.3.9 Paint Opacity Attribute Module

The Paint Opacity Attribute Module defines the Opacity.attrib attribute collection.

Collection name	Attributes in collection
Opacity.attrib	opacity, stroke-opacity, fill-opacity

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Paint Opacity Attribute Module ............................... -->
<!-- file: svg-opacity-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-opacity-attrib.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Paint Opacity Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-opacity-attrib.mod"

     ..................................................................... -->

<!-- Paint Opacity Attribute

        opacity, fill-opacity, stroke-opacity

     This module defines the Opacity attribute set.
-->

<!ENTITY % SVG.opacity.attrib
    "opacity %OpacityValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.fill-opacity.attrib
    "fill-opacity %OpacityValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.stroke-opacity.attrib
    "stroke-opacity %OpacityValue.datatype; #IMPLIED"
>

<!ENTITY % SVG.Opacity.extra.attrib "" >

<!ENTITY % SVG.Opacity.attrib
    "%SVG.opacity.attrib;
     %SVG.fill-opacity.attrib;
     %SVG.stroke-opacity.attrib;
     %SVG.Opacity.extra.attrib;"
>
```

```
<!-- end of svg-opacity-attrib.mod -->
```

A.3.10 Graphics Attribute Module

The Graphics Attribute Module defines the Graphics.attrib attribute collection.

Collection name	Attributes in collection
Graphics.attrib	display, image-rendering, pointer-events, shape-rendering, text-rendering, visibility

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Graphics Attribute Module ..................................... -->
<!-- file: svg-graphics-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-graphics-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Graphics Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-graphics-attrib.mod"

     ..................................................................... -->

<!-- Graphics Attribute

        display, image-rendering, pointer-events, shape-rendering,
        text-rendering, visibility

     This module defines the Graphics attribute set.
-->

<!ENTITY % SVG.display.attrib
    "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"
>

<!ENTITY % SVG.image-rendering.attrib
    "image-rendering ( auto | optimizeSpeed | optimizeQuality | inherit )
                       #IMPLIED"
>

<!ENTITY % SVG.pointer-events.attrib
    "pointer-events ( visiblePainted | visibleFill | visibleStroke | visible |
                      painted | fill | stroke | all | none | inherit )
                      #IMPLIED"
>

<!ENTITY % SVG.shape-rendering.attrib
    "shape-rendering ( auto | optimizeSpeed | crispEdges | geometricPrecision |
                       inherit ) #IMPLIED"
>

<!ENTITY % SVG.text-rendering.attrib
    "text-rendering ( auto | optimizeSpeed | optimizeLegibility |
                      geometricPrecision | inherit ) #IMPLIED"
>

<!ENTITY % SVG.visibility.attrib
    "visibility ( visible | hidden | inherit ) #IMPLIED"
>

<!ENTITY % SVG.Graphics.extra.attrib "" >

<!ENTITY % SVG.Graphics.attrib
```

```
    "%SVG.display.attrib;
     %SVG.image-rendering.attrib;
     %SVG.pointer-events.attrib;
     %SVG.shape-rendering.attrib;
     %SVG.text-rendering.attrib;
     %SVG.visibility.attrib;
     %SVG.Graphics.extra.attrib;"
>

<!-- end of svg-graphics-attrib.mod -->
```

A.3.11 Basic Graphics Attribute Module

The Basic Graphics Attribute Module defines the Graphics.attrib attribute collection.

Collection name	Attributes in collection
Graphics.attrib	display, visibility

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Basic Graphics Attribute Module ............................. -->
<!-- file: svg-basic-graphics-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-graphics-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Basic Graphics Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-graphics-attrib.mod"

     ..................................................................... -->

<!-- Basic Graphics Attribute

        display, visibility

     This module defines the Graphics attribute set.
-->

<!ENTITY % SVG.display.attrib
    "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"
>

<!ENTITY % SVG.visibility.attrib
    "visibility ( visible | hidden | inherit ) #IMPLIED"
>

<!ENTITY % SVG.Graphics.extra.attrib "" >

<!ENTITY % SVG.Graphics.attrib
    "%SVG.display.attrib;
     %SVG.visibility.attrib;
     %SVG.Graphics.extra.attrib;"
>

<!-- end of svg-basic-graphics-attrib.mod -->
```

A.3.12 Document Events Attribute Module

The Document Events Attribute Module defines the DocumentEvents.attrib attribute collection.

Collection name	Attributes in collection
DocumentEvents.attrib	onunload, onabort, onerror, onresize, onscroll, onzoom

```
<!-- ....................................................................... -->
<!-- SVG 1.1 Document Events Attribute Module ............................. -->
<!-- file: svg-docevents-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-docevents-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Document Events Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-docevents-attrib.mod"

     ....................................................................... -->

<!-- Document Events Attribute

        onunload, onabort, onerror, onresize, onscroll, onzoom

     This module defines the DocumentEvents attribute set.
-->

<!ENTITY % SVG.onunload.attrib
    "onunload %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onabort.attrib
    "onabort %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onerror.attrib
    "onerror %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onresize.attrib
    "onresize %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onscroll.attrib
    "onscroll %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onzoom.attrib
    "onzoom %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.DocumentEvents.extra.attrib "" >

<!ENTITY % SVG.DocumentEvents.attrib
    "%SVG.onunload.attrib;
     %SVG.onabort.attrib;
     %SVG.onerror.attrib;
     %SVG.onresize.attrib;
     %SVG.onscroll.attrib;
     %SVG.onzoom.attrib;
     %SVG.DocumentEvents.extra.attrib;"
>

<!-- end of svg-docevents-attrib.mod -->
```

A.3.13 Graphical Element Events Attribute Module

The Graphical Events Attribute Module defines the GraphicalEvents.attrib attribute collection.

Collection name	Attributes in collection
GraphicalEvents.attrib	onfocusin, onfocusout, onactivate, onclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onload

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Graphical Element Events Attribute Module ..................... -->
<!-- file: svg-graphevents-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-graphevents-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Graphical Element Events Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-graphevents-attrib.mod"

     ..................................................................... -->

<!-- Graphical Element Events Attribute

        onfocusin, onfocusout, onactivate, onclick, onmousedown, onmouseup,
        onmouseover, onmousemove, onmouseout, onload

     This module defines the GraphicalEvents attribute set.
-->

<!ENTITY % SVG.onfocusin.attrib
    "onfocusin %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onfocusout.attrib
    "onfocusout %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onactivate.attrib
    "onactivate %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onclick.attrib
    "onclick %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onmousedown.attrib
    "onmousedown %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onmouseup.attrib
    "onmouseup %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onmouseover.attrib
    "onmouseover %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onmousemove.attrib
    "onmousemove %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onmouseout.attrib
    "onmouseout %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onload.attrib
    "onload %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.GraphicalEvents.extra.attrib "" >

<!ENTITY % SVG.GraphicalEvents.attrib
    "%SVG.onfocusin.attrib;
     %SVG.onfocusout.attrib;
     %SVG.onactivate.attrib;
```

```
     %SVG.onclick.attrib;
     %SVG.onmousedown.attrib;
     %SVG.onmouseup.attrib;
     %SVG.onmouseover.attrib;
     %SVG.onmousemove.attrib;
     %SVG.onmouseout.attrib;
     %SVG.onload.attrib;
     %SVG.GraphicalEvents.extra.attrib;"
>

<!-- end of svg-graphevents-attrib.mod -->
```

A.3.14 Animation Events Attribute Module

The Animation Events Attribute Module defines the AnimationEvents.attrib attribute collection.

Collection name	Attributes in collection
AnimationEvents.attrib	onbegin, onend, onrepeat, onload

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Animation Events Attribute Module ........................... -->
<!-- file: svg-animevents-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-animevents-attrib.mod,v 1.1 2008/12/11 06:05:54 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Animation Events Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-animevents-attrib.mod"

     ..................................................................... -->

<!-- Animation Events Attribute

        onbegin, onend, onrepeat, onload

     This module defines the AnimationEvents attribute set.
-->

<!ENTITY % SVG.onbegin.attrib
    "onbegin %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onend.attrib
    "onend %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onrepeat.attrib
    "onrepeat %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.onload.attrib
    "onload %Script.datatype; #IMPLIED"
>

<!ENTITY % SVG.AnimationEvents.extra.attrib "" >

<!ENTITY % SVG.AnimationEvents.attrib
    "%SVG.onbegin.attrib;
     %SVG.onend.attrib;
     %SVG.onrepeat.attrib;
     %SVG.onload.attrib;
     %SVG.AnimationEvents.extra.attrib;"
>

<!-- end of svg-animevents-attrib.mod -->
```

A.3.15 XLink Attribute Module

The XLink Attribute Module defines the XLink.attrib, XLinkRequired.attrib, XLinkEmbed.attrib and XLinkReplace.attrib attribute collections. These collections differ only in whether the 'xlink:href' attribute is required or what the default value for the 'xlink:show' attribute is.

Collection name	Attributes in collection
XLink.attrib	xlink:type, xlink:href, xlink:role, xlink:arcrole, xlink:title, xlink:show, xlink:actuate
XLinkRequired.attrib	xlink:type, xlink:href, xlink:role, xlink:arcrole, xlink:title, xlink:show, xlink:actuate
XLinkEmbed.attrib	xlink:type, xlink:href, xlink:role, xlink:arcrole, xlink:title, xlink:show, xlink:actuate
XLinkReplace.attrib	xlink:type, xlink:href, xlink:role, xlink:arcrole, xlink:title, xlink:show, xlink:actuate

```
<!-- ..................................................................... -->
<!-- SVG 1.1 XLink Attribute Module ..................................... -->
<!-- file: svg-xlink-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-xlink-attrib.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 XLink Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-xlink-attrib.mod"

     ..................................................................... -->

<!-- XLink Attribute

       type, href, role, arcrole, title, show, actuate

     This module defines the XLink, XLinkRequired, XLinkEmbed, and
     XLinkReplace attribute set.
-->

<!ENTITY % SVG.XLink.extra.attrib "" >

<!ENTITY % SVG.XLink.attrib
    "%XLINK.xmlns.attrib;
     %XLINK.pfx;type ( simple ) #FIXED 'simple'
     %XLINK.pfx;href %URI.datatype; #IMPLIED
     %XLINK.pfx;role %URI.datatype; #IMPLIED
     %XLINK.pfx;arcrole %URI.datatype; #IMPLIED
     %XLINK.pfx;title CDATA #IMPLIED
     %XLINK.pfx;show ( other ) 'other'
     %XLINK.pfx;actuate ( onLoad ) #FIXED 'onLoad'
     %SVG.XLink.extra.attrib;"
>

<!ENTITY % SVG.XLinkRequired.extra.attrib "" >

<!ENTITY % SVG.XLinkRequired.attrib
    "%XLINK.xmlns.attrib;
     %XLINK.pfx;type ( simple ) #FIXED 'simple'
     %XLINK.pfx;href %URI.datatype; #REQUIRED
     %XLINK.pfx;role %URI.datatype; #IMPLIED
     %XLINK.pfx;arcrole %URI.datatype; #IMPLIED
     %XLINK.pfx;title CDATA #IMPLIED
     %XLINK.pfx;show ( other ) 'other'
     %XLINK.pfx;actuate ( onLoad ) #FIXED 'onLoad'
     %SVG.XLinkRequired.extra.attrib;"
>

<!ENTITY % SVG.XLinkEmbed.extra.attrib "" >
```

```
<!ENTITY % SVG.XLinkEmbed.attrib
    "%XLINK.xmlns.attrib;
     %XLINK.pfx;type ( simple ) #FIXED 'simple'
     %XLINK.pfx;href %URI.datatype; #REQUIRED
     %XLINK.pfx;role %URI.datatype; #IMPLIED
     %XLINK.pfx;arcrole %URI.datatype; #IMPLIED
     %XLINK.pfx;title CDATA #IMPLIED
     %XLINK.pfx;show ( embed ) 'embed'
     %XLINK.pfx;actuate ( onLoad ) #FIXED 'onLoad'
     %SVG.XLinkEmbed.extra.attrib;"
>

<!ENTITY % SVG.XLinkReplace.extra.attrib "" >

<!ENTITY % SVG.XLinkReplace.attrib
    "%XLINK.xmlns.attrib;
     %XLINK.pfx;type ( simple ) #FIXED 'simple'
     %XLINK.pfx;href %URI.datatype; #REQUIRED
     %XLINK.pfx;role %URI.datatype; #IMPLIED
     %XLINK.pfx;arcrole %URI.datatype; #IMPLIED
     %XLINK.pfx;title CDATA #IMPLIED
     %XLINK.pfx;show ( new | replace ) 'replace'
     %XLINK.pfx;actuate ( onRequest ) #FIXED 'onRequest'
     %SVG.XLinkReplace.extra.attrib;"
>

<!-- end of svg-xlink-attrib.mod -->
```

A.3.16 External Resources Attribute Module

The External Resources Attribute Module defines the External.attrib attribute collection.

Collection name	Attributes in collection
External.attrib	externalResourcesRequired

```
<!-- ..................................................................... -->
<!-- SVG 1.1 External Resources Attribute Module ......................... -->
<!-- file: svg-extresources-attrib.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-extresources-attrib.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 External Resources Attribute//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-extresources-attrib.mod"

     ..................................................................... -->

<!-- External Resources Attribute

        externalResourcesRequired

     This module defines the External attribute set.
-->

<!ENTITY % SVG.externalResourcesRequired.attrib
    "externalResourcesRequired %Boolean.datatype; #IMPLIED"
>

<!ENTITY % SVG.External.extra.attrib "" >

<!ENTITY % SVG.External.attrib
    "%SVG.externalResourcesRequired.attrib;
     %SVG.External.extra.attrib;"
>
```

```
<!-- end of svg-extresources-attrib.mod -->
```

A.3.17 Structure Module

The Structure Module defines the Description.class, Structure.class and Use.class element collections.

Collection name	Elements in collection
Description.class	desc, title, metadata
Use.class	use
Structure.class	svg, g, defs, symbol, Use.class

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Structure Module ............................................ -->
<!-- file: svg-structure.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-structure.mod,v 1.4 2009/06/11 15:58:26 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Structure//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-structure.mod"

     ..................................................................... -->

<!-- Structure

        svg, g, defs, desc, title, metadata, symbol, use

     This module declares the major structural elements and their attributes.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.svg.qname "svg" >
<!ENTITY % SVG.g.qname "g" >
<!ENTITY % SVG.defs.qname "defs" >
<!ENTITY % SVG.desc.qname "desc" >
<!ENTITY % SVG.title.qname "title" >
<!ENTITY % SVG.metadata.qname "metadata" >
<!ENTITY % SVG.symbol.qname "symbol" >
<!ENTITY % SVG.use.qname "use" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
```

```
<!ENTITY % SVG.DocumentEvents.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkEmbed.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Description.class ........................... -->

<!ENTITY % SVG.Description.extra.class "" >

<!ENTITY % SVG.Description.class
    "%SVG.desc.qname; | %SVG.title.qname; | %SVG.metadata.qname;
     %SVG.Description.extra.class;"
>

<!-- SVG.Use.class ................................... -->

<!ENTITY % SVG.Use.extra.class "" >

<!ENTITY % SVG.Use.class
    "| %SVG.use.qname; %SVG.Use.extra.class;"
>

<!-- SVG.Structure.class ............................. -->

<!ENTITY % SVG.Structure.extra.class "" >

<!ENTITY % SVG.Structure.class
    "| %SVG.svg.qname; | %SVG.g.qname; | %SVG.defs.qname; | %SVG.symbol.qname;
       %SVG.Use.class; %SVG.Structure.extra.class;"
>

<!-- SVG.Presentation.attrib ......................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- svg: SVG Document Element ........................ -->

<!ENTITY % SVG.svg.extra.content "" >

<!ENTITY % SVG.svg.element "INCLUDE" >
<![%SVG.svg.element;[
<!ENTITY % SVG.svg.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.svg.extra.content; )*"
>
<!ELEMENT %SVG.svg.qname; %SVG.svg.content; >
<!-- end of SVG.svg.element -->]]>
```

```
<!ENTITY % SVG.svg.attlist "INCLUDE" >
<![%SVG.svg.attlist;[
<!ATTLIST %SVG.svg.qname;
    %SVG.xmlns.attrib;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.DocumentEvents.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
    zoomAndPan ( disable | magnify ) 'magnify'
    version %Number.datatype; #FIXED '1.1'
    baseProfile %Text.datatype; #IMPLIED
    contentScriptType %ContentType.datatype; 'application/ecmascript'
    contentStyleType %ContentType.datatype; 'text/css'
>
<!-- end of SVG.svg.attlist -->]]>

<!-- g: Group Element ................................ -->

<!ENTITY % SVG.g.extra.content "" >

<!ENTITY % SVG.g.element "INCLUDE" >
<![%SVG.g.element;[
<!ENTITY % SVG.g.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.g.extra.content; )*"
>
<!ELEMENT %SVG.g.qname; %SVG.g.content; >
<!-- end of SVG.g.element -->]]>

<!ENTITY % SVG.g.attlist "INCLUDE" >
<![%SVG.g.attlist;[
<!ATTLIST %SVG.g.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.g.attlist -->]]>

<!-- defs: Definisions Element ........................ -->

<!ENTITY % SVG.defs.extra.content "" >

<!ENTITY % SVG.defs.element "INCLUDE" >
<![%SVG.defs.element;[
<!ENTITY % SVG.defs.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.defs.extra.content; )*"
>
<!ELEMENT %SVG.defs.qname; %SVG.defs.content; >
<!-- end of SVG.defs.element -->]]>

<!ENTITY % SVG.defs.attlist "INCLUDE" >
<![%SVG.defs.attlist;[
```

```
<!ATTLIST %SVG.defs.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.defs.attlist -->]]>

<!-- desc: Description Element ........................ -->

<!ENTITY % SVG.desc.extra.content "" >

<!ENTITY % SVG.desc.element "INCLUDE" >
<![%SVG.desc.element;[
<!ENTITY % SVG.desc.content
    "( #PCDATA %SVG.desc.extra.content; )*"
>
<!ELEMENT %SVG.desc.qname; %SVG.desc.content; >
<!-- end of SVG.desc.element -->]]>

<!ENTITY % SVG.desc.attlist "INCLUDE" >
<![%SVG.desc.attlist;[
<!ATTLIST %SVG.desc.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
>
<!-- end of SVG.desc.attlist -->]]>

<!-- title: Title Element ............................. -->

<!ENTITY % SVG.title.extra.content "" >

<!ENTITY % SVG.title.element "INCLUDE" >
<![%SVG.title.element;[
<!ENTITY % SVG.title.content
    "( #PCDATA %SVG.title.extra.content; )*"
>
<!ELEMENT %SVG.title.qname; %SVG.title.content; >
<!-- end of SVG.title.element -->]]>

<!ENTITY % SVG.title.attlist "INCLUDE" >
<![%SVG.title.attlist;[
<!ATTLIST %SVG.title.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
>
<!-- end of SVG.title.attlist -->]]>

<!-- metadata: Metadata Element ....................... -->

<!ENTITY % SVG.metadata.extra.content "" >

<!ENTITY % SVG.metadata.element "INCLUDE" >
<![%SVG.metadata.element;[
<!ENTITY % SVG.metadata.content
    "( #PCDATA %SVG.metadata.extra.content; )*"
>
<!ELEMENT %SVG.metadata.qname; %SVG.metadata.content; >
<!-- end of SVG.metadata.element -->]]>

<!ENTITY % SVG.metadata.attlist "INCLUDE" >
<![%SVG.metadata.attlist;[
<!ATTLIST %SVG.metadata.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.metadata.attlist -->]]>

<!-- symbol: Symbol Element ........................... -->

<!ENTITY % SVG.symbol.extra.content "" >

<!ENTITY % SVG.symbol.element "INCLUDE" >
<![%SVG.symbol.element;[
<!ENTITY % SVG.symbol.content
```

```
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.symbol.extra.content; )*"
>
<!ELEMENT %SVG.symbol.qname; %SVG.symbol.content; >
<!-- end of SVG.symbol.element -->]]>

<!ENTITY % SVG.symbol.attlist "INCLUDE" >
<![%SVG.symbol.attlist;[
<!ATTLIST %SVG.symbol.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
<!-- end of SVG.symbol.attlist -->]]>

<!-- use: Use Element ............................... -->

<!ENTITY % SVG.use.extra.content "" >

<!ENTITY % SVG.use.element "INCLUDE" >
<![%SVG.use.element;[
<!ENTITY % SVG.use.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.use.extra.content; )*"
>
<!ELEMENT %SVG.use.qname; %SVG.use.content; >
<!-- end of SVG.use.element -->]]>

<!ENTITY % SVG.use.attlist "INCLUDE" >
<![%SVG.use.attlist;[
<!ATTLIST %SVG.use.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.use.attlist -->]]>

<!-- end of svg-structure.mod -->
```

A.3.18 Basic Structure Module

The Basic Structure Module defines the Description.class, Structure.class and Use.class element collections.

Collection name	Elements in collection
Description.class	desc, title, metadata
Use.class	use
Structure.class	svg, g, defs, Use.class

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Basic Structure Module ....................................... -->
<!-- file: svg-basic-structure.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-structure.mod,v 1.3 2009/06/11 15:58:25 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Basic Structure//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-structure.mod"

     ..................................................................... -->

<!-- Basic Structure

        svg, g, defs, desc, title, metadata, use

     This module declares the major structural elements and their attributes.
-->

<!-- Qualified Names (Default) ......................... -->

<!ENTITY % SVG.svg.qname "svg" >
<!ENTITY % SVG.g.qname "g" >
<!ENTITY % SVG.defs.qname "defs" >
<!ENTITY % SVG.desc.qname "desc" >
<!ENTITY % SVG.title.qname "title" >
<!ENTITY % SVG.metadata.qname "metadata" >
<!ENTITY % SVG.use.qname "use" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.DocumentEvents.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkEmbed.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Description.class ............................. -->

<!ENTITY % SVG.Description.extra.class "" >

<!ENTITY % SVG.Description.class
    "%SVG.desc.qname; | %SVG.title.qname; | %SVG.metadata.qname;
     %SVG.Description.extra.class;"
>

<!-- SVG.Use.class ..................................... -->

<!ENTITY % SVG.Use.extra.class "" >

<!ENTITY % SVG.Use.class
    "| %SVG.use.qname; %SVG.Use.extra.class;"
>

<!-- SVG.Structure.class ............................... -->
```

```
<!ENTITY % SVG.Structure.extra.class "" >

<!ENTITY % SVG.Structure.class
    "| %SVG.g.qname; | %SVG.defs.qname; %SVG.Use.class;
       %SVG.Structure.extra.class;"
>

<!-- SVG.Presentation.attrib ......................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- svg: SVG Document Element ........................ -->

<!ENTITY % SVG.svg.extra.content "" >

<!ENTITY % SVG.svg.element "INCLUDE" >
<![%SVG.svg.element;[
<!ENTITY % SVG.svg.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.svg.extra.content; )*"
>
<!ELEMENT %SVG.svg.qname; %SVG.svg.content; >
<!-- end of SVG.svg.element -->]]>

<!ENTITY % SVG.svg.attlist "INCLUDE" >
<![%SVG.svg.attlist;[
<!ATTLIST %SVG.svg.qname;
    %SVG.xmlns.attrib;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.DocumentEvents.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
    zoomAndPan ( disable | magnify ) 'magnify'
    version %Number.datatype; #FIXED '1.1'
    baseProfile %Text.datatype; #IMPLIED
>
<!-- end of SVG.svg.attlist -->]]>
```

```
<!-- g: Group Element ................................ -->

<!ENTITY % SVG.g.extra.content "" >

<!ENTITY % SVG.g.element "INCLUDE" >
<![%SVG.g.element;[
<!ENTITY % SVG.g.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.g.extra.content; )*"
>
<!ELEMENT %SVG.g.qname; %SVG.g.content; >
<!-- end of SVG.g.element -->]]>

<!ENTITY % SVG.g.attlist "INCLUDE" >
<![%SVG.g.attlist;[
<!ATTLIST %SVG.g.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.g.attlist -->]]>

<!-- defs: Definisions Element ........................ -->

<!ENTITY % SVG.defs.extra.content "" >

<!ENTITY % SVG.defs.element "INCLUDE" >
<![%SVG.defs.element;[
<!ENTITY % SVG.defs.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.defs.extra.content; )*"
>
<!ELEMENT %SVG.defs.qname; %SVG.defs.content; >
<!-- end of SVG.defs.element -->]]>

<!ENTITY % SVG.defs.attlist "INCLUDE" >
<![%SVG.defs.attlist;[
<!ATTLIST %SVG.defs.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.defs.attlist -->]]>

<!-- desc: Description Element ........................ -->

<!ENTITY % SVG.desc.extra.content "" >

<!ENTITY % SVG.desc.element "INCLUDE" >
<![%SVG.desc.element;[
<!ENTITY % SVG.desc.content
    "( #PCDATA %SVG.desc.extra.content; )*"
>
<!ELEMENT %SVG.desc.qname; %SVG.desc.content; >
<!-- end of SVG.desc.element -->]]>

<!ENTITY % SVG.desc.attlist "INCLUDE" >
<![%SVG.desc.attlist;[
<!ATTLIST %SVG.desc.qname;
```

```
    %SVG.Core.attrib;
    %SVG.Style.attrib;
>
<!-- end of SVG.desc.attlist -->]]>

<!-- title: Title Element ............................. -->

<!ENTITY % SVG.title.extra.content "" >

<!ENTITY % SVG.title.element "INCLUDE" >
<![%SVG.title.element;[
<!ENTITY % SVG.title.content
    "( #PCDATA %SVG.title.extra.content; )*"
>
<!ELEMENT %SVG.title.qname; %SVG.title.content; >
<!-- end of SVG.title.element -->]]>

<!ENTITY % SVG.title.attlist "INCLUDE" >
<![%SVG.title.attlist;[
<!ATTLIST %SVG.title.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
>
<!-- end of SVG.title.attlist -->]]>

<!-- metadata: Metadata Element ....................... -->

<!ENTITY % SVG.metadata.extra.content "" >

<!ENTITY % SVG.metadata.element "INCLUDE" >
<![%SVG.metadata.element;[
<!ENTITY % SVG.metadata.content
    "( #PCDATA %SVG.metadata.extra.content; )*"
>
<!ELEMENT %SVG.metadata.qname; %SVG.metadata.content; >
<!-- end of SVG.metadata.element -->]]>

<!ENTITY % SVG.metadata.attlist "INCLUDE" >
<![%SVG.metadata.attlist;[
<!ATTLIST %SVG.metadata.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.metadata.attlist -->]]>

<!-- use: Use Element ................................. -->

<!ENTITY % SVG.use.extra.content "" >

<!ENTITY % SVG.use.element "INCLUDE" >
<![%SVG.use.element;[
<!ENTITY % SVG.use.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.use.extra.content; )*"
>
<!ELEMENT %SVG.use.qname; %SVG.use.content; >
<!-- end of SVG.use.element -->]]>

<!ENTITY % SVG.use.attlist "INCLUDE" >
<![%SVG.use.attlist;[
<!ATTLIST %SVG.use.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.use.attlist -->]]>

<!-- end of svg-basic-structure.mod -->
```

A.3.19 Conditional Processing Module

The Conditional Processing Module defines the Conditional.class element collection and the Conditional.attrib attribute collection.

Collection name	Elements in collection
Conditional.class	switch

Collection name	Attributes in collection
Conditional.attrib	requiredFeatures, requiredExtensions, systemLanguage

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Conditional Processing Module ............................... -->
<!-- file: svg-conditional.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-conditional.mod,v 1.2 2009/06/11 15:58:25 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Conditional Processing//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-conditional.mod"

     ..................................................................... -->

<!-- Conditional Processing

        switch

     This module declares markup to provide support for conditional processing.
-->

<!-- extension list specification -->
<!ENTITY % ExtensionList.datatype "CDATA" >

<!-- feature list specification -->
<!ENTITY % FeatureList.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.switch.qname "switch" >

<!-- Attribute Collections (Default) .................. -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
```

```
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Conditional.class ............................ -->

<!ENTITY % SVG.Conditional.extra.class "" >

<!ENTITY % SVG.Conditional.class
    "| %SVG.switch.qname; %SVG.Conditional.extra.class;"
>

<!-- SVG.Conditional.attrib ........................... -->

<!ENTITY % SVG.Conditional.extra.attrib "" >

<!ENTITY % SVG.Conditional.attrib
    "requiredFeatures %FeatureList.datatype; #IMPLIED
     requiredExtensions %ExtensionList.datatype; #IMPLIED
     systemLanguage %LanguageCodes.datatype; #IMPLIED
     %SVG.Conditional.extra.attrib;"
>

<!-- SVG.Presentation.attrib .......................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- switch: Switch Element ........................... -->

<!ENTITY % SVG.switch.extra.content "" >

<!ENTITY % SVG.switch.element "INCLUDE" >
<![%SVG.switch.element;[
<!ENTITY % SVG.switch.content
    "( %SVG.Description.class; | %SVG.svg.qname; | %SVG.g.qname;
     | %SVG.use.qname; | %SVG.text.qname; | %SVG.Animation.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Shape.class;
       %SVG.Hyperlink.class; %SVG.Extensibility.class;
       %SVG.switch.extra.content; )*"
>
<!ELEMENT %SVG.switch.qname; %SVG.switch.content; >
<!-- end of SVG.switch.element -->]]>

<!ENTITY % SVG.switch.attlist "INCLUDE" >
<![%SVG.switch.attlist;[
<!ATTLIST %SVG.switch.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.switch.attlist -->]]>
```

```
<!-- end of svg-conditional.mod -->
```

A.3.20 Image Module

The Image Module defines the Image.class element collection.

Collection name	Elements in collection
Image.class	image

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Image Module ................................................ -->
<!-- file: svg-image.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-image.mod,v 1.3 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Image//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-image.mod"

     ..................................................................... -->

<!-- Image

        image

     This module declares markup to provide support for image.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.image.qname "image" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkEmbed.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Image.class ................................. -->

<!ENTITY % SVG.Image.extra.class "" >

<!ENTITY % SVG.Image.class
    "| %SVG.image.qname; %SVG.Image.extra.class;"
>

<!-- image: Image Element ............................ -->

<!ENTITY % SVG.image.extra.content "" >

<!ENTITY % SVG.image.element "INCLUDE" >
<![%SVG.image.element;[
<!ENTITY % SVG.image.content
```

```
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.image.extra.content; )*"
>
<!ELEMENT %SVG.image.qname; %SVG.image.content; >
<!-- end of SVG.image.element -->]]>

<!ENTITY % SVG.image.attlist "INCLUDE" >
<![%SVG.image.attlist;[
<!ATTLIST %SVG.image.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #REQUIRED
    height %Length.datatype; #REQUIRED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.image.attlist -->]]>

<!-- end of svg-image.mod -->
```

A.3.21 Style Module

The Style Module defines the Style.class element collection and the Style.attrib attribute collection.

Collection name	Elements in collection
Style.class	style

Collection name	Attributes in collection
Style.attrib	style, class

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Style Module ................................................ -->
<!-- file: svg-style.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-style.mod,v 1.2 2009/06/11 17:20:55 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Style//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-style.mod"

     ..................................................................... -->

<!-- Style

        style

     This module declares markup to provide support for stylesheet.
-->

<!-- list of classes -->
<!ENTITY % ClassList.datatype "CDATA" >

<!-- comma-separated list of media descriptors. -->
<!ENTITY % MediaDesc.datatype "CDATA" >
```

```
<!-- style sheet data -->
<!ENTITY % StyleSheet.datatype "CDATA" >

<!-- Qualified Names (Default) ....................... -->

<!ENTITY % SVG.style.qname "style" >

<!-- Attribute Collections (Default) ................. -->

<!ENTITY % SVG.Core.attrib "" >

<!-- SVG.Style.class ................................. -->

<!ENTITY % SVG.Style.extra.class "" >

<!ENTITY % SVG.Style.class
    "| %SVG.style.qname; %SVG.Style.extra.class;"
>

<!-- SVG.Style.attrib ................................ -->

<!ENTITY % SVG.Style.extra.attrib "" >

<!ENTITY % SVG.Style.attrib
    "style %StyleSheet.datatype; #IMPLIED
     class %ClassList.datatype; #IMPLIED
     %SVG.Style.extra.attrib;"
>

<!-- style: Style Element ............................ -->

<!ENTITY % SVG.style.extra.content "" >

<!ENTITY % SVG.style.element "INCLUDE" >
<![%SVG.style.element;[
<!ENTITY % SVG.style.content
    "( #PCDATA %SVG.style.extra.content; )*"
>
<!ELEMENT %SVG.style.qname; %SVG.style.content; >
<!-- end of SVG.style.element -->]]>

<!ENTITY % SVG.style.attlist "INCLUDE" >
<![%SVG.style.attlist;[
<!ATTLIST %SVG.style.qname;
    xml:space ( preserve ) #FIXED 'preserve'
    %SVG.id.attrib;
    %SVG.base.attrib;
    %SVG.lang.attrib;
    %SVG.Core.extra.attrib;
    type %ContentType.datatype; #REQUIRED
    media %MediaDesc.datatype; #IMPLIED
    title %Text.datatype; #IMPLIED
>
<!-- end of SVG.style.attlist -->]]>

<!-- end of svg-style.mod -->
```

A.3.22 Shape Module

The Shape Module defines the Shape.class element collection.

Collection name	Elements in collection
Shape.class	rect, circle, line, polyline, polygon, ellipse, path

```
<!-- ................................................................ -->
<!-- SVG 1.1 Shape Module ........................................... -->
<!-- file: svg-shape.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
```

```
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-shape.mod,v 1.3 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Shape//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-shape.mod"

     ..................................................................... -->

<!-- Shape

        path, rect, circle, line, ellipse, polyline, polygon

     This module declares markup to provide support for graphical shapes.
-->

<!-- a list of points -->
<!ENTITY % Points.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.path.qname "path" >
<!ENTITY % SVG.rect.qname "rect" >
<!ENTITY % SVG.circle.qname "circle" >
<!ENTITY % SVG.line.qname "line" >
<!ENTITY % SVG.ellipse.qname "ellipse" >
<!ENTITY % SVG.polyline.qname "polyline" >
<!ENTITY % SVG.polygon.qname "polygon" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Shape.class ................................. -->

<!ENTITY % SVG.Shape.extra.class "" >

<!ENTITY % SVG.Shape.class
    "| %SVG.path.qname; | %SVG.rect.qname; | %SVG.circle.qname;
     | %SVG.line.qname; | %SVG.ellipse.qname; | %SVG.polyline.qname;
     | %SVG.polygon.qname; %SVG.Shape.extra.class;"
>

<!-- path: Path Element .............................. -->

<!ENTITY % SVG.path.extra.content "" >

<!ENTITY % SVG.path.element "INCLUDE" >
<![%SVG.path.element;[
<!ENTITY % SVG.path.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.path.extra.content; )*"
>
<!ELEMENT %SVG.path.qname; %SVG.path.content; >
<!-- end of SVG.path.element -->]]>

<!ENTITY % SVG.path.attlist "INCLUDE" >
<![%SVG.path.attlist;[
<!ATTLIST %SVG.path.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
```

```
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    d %PathData.datatype; #REQUIRED
    pathLength %Number.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.path.attlist -->]]>

<!-- rect: Rectangle Element ......................... -->

<!ENTITY % SVG.rect.extra.content "" >

<!ENTITY % SVG.rect.element "INCLUDE" >
<![%SVG.rect.element;[
<!ENTITY % SVG.rect.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.rect.extra.content; )*"
>
<!ELEMENT %SVG.rect.qname; %SVG.rect.content; >
<!-- end of SVG.rect.element -->]]>

<!ENTITY % SVG.rect.attlist "INCLUDE" >
<![%SVG.rect.attlist;[
<!ATTLIST %SVG.rect.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #REQUIRED
    height %Length.datatype; #REQUIRED
    rx %Length.datatype; #IMPLIED
    ry %Length.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.rect.attlist -->]]>

<!-- circle: Circle Element ......................... -->

<!ENTITY % SVG.circle.extra.content "" >

<!ENTITY % SVG.circle.element "INCLUDE" >
<![%SVG.circle.element;[
<!ENTITY % SVG.circle.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.circle.extra.content; )*"
>
<!ELEMENT %SVG.circle.qname; %SVG.circle.content; >
<!-- end of SVG.circle.element -->]]>

<!ENTITY % SVG.circle.attlist "INCLUDE" >
<![%SVG.circle.attlist;[
<!ATTLIST %SVG.circle.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    cx %Coordinate.datatype; #IMPLIED
    cy %Coordinate.datatype; #IMPLIED
    r %Length.datatype; #REQUIRED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.circle.attlist -->]]>

<!-- line: Line Element ............................. -->

<!ENTITY % SVG.line.extra.content "" >

<!ENTITY % SVG.line.element "INCLUDE" >
<![%SVG.line.element;[
<!ENTITY % SVG.line.content
    "( %SVG.Description.class; | %SVG.Animation.class;
```

```
        %SVG.line.extra.content; )*"
>
<!ELEMENT %SVG.line.qname; %SVG.line.content; >
<!-- end of SVG.line.element -->]]>

<!ENTITY % SVG.line.attlist "INCLUDE" >
<![%SVG.line.attlist;[
<!ATTLIST %SVG.line.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x1 %Coordinate.datatype; #IMPLIED
    y1 %Coordinate.datatype; #IMPLIED
    x2 %Coordinate.datatype; #IMPLIED
    y2 %Coordinate.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.line.attlist -->]]>

<!-- ellipse: Ellipse Element ......................... -->

<!ENTITY % SVG.ellipse.extra.content "" >

<!ENTITY % SVG.ellipse.element "INCLUDE" >
<![%SVG.ellipse.element;[
<!ENTITY % SVG.ellipse.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.ellipse.extra.content; )*"
>
<!ELEMENT %SVG.ellipse.qname; %SVG.ellipse.content; >
<!-- end of SVG.ellipse.element -->]]>

<!ENTITY % SVG.ellipse.attlist "INCLUDE" >
<![%SVG.ellipse.attlist;[
<!ATTLIST %SVG.ellipse.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    cx %Coordinate.datatype; #IMPLIED
    cy %Coordinate.datatype; #IMPLIED
    rx %Length.datatype; #REQUIRED
    ry %Length.datatype; #REQUIRED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.ellipse.attlist -->]]>

<!-- polyline: Polyline Element ....................... -->

<!ENTITY % SVG.polyline.extra.content "" >

<!ENTITY % SVG.polyline.element "INCLUDE" >
<![%SVG.polyline.element;[
<!ENTITY % SVG.polyline.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.polyline.extra.content; )*"
>
<!ELEMENT %SVG.polyline.qname; %SVG.polyline.content; >
<!-- end of SVG.polyline.element -->]]>

<!ENTITY % SVG.polyline.attlist "INCLUDE" >
<![%SVG.polyline.attlist;[
<!ATTLIST %SVG.polyline.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    points %Points.datatype; #REQUIRED
    transform %TransformList.datatype; #IMPLIED
>
```

```
<!-- end of SVG.polyline.attlist -->]]>

<!-- polygon: Polygon Element ......................... -->

<!ENTITY % SVG.polygon.extra.content "" >

<!ENTITY % SVG.polygon.element "INCLUDE" >
<![%SVG.polygon.element;[
<!ENTITY % SVG.polygon.content
    "( %SVG.Description.class; | %SVG.Animation.class;
        %SVG.polygon.extra.content; )*"
>
<!ELEMENT %SVG.polygon.qname; %SVG.polygon.content; >
<!-- end of SVG.polygon.element -->]]>

<!ENTITY % SVG.polygon.attlist "INCLUDE" >
<![%SVG.polygon.attlist;[
<!ATTLIST %SVG.polygon.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    points %Points.datatype; #REQUIRED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.polygon.attlist -->]]>

<!-- end of svg-shape.mod -->
```

A.3.23 Text Module

The Text Module defines the Text.class and TextContent.class element collections and the Text.attrib, TextContent.attrib and Font.attrib attribute sets.

Collection name	Elements in collection
Text.class	text, altGlyphDef
TextContent.class	tspan, tref, textPath, altGlyph

Collection name	Attributes in collection
Text.attrib	writing-mode
TextContent.attrib	alignment-baseline, baseline-shift, direction, dominant-baseline, glyph-orientation-horizontal, glyph-orientation-vertical, kerning, letter-spacing, text-anchor, text-decoration, unicode-bidi, word-spacing
Font.attrib	font-family, font-size, font-size-adjust, font-stretch, font-style, font-variant, font-weight

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Text Module .................................................. -->
<!-- file: svg-text.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-text.mod,v 1.2 2009/07/16 07:04:55 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

       PUBLIC "-//W3C//ELEMENTS SVG 1.1 Text//EN"
```

```
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-text.mod"

     ..................................................................... -->

<!-- Text

        text, tspan, tref, textPath, altGlyph, altGlyphDef, altGlyphItem,
        glyphRef

     This module declares markup to provide support for alternate glyph.
-->

<!-- 'baseline-shift' property/attribute value (e.g., 'baseline', 'sub', etc.) -->
<!ENTITY % BaselineShiftValue.datatype "CDATA" >

<!-- 'font-family' property/attribute value (i.e., list of fonts) -->
<!ENTITY % FontFamilyValue.datatype "CDATA" >

<!-- 'font-size' property/attribute value -->
<!ENTITY % FontSizeValue.datatype "CDATA" >

<!-- 'font-size-adjust' property/attribute value -->
<!ENTITY % FontSizeAdjustValue.datatype "CDATA" >

<!-- 'glyph-orientation-horizontal' property/attribute value (e.g., <angle>) -->
<!ENTITY % GlyphOrientationHorizontalValue.datatype "CDATA" >

<!-- 'glyph-orientation-vertical' property/attribute value (e.g., 'auto', <angle>) -->
<!ENTITY % GlyphOrientationVerticalValue.datatype "CDATA" >

<!-- 'kerning' property/attribute value (e.g., 'auto', <length>) -->
<!ENTITY % KerningValue.datatype "CDATA" >

<!-- 'letter-spacing' or 'word-spacing' property/attribute value (e.g., 'normal', <length>) -->
<!ENTITY % SpacingValue.datatype "CDATA" >

<!-- 'text-decoration' property/attribute value (e.g., 'none', 'underline') -->
<!ENTITY % TextDecorationValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.text.qname "text" >
<!ENTITY % SVG.tspan.qname "tspan" >
<!ENTITY % SVG.tref.qname "tref" >
<!ENTITY % SVG.textPath.qname "textPath" >
<!ENTITY % SVG.altGlyph.qname "altGlyph" >
<!ENTITY % SVG.altGlyphDef.qname "altGlyphDef" >
<!ENTITY % SVG.altGlyphItem.qname "altGlyphItem" >
<!ENTITY % SVG.glyphRef.qname "glyphRef" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.XLinkRequired.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Text.class ................................... -->

<!ENTITY % SVG.Text.extra.class "" >

<!ENTITY % SVG.Text.class
    "| %SVG.text.qname; | %SVG.altGlyphDef.qname; %SVG.Text.extra.class;"
>

<!-- SVG.TextContent.class ............................ -->
```

```
<!ENTITY % SVG.TextContent.extra.class "" >

<!ENTITY % SVG.TextContent.class
    "| %SVG.tspan.qname; | %SVG.tref.qname; | %SVG.textPath.qname;
     | %SVG.altGlyph.qname; %SVG.TextContent.extra.class;"
>

<!-- SVG.Text.attrib ................................. -->

<!ENTITY % SVG.Text.extra.attrib "" >

<!ENTITY % SVG.Text.attrib
    "writing-mode ( lr-tb | rl-tb | tb-rl | lr | rl | tb | inherit ) #IMPLIED
     %SVG.Text.extra.attrib;"
>

<!-- SVG.TextContent.attrib ........................... -->

<!ENTITY % SVG.TextContent.extra.attrib "" >

<!ENTITY % SVG.TextContent.attrib
    "alignment-baseline ( auto | baseline | before-edge | text-before-edge |
                          middle | central | after-edge | text-after-edge |
                          ideographic | alphabetic | hanging | mathematical |
                          inherit ) #IMPLIED
     baseline-shift %BaselineShiftValue.datatype; #IMPLIED
     direction ( ltr | rtl | inherit ) #IMPLIED
     dominant-baseline ( auto | use-script | no-change | reset-size |
                         ideographic | alphabetic | hanging | mathematical |
                         central | middle | text-after-edge | text-before-edge |
                         inherit ) #IMPLIED
     glyph-orientation-horizontal %GlyphOrientationHorizontalValue.datatype;
                                  #IMPLIED
     glyph-orientation-vertical %GlyphOrientationVerticalValue.datatype;
                                #IMPLIED
     kerning %KerningValue.datatype; #IMPLIED
     letter-spacing %SpacingValue.datatype; #IMPLIED
     text-anchor ( start | middle | end | inherit ) #IMPLIED
     text-decoration %TextDecorationValue.datatype; #IMPLIED
     unicode-bidi ( normal | embed | bidi-override | inherit ) #IMPLIED
     word-spacing %SpacingValue.datatype; #IMPLIED
     %SVG.TextContent.extra.attrib;"
>

<!-- SVG.Font.attrib ................................. -->

<!ENTITY % SVG.Font.extra.attrib "" >

<!ENTITY % SVG.Font.attrib
    "font-family %FontFamilyValue.datatype; #IMPLIED
     font-size %FontSizeValue.datatype; #IMPLIED
     font-size-adjust %FontSizeAdjustValue.datatype; #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
     %SVG.Font.extra.attrib;"
>

<!-- text: Text Element ............................... -->

<!ENTITY % SVG.text.extra.content "" >

<!ENTITY % SVG.text.element "INCLUDE" >
<![%SVG.text.element;[
<!ENTITY % SVG.text.content
    "( #PCDATA | %SVG.Description.class; | %SVG.Animation.class;
       %SVG.TextContent.class; %SVG.Hyperlink.class;
       %SVG.text.extra.content; )*"
>
<!ELEMENT %SVG.text.qname; %SVG.text.content; >
<!-- end of SVG.text.element -->]]>
```

```
<!ENTITY % SVG.text.attlist "INCLUDE" >
<![%SVG.text.attlist;[
<!ATTLIST %SVG.text.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinates.datatype; #IMPLIED
    y %Coordinates.datatype; #IMPLIED
    dx %Lengths.datatype; #IMPLIED
    dy %Lengths.datatype; #IMPLIED
    rotate %Numbers.datatype; #IMPLIED
    textLength %Length.datatype; #IMPLIED
    lengthAdjust ( spacing | spacingAndGlyphs ) #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.text.attlist -->]]>

<!-- tspan: Text Span Element ......................... -->

<!ENTITY % SVG.tspan.extra.content "" >

<!ENTITY % SVG.tspan.element "INCLUDE" >
<![%SVG.tspan.element;[
<!ENTITY % SVG.tspan.content
    "( #PCDATA | %SVG.tspan.qname; | %SVG.tref.qname; | %SVG.altGlyph.qname;
     | %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
     | %SVG.Description.class; %SVG.Hyperlink.class;
       %SVG.tspan.extra.content; )*"
>
<!ELEMENT %SVG.tspan.qname; %SVG.tspan.content; >
<!-- end of SVG.tspan.element -->]]>

<!ENTITY % SVG.tspan.attlist "INCLUDE" >
<![%SVG.tspan.attlist;[
<!ATTLIST %SVG.tspan.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinates.datatype; #IMPLIED
    y %Coordinates.datatype; #IMPLIED
    dx %Lengths.datatype; #IMPLIED
    dy %Lengths.datatype; #IMPLIED
    rotate %Numbers.datatype; #IMPLIED
    textLength %Length.datatype; #IMPLIED
    lengthAdjust ( spacing | spacingAndGlyphs ) #IMPLIED
>
<!-- end of SVG.tspan.attlist -->]]>

<!-- tref: Text Reference Element ..................... -->

<!ENTITY % SVG.tref.extra.content "" >

<!ENTITY % SVG.tref.element "INCLUDE" >
<![%SVG.tref.element;[
<!ENTITY % SVG.tref.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
     | %SVG.Description.class; %SVG.tref.extra.content; )*"
>
<!ELEMENT %SVG.tref.qname; %SVG.tref.content; >
<!-- end of SVG.tref.element -->]]>

<!ENTITY % SVG.tref.attlist "INCLUDE" >
<![%SVG.tref.attlist;[
<!ATTLIST %SVG.tref.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkRequired.attrib;
```

```
    %SVG.External.attrib;
    x %Coordinates.datatype; #IMPLIED
    y %Coordinates.datatype; #IMPLIED
    dx %Lengths.datatype; #IMPLIED
    dy %Lengths.datatype; #IMPLIED
    rotate %Numbers.datatype; #IMPLIED
    textLength %Length.datatype; #IMPLIED
    lengthAdjust ( spacing | spacingAndGlyphs ) #IMPLIED
>
<!-- end of SVG.tref.attlist -->]]>

<!-- textPath: Text Path Element ..................... -->

<!ENTITY % SVG.textPath.extra.content "" >

<!ENTITY % SVG.textPath.element "INCLUDE" >
<![%SVG.textPath.element;[
<!ENTITY % SVG.textPath.content
    "( #PCDATA | %SVG.tspan.qname; | %SVG.tref.qname; | %SVG.altGlyph.qname;
     | %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
     | %SVG.Description.class; %SVG.Hyperlink.class;
       %SVG.textPath.extra.content; )*"
>
<!ELEMENT %SVG.textPath.qname; %SVG.textPath.content; >
<!-- end of SVG.textPath.element -->]]>

<!ENTITY % SVG.textPath.attlist "INCLUDE" >
<![%SVG.textPath.attlist;[
<!ATTLIST %SVG.textPath.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkRequired.attrib;
    %SVG.External.attrib;
    startOffset %Length.datatype; #IMPLIED
    textLength %Length.datatype; #IMPLIED
    lengthAdjust ( spacing | spacingAndGlyphs ) #IMPLIED
    method ( align | stretch ) #IMPLIED
    spacing ( auto | exact ) #IMPLIED
>
<!-- end of SVG.textPath.attlist -->]]>

<!-- altGlyph: Alternate Glyph Element ................. -->

<!ENTITY % SVG.altGlyph.extra.content "" >

<!ENTITY % SVG.altGlyph.element "INCLUDE" >
<![%SVG.altGlyph.element;[
<!ENTITY % SVG.altGlyph.content
    "( #PCDATA %SVG.altGlyph.extra.content; )*"
>
<!ELEMENT %SVG.altGlyph.qname; %SVG.altGlyph.content; >
<!-- end of SVG.altGlyph.element -->]]>

<!ENTITY % SVG.altGlyph.attlist "INCLUDE" >
<![%SVG.altGlyph.attlist;[
<!ATTLIST %SVG.altGlyph.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x %Coordinates.datatype; #IMPLIED
    y %Coordinates.datatype; #IMPLIED
    dx %Lengths.datatype; #IMPLIED
    dy %Lengths.datatype; #IMPLIED
    glyphRef CDATA #IMPLIED
    format CDATA #IMPLIED
    rotate %Numbers.datatype; #IMPLIED
>
<!-- end of SVG.altGlyph.attlist -->]]>

<!-- altGlyphDef: Alternate Glyph Definition Element ... -->
```

```
<!ENTITY % SVG.altGlyphDef.extra.content "" >

<!ENTITY % SVG.altGlyphDef.element "INCLUDE" >
<![%SVG.altGlyphDef.element;[
<!ENTITY % SVG.altGlyphDef.content
    "(( %SVG.glyphRef.qname;+ | %SVG.altGlyphItem.qname;+ )
        %SVG.altGlyphDef.extra.content; )"
>
<!ELEMENT %SVG.altGlyphDef.qname; %SVG.altGlyphDef.content; >
<!-- end of SVG.altGlyphDef.element -->]]>

<!ENTITY % SVG.altGlyphDef.attlist "INCLUDE" >
<![%SVG.altGlyphDef.attlist;[
<!ATTLIST %SVG.altGlyphDef.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.altGlyphDef.attlist -->]]>

<!-- altGlyphItem: Alternate Glyph Item Element ........ -->

<!ENTITY % SVG.altGlyphItem.extra.content "" >

<!ENTITY % SVG.altGlyphItem.element "INCLUDE" >
<![%SVG.altGlyphItem.element;[
<!ENTITY % SVG.altGlyphItem.content
    "( %SVG.glyphRef.qname;+ %SVG.altGlyphItem.extra.content; )"
>
<!ELEMENT %SVG.altGlyphItem.qname; %SVG.altGlyphItem.content; >
<!-- end of SVG.altGlyphItem.element -->]]>

<!ENTITY % SVG.altGlyphItem.attlist "INCLUDE" >
<![%SVG.altGlyphItem.attlist;[
<!ATTLIST %SVG.altGlyphItem.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.altGlyphItem.attlist -->]]>

<!-- glyphRef: Glyph Reference Element ................. -->

<!ENTITY % SVG.glyphRef.element "INCLUDE" >
<![%SVG.glyphRef.element;[
<!ENTITY % SVG.glyphRef.content "EMPTY" >
<!ELEMENT %SVG.glyphRef.qname; %SVG.glyphRef.content; >
<!-- end of SVG.glyphRef.element -->]]>

<!ENTITY % SVG.glyphRef.attlist "INCLUDE" >
<![%SVG.glyphRef.attlist;[
<!ATTLIST %SVG.glyphRef.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    x %Number.datatype; #IMPLIED
    y %Number.datatype; #IMPLIED
    dx %Number.datatype; #IMPLIED
    dy %Number.datatype; #IMPLIED
    glyphRef CDATA #IMPLIED
    format CDATA #IMPLIED
>
<!-- end of SVG.glyphRef.attlist -->]]>

<!-- end of svg-text.mod -->
```

A.3.24 Basic Text Module

The Basic Text Module defines the Text.class and TextContent.class element collections and the TextContent.attrib and Font.attrib attribute sets.

Collection name	Elements in collection
Text.class	text
TextContent.class	(empty)

Collection name	Attributes in collection
TextContent.attrib	text-anchor
Font.attrib	font-family, font-size, font-style, font-weight

```
<!-- ....................................................................... -->
<!-- SVG 1.1 Basic Text Module .............................................. -->
<!-- file: svg-basic-text.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-text.mod,v 1.2 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Basic Text//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-text.mod"

     ....................................................................... -->

<!-- Basic Text

        text

     This module declares markup to provide support for text.
-->

<!-- 'font-family' property/attribute value (i.e., list of fonts) -->
<!ENTITY % FontFamilyValue.datatype "CDATA" >

<!-- 'font-size' property/attribute value -->
<!ENTITY % FontSizeValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.text.qname "text" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Text.class ................................... -->

<!ENTITY % SVG.Text.extra.class "" >

<!ENTITY % SVG.Text.class
    "| %SVG.text.qname; %SVG.Text.extra.class;"
>

<!-- SVG.TextContent.attrib ........................... -->

<!ENTITY % SVG.TextContent.extra.attrib "" >
```

```
<!ENTITY % SVG.TextContent.attrib
    "text-anchor ( start | middle | end | inherit ) #IMPLIED
     %SVG.TextContent.extra.attrib;"
>

<!-- SVG.Font.attrib ................................ -->

<!ENTITY % SVG.Font.extra.attrib "" >

<!ENTITY % SVG.Font.attrib
    "font-family %FontFamilyValue.datatype; #IMPLIED
     font-size %FontSizeValue.datatype; #IMPLIED
     font-style ( normal | italic | oblique | inherit ) #IMPLIED
     font-weight ( normal | bold | bolder | lighter | 100 | 200 | 300 | 400 |
                   500 | 600 | 700 | 800 | 900 | inherit ) #IMPLIED
     %SVG.Font.extra.attrib;"
>

<!-- text: Text Element .............................. -->

<!ENTITY % SVG.text.extra.content "" >

<!ENTITY % SVG.text.element "INCLUDE" >
<![%SVG.text.element;[
<!ENTITY % SVG.text.content
    "( #PCDATA | %SVG.Description.class; | %SVG.Animation.class;
       %SVG.Hyperlink.class; %SVG.text.extra.content; )*"
>
<!ELEMENT %SVG.text.qname; %SVG.text.content; >
<!-- end of SVG.text.element -->]]>

<!ENTITY % SVG.text.attlist "INCLUDE" >
<![%SVG.text.attlist;[
<!ATTLIST %SVG.text.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.External.attrib;
    x %Coordinates.datatype; #IMPLIED
    y %Coordinates.datatype; #IMPLIED
    rotate %Numbers.datatype; #IMPLIED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.text.attlist -->]]>

<!-- end of svg-basic-text.mod -->
```

A.3.25 Marker Module

The Marker Module defines the Marker.class element collection and the Marker.attrib attribute set.

Collection name	Elements in collection
Marker.class	marker

Collection name	Attributes in collection
Marker.attrib	marker-start, marker-mid, marker-end

```
<!-- ....................................................................... -->
<!-- SVG 1.1 Marker Module ................................................. -->
<!-- file: svg-marker.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
```

```
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-marker.mod,v 1.2 2009/05/30 04:21:38 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Marker//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-marker.mod"

     ..................................................................... -->

<!-- Marker

        marker

     This module declares markup to provide support for marker.
-->

<!-- 'marker' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % MarkerValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.marker.qname "marker" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Marker.class ................................ -->

<!ENTITY % SVG.Marker.extra.class "" >

<!ENTITY % SVG.Marker.class
    "| %SVG.marker.qname; %SVG.Marker.extra.class;"
>

<!-- SVG.Marker.attrib ................................ -->

<!ENTITY % SVG.Marker.extra.attrib "" >

<!ENTITY % SVG.Marker.attrib
    "marker-start %MarkerValue.datatype; #IMPLIED
     marker-mid %MarkerValue.datatype; #IMPLIED
     marker-end %MarkerValue.datatype; #IMPLIED
     %SVG.Marker.extra.attrib;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
```

```
      %SVG.Opacity.attrib;
      %SVG.Graphics.attrib;
      %SVG.Marker.attrib;
      %SVG.ColorProfile.attrib;
      %SVG.Gradient.attrib;
      %SVG.Clip.attrib;
      %SVG.Mask.attrib;
      %SVG.Filter.attrib;
      %SVG.FilterColor.attrib;
      %SVG.Cursor.attrib;
      flood-color %SVGColor.datatype; #IMPLIED
      flood-opacity %OpacityValue.datatype; #IMPLIED
      lighting-color %SVGColor.datatype; #IMPLIED
      %SVG.Presentation.extra.attrib;"
>

<!-- marker: Marker Element ........................... -->

<!ENTITY % SVG.marker.extra.content "" >

<!ENTITY % SVG.marker.element "INCLUDE" >
<![%SVG.marker.element;[
<!ENTITY % SVG.marker.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.marker.extra.content; )*"
>
<!ELEMENT %SVG.marker.qname; %SVG.marker.content; >
<!-- end of SVG.marker.element -->]]>

<!ENTITY % SVG.marker.attlist "INCLUDE" >
<![%SVG.marker.attlist;[
<!ATTLIST %SVG.marker.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.External.attrib;
    refX %Coordinate.datatype; #IMPLIED
    refY %Coordinate.datatype; #IMPLIED
    markerUnits ( strokeWidth | userSpaceOnUse ) #IMPLIED
    markerWidth  %Length.datatype; #IMPLIED
    markerHeight %Length.datatype; #IMPLIED
    orient CDATA #IMPLIED
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
<!-- end of SVG.marker.attlist -->]]>

<!-- end of svg-marker.mod -->
```

A.3.26 Color Profile Module

The Color Profile Module defines the ColorProfile.class element collection.

Collection name	Elements in collection
ColorProfile.class	color-profile

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Color Profile Module ....................................... -->
<!-- file: svg-profile.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-profile.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $
```

```
     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Color Profile//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-profile.mod"

     ..................................................................... -->

<!-- Color Profile

        color-profile

     This module declares markup to provide support for color profile.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.color-profile.qname "color-profile" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >

<!-- SVG.ColorProfile.class ........................... -->

<!ENTITY % SVG.ColorProfile.extra.class "" >

<!ENTITY % SVG.ColorProfile.class
    "| %SVG.color-profile.qname; %SVG.ColorProfile.extra.class;"
>

<!-- SVG.ColorProfile.attrib ........................... -->

<!ENTITY % SVG.ColorProfile.extra.attrib "" >

<!ENTITY % SVG.ColorProfile.attrib
    "color-profile CDATA #IMPLIED
     %SVG.ColorProfile.extra.attrib;"
>

<!-- color-profile: Color Profile Element .............. -->

<!ENTITY % SVG.color-profile.extra.content "" >

<!ENTITY % SVG.color-profile.element "INCLUDE" >
<![%SVG.color-profile.element;[
<!ENTITY % SVG.color-profile.content
    "( %SVG.Description.class; %SVG.color-profile.extra.content; )*"
>
<!ELEMENT %SVG.color-profile.qname; %SVG.color-profile.content; >
<!-- end of SVG.color-profile.element -->]]>

<!ENTITY % SVG.color-profile.attlist "INCLUDE" >
<![%SVG.color-profile.attlist;[
<!ATTLIST %SVG.color-profile.qname;
    %SVG.Core.attrib;
    %SVG.XLink.attrib;
    local CDATA #IMPLIED
    name CDATA #REQUIRED
    rendering-intent ( auto | perceptual | relative-colorimetric | saturation |
                       absolute-colorimetric ) 'auto'
>
<!-- end of SVG.color-profile.attlist -->]]>

<!-- end of svg-profile.mod -->
```

A.3.27 Gradient Module

The Gradient Module defines the Gradient.class element collection and the Gradient.attrib attribute set.

Collection name	Elements in collection
Gradient.class	linearGradient, radialGradient

Collection name	Attributes in collection
Gradient.attrib	stop-color, stop-opacity

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Gradient Module ............................................. -->
<!-- file: svg-gradient.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-gradient.mod,v 1.4 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Gradient//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-gradient.mod"

     ..................................................................... -->

<!-- Gradient

        linearGradient, radialGradient, stop

     This module declares markup to provide support for gradient fill.
-->

<!-- a <number> or a <percentage> -->
<!ENTITY % NumberOrPercentage.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.linearGradient.qname "linearGradient" >
<!ENTITY % SVG.radialGradient.qname "radialGradient" >
<!ENTITY % SVG.stop.qname "stop" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Gradient.class ............................... -->

<!ENTITY % SVG.Gradient.extra.class "" >

<!ENTITY % SVG.Gradient.class
    "| %SVG.linearGradient.qname; | %SVG.radialGradient.qname;
       %SVG.Gradient.extra.class;"
>

<!-- SVG.Gradient.attrib .............................. -->

<!ENTITY % SVG.Gradient.extra.attrib "" >

<!ENTITY % SVG.Gradient.attrib
    "stop-color %SVGColor.datatype; #IMPLIED
     stop-opacity %OpacityValue.datatype; #IMPLIED
     %SVG.Gradient.extra.attrib;"
>

<!-- linearGradient: Linear Gradient Element ........... -->

<!ENTITY % SVG.linearGradient.extra.content "" >

<!ENTITY % SVG.linearGradient.element "INCLUDE" >
<![%SVG.linearGradient.element;[
```

```
<!ENTITY % SVG.linearGradient.content
    "( %SVG.Description.class; | %SVG.stop.qname; | %SVG.animate.qname;
     | %SVG.set.qname; | %SVG.animateTransform.qname;
       %SVG.linearGradient.extra.content; )*"
>
<!ELEMENT %SVG.linearGradient.qname; %SVG.linearGradient.content; >
<!-- end of SVG.linearGradient.element -->]]>

<!ENTITY % SVG.linearGradient.attlist "INCLUDE" >
<![%SVG.linearGradient.attlist;[
<!ATTLIST %SVG.linearGradient.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x1 %Coordinate.datatype; #IMPLIED
    y1 %Coordinate.datatype; #IMPLIED
    x2 %Coordinate.datatype; #IMPLIED
    y2 %Coordinate.datatype; #IMPLIED
    gradientUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    gradientTransform %TransformList.datatype; #IMPLIED
    spreadMethod ( pad | reflect | repeat ) #IMPLIED
>
<!-- end of SVG.linearGradient.attlist -->]]>

<!-- radialGradient: Radial Gradient Element ........... -->

<!ENTITY % SVG.radialGradient.extra.content "" >

<!ENTITY % SVG.radialGradient.element "INCLUDE" >
<![%SVG.radialGradient.element;[
<!ENTITY % SVG.radialGradient.content
    "( %SVG.Description.class; | %SVG.stop.qname; | %SVG.animate.qname;
     | %SVG.set.qname; | %SVG.animateTransform.qname;
       %SVG.radialGradient.extra.content; )*"
>
<!ELEMENT %SVG.radialGradient.qname; %SVG.radialGradient.content; >
<!-- end of SVG.radialGradient.element -->]]>

<!ENTITY % SVG.radialGradient.attlist "INCLUDE" >
<![%SVG.radialGradient.attlist;[
<!ATTLIST %SVG.radialGradient.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    cx %Coordinate.datatype; #IMPLIED
    cy %Coordinate.datatype; #IMPLIED
    r %Length.datatype; #IMPLIED
    fx %Coordinate.datatype; #IMPLIED
    fy %Coordinate.datatype; #IMPLIED
    gradientUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    gradientTransform %TransformList.datatype; #IMPLIED
    spreadMethod ( pad | reflect | repeat ) #IMPLIED
>
<!-- end of SVG.radialGradient.attlist -->]]>

<!-- stop: Stop Element ............................... -->

<!ENTITY % SVG.stop.extra.content "" >

<!ENTITY % SVG.stop.element "INCLUDE" >
<![%SVG.stop.element;[
<!ENTITY % SVG.stop.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
       %SVG.stop.extra.content; )*"
>
<!ELEMENT %SVG.stop.qname; %SVG.stop.content; >
<!-- end of SVG.stop.element -->]]>

<!ENTITY % SVG.stop.attlist "INCLUDE" >
<![%SVG.stop.attlist;[
<!ATTLIST %SVG.stop.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
```

```
    %SVG.Presentation.attrib;
    offset %NumberOrPercentage.datatype; #REQUIRED
>
<!-- end of SVG.stop.attlist -->]]>

<!-- end of svg-gradient.mod -->
```

A.3.28 Pattern Module

The Pattern Module defines the Pattern.class element collection.

Collection name	Elements in collection
Pattern.class	pattern

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Pattern Module ............................................... -->
<!-- file: svg-pattern.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-pattern.mod,v 1.2 2009/05/30 04:21:39 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Pattern//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-pattern.mod"

     ..................................................................... -->

<!-- Pattern

        pattern

     This module declares markup to provide support for pattern fill.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.pattern.qname "pattern" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Pattern.class ............................... -->

<!ENTITY % SVG.Pattern.extra.class "" >
```

```
<!ENTITY % SVG.Pattern.class
    "| %SVG.pattern.qname; %SVG.Pattern.extra.class;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- pattern: Pattern Element ......................... -->

<!ENTITY % SVG.pattern.extra.content "" >

<!ENTITY % SVG.pattern.element "INCLUDE" >
<![%SVG.pattern.element;[
<!ENTITY % SVG.pattern.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.pattern.extra.content; )*"
>
<!ELEMENT %SVG.pattern.qname; %SVG.pattern.content; >
<!-- end of SVG.pattern.element -->]]>

<!ENTITY % SVG.pattern.attlist "INCLUDE" >
<![%SVG.pattern.attlist;[
<!ATTLIST %SVG.pattern.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    patternUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    patternContentUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    patternTransform %TransformList.datatype; #IMPLIED
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
<!-- end of SVG.pattern.attlist -->]]>

<!-- end of svg-pattern.mod -->
```

A.3.29 Clip Module

The Clip Module defines the Clip.class element collection and the Clip.attrib attribute collection.

Collection name	Elements in collection
Clip.class	clipPath

Collection name	Attributes in collection
Clip.attrib	clip-path, clip-rule

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Clip Module .................................................. -->
<!-- file: svg-clip.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-clip.mod,v 1.3 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Clip//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-clip.mod"

     ..................................................................... -->

<!-- Clip

        clipPath

     This module declares markup to provide support for clipping.
-->

<!-- 'clip-path' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % ClipPathValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.clipPath.qname "clipPath" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Clip.class ................................... -->

<!ENTITY % SVG.Clip.extra.class "" >

<!ENTITY % SVG.Clip.class
    "| %SVG.clipPath.qname; %SVG.Clip.extra.class;"
>

<!-- SVG.Clip.attrib ................................. -->
```

```
<!ENTITY % SVG.Clip.extra.attrib "" >

<!ENTITY % SVG.Clip.attrib
    "clip-path %ClipPathValue.datatype; #IMPLIED
     clip-rule %ClipFillRule.datatype; #IMPLIED
     %SVG.Clip.extra.attrib;"
>

<!-- clipPath: Clip Path Element ..................... -->

<!ENTITY % SVG.clipPath.extra.content "" >

<!ENTITY % SVG.clipPath.element "INCLUDE" >
<![%SVG.clipPath.element;[
<!ENTITY % SVG.clipPath.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Use.class;
       %SVG.Shape.class; | %SVG.text.qname; %SVG.clipPath.extra.content; )*"
>
<!ELEMENT %SVG.clipPath.qname; %SVG.clipPath.content; >
<!-- end of SVG.clipPath.element -->]]>

<!ENTITY % SVG.clipPath.attlist "INCLUDE" >
<![%SVG.clipPath.attlist;[
<!ATTLIST %SVG.clipPath.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
    clipPathUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
<!-- end of SVG.clipPath.attlist -->]]>

<!-- end of svg-clip.mod -->
```

A.3.30 Basic Clip Module

The Basic Clip Module defines the Clip.class element collection and the Clip.attrib attribute collection.

Collection name	Elements in collection
Clip.class	clipPath

Collection name	Attributes in collection
Clip.attrib	clip-path, clip-rule

```
<!-- ................................................................... -->
<!-- SVG 1.1 Basic Clip Module ........................................ -->
<!-- file: svg-basic-clip.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-clip.mod,v 1.3 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Basic Clip//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-clip.mod"

     ................................................................... -->

<!-- Basic Clip

        clipPath
```

```
     This module declares markup to provide support for clipping.
-->

<!-- 'clip-path' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % ClipPathValue.datatype "CDATA" >

<!-- Qualified Names (Default) ....................... -->

<!ENTITY % SVG.clipPath.qname "clipPath" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Clip.class ................................. -->

<!ENTITY % SVG.Clip.extra.class "" >

<!ENTITY % SVG.Clip.class
    "| %SVG.clipPath.qname; %SVG.Clip.extra.class;"
>

<!-- SVG.Clip.attrib ................................ -->

<!ENTITY % SVG.Clip.extra.attrib "" >

<!ENTITY % SVG.Clip.attrib
    "clip-path %ClipPathValue.datatype; #IMPLIED
     clip-rule %ClipFillRule.datatype; #IMPLIED
     %SVG.Clip.extra.attrib;"
>

<!-- clipPath: Clip Path Element ..................... -->

<!ENTITY % SVG.clipPath.extra.content "" >

<!ENTITY % SVG.clipPath.element "INCLUDE" >
<![%SVG.clipPath.element;[
<!ENTITY % SVG.clipPath.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Use.class;
       %SVG.Shape.class; | %SVG.text.qname; %SVG.clipPath.extra.content; )*"
>
<!ELEMENT %SVG.clipPath.qname; %SVG.clipPath.content; >
<!-- end of SVG.clipPath.element -->]]>

<!ENTITY % SVG.clipPath.attlist "INCLUDE" >
<![%SVG.clipPath.attlist;[
<!ATTLIST %SVG.clipPath.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
    clipPathUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
<!-- end of SVG.clipPath.attlist -->]]>

<!-- end of svg-basic-clip.mod -->
```

A.3.31 Mask Module

The Mask Module defines the Mask.class element collection and the Mask.attrib attribute collection.

Collection name	Elements in collection
Mask.class	mask

Collection name	Attributes in collection
Mask.attrib	mask

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Mask Module .................................................. -->
<!-- file: svg-mask.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-mask.mod,v 1.2 2009/05/30 04:21:38 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Mask//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-mask.mod"

     ..................................................................... -->

<!-- Mask

        mask

     This module declares markup to provide support for masking.
-->

<!-- 'mask' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % MaskValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.mask.qname "mask" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Mask.class ................................... -->

<!ENTITY % SVG.Mask.extra.class "" >
```

```
<!ENTITY % SVG.Mask.class
    "| %SVG.mask.qname; %SVG.Mask.extra.class;"
>

<!-- SVG.Mask.attrib ................................ -->

<!ENTITY % SVG.Mask.extra.attrib "" >

<!ENTITY % SVG.Mask.attrib
    "mask %MaskValue.datatype; #IMPLIED
     %SVG.Mask.extra.attrib;"
>

<!-- SVG.Presentation.attrib ........................ -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- mask: Mask Element ............................. -->

<!ENTITY % SVG.mask.extra.content "" >

<!ENTITY % SVG.mask.element "INCLUDE" >
<![%SVG.mask.element;[
<!ENTITY % SVG.mask.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.mask.extra.content; )*"
>
<!ELEMENT %SVG.mask.qname; %SVG.mask.content; >
<!-- end of SVG.mask.element -->]]>

<!ENTITY % SVG.mask.attlist "INCLUDE" >
<![%SVG.mask.attlist;[
<!ATTLIST %SVG.mask.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    maskUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    maskContentUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
<!-- end of SVG.mask.attlist -->]]>

<!-- end of svg-mask.mod -->
```

A.3.32 Filter Module

The Filter Module defines the Filter.class and FilterPrimitive.class element collections and the Filter.attrib, FilterColor.attrib, FilterPrimitive.attrib and FilterPrimitiveWithIn.attrib attribute collections.

Collection name	Elements in collection
Filter.class	filter
FilterPrimitive.class	feBlend, feFlood, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feGaussianBlur, feImage, feMerge, feMorphology, feOffset, feSpecularLighting, feTile, feTurbulence

Collection name	Attributes in collection
Filter.attrib	filter
FilterColor.attrib	color-interpolation-filters
FilterPrimitive.attrib	x, y, width, height, result
FilterPrimitiveWithIn.attrib	FilterPrimitive.attrib, in

```
<!-- ....................................................................... -->
<!-- SVG 1.1 Filter Module ................................................. -->
<!-- file: svg-filter.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-filter.mod,v 1.5 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Filter//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-filter.mod"

     ....................................................................... -->

<!-- Filter

        filter, feBlend, feColorMatrix, feComponentTransfer, feComposite,
        feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feFlood,
        feGaussianBlur, feImage, feMerge, feMergeNode, feMorphology, feOffset,
        feSpecularLighting, feTile, feTurbulence, feDistantLight, fePointLight,
        feSpotLight, feFuncR, feFuncG, feFuncB, feFuncA

     This module declares markup to provide support for filter effect.
-->

<!-- 'filter' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % FilterValue.datatype "CDATA" >

<!-- list of <number>s, but at least one and at most two -->
<!ENTITY % NumberOptionalNumber.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.filter.qname "filter" >
<!ENTITY % SVG.feBlend.qname "feBlend" >
<!ENTITY % SVG.feColorMatrix.qname "feColorMatrix" >
<!ENTITY % SVG.feComponentTransfer.qname "feComponentTransfer" >
<!ENTITY % SVG.feComposite.qname "feComposite" >
<!ENTITY % SVG.feConvolveMatrix.qname "feConvolveMatrix" >
<!ENTITY % SVG.feDiffuseLighting.qname "feDiffuseLighting" >
```

```
<!ENTITY % SVG.feDisplacementMap.qname "feDisplacementMap" >
<!ENTITY % SVG.feFlood.qname "feFlood" >
<!ENTITY % SVG.feGaussianBlur.qname "feGaussianBlur" >
<!ENTITY % SVG.feImage.qname "feImage" >
<!ENTITY % SVG.feMerge.qname "feMerge" >
<!ENTITY % SVG.feMergeNode.qname "feMergeNode" >
<!ENTITY % SVG.feMorphology.qname "feMorphology" >
<!ENTITY % SVG.feOffset.qname "feOffset" >
<!ENTITY % SVG.feSpecularLighting.qname "feSpecularLighting" >
<!ENTITY % SVG.feTile.qname "feTile" >
<!ENTITY % SVG.feTurbulence.qname "feTurbulence" >
<!ENTITY % SVG.feDistantLight.qname "feDistantLight" >
<!ENTITY % SVG.fePointLight.qname "fePointLight" >
<!ENTITY % SVG.feSpotLight.qname "feSpotLight" >
<!ENTITY % SVG.feFuncR.qname "feFuncR" >
<!ENTITY % SVG.feFuncG.qname "feFuncG" >
<!ENTITY % SVG.feFuncB.qname "feFuncB" >
<!ENTITY % SVG.feFuncA.qname "feFuncA" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.XLinkEmbed.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Filter.class ................................ -->

<!ENTITY % SVG.Filter.extra.class "" >

<!ENTITY % SVG.Filter.class
    "| %SVG.filter.qname; %SVG.Filter.extra.class;"
>

<!-- SVG.FilterPrimitive.class ........................ -->

<!ENTITY % SVG.FilterPrimitive.extra.class "" >

<!ENTITY % SVG.FilterPrimitive.class
    "| %SVG.feBlend.qname; | %SVG.feColorMatrix.qname;
     | %SVG.feComponentTransfer.qname; | %SVG.feComposite.qname;
     | %SVG.feConvolveMatrix.qname; | %SVG.feDiffuseLighting.qname;
     | %SVG.feDisplacementMap.qname; | %SVG.feFlood.qname;
     | %SVG.feGaussianBlur.qname; | %SVG.feImage.qname; | %SVG.feMerge.qname;
     | %SVG.feMorphology.qname; | %SVG.feOffset.qname;
     | %SVG.feSpecularLighting.qname; | %SVG.feTile.qname;
     | %SVG.feTurbulence.qname; %SVG.FilterPrimitive.extra.class;"
>

<!-- SVG.Filter.attrib ................................ -->

<!ENTITY % SVG.Filter.extra.attrib "" >

<!ENTITY % SVG.Filter.attrib
    "filter %FilterValue.datatype; #IMPLIED
     %SVG.Filter.extra.attrib;"
>

<!-- SVG.FilterColor.attrib ........................... -->

<!ENTITY % SVG.FilterColor.extra.attrib "" >
```

```
<!ENTITY % SVG.FilterColor.attrib
    "color-interpolation-filters ( auto | sRGB | linearRGB | inherit )
                                 #IMPLIED
     %SVG.FilterColor.extra.attrib;"
>

<!-- SVG.FilterPrimitive.attrib ....................... -->

<!ENTITY % SVG.FilterPrimitive.extra.attrib "" >

<!ENTITY % SVG.FilterPrimitive.attrib
    "x %Coordinate.datatype; #IMPLIED
     y %Coordinate.datatype; #IMPLIED
     width %Length.datatype; #IMPLIED
     height %Length.datatype; #IMPLIED
     result CDATA #IMPLIED
     %SVG.FilterPrimitive.extra.attrib;"
>

<!-- SVG.FilterPrimitiveWithIn.attrib .................. -->

<!ENTITY % SVG.FilterPrimitiveWithIn.extra.attrib "" >

<!ENTITY % SVG.FilterPrimitiveWithIn.attrib
    "%SVG.FilterPrimitive.attrib;
     in CDATA #IMPLIED
     %SVG.FilterPrimitiveWithIn.extra.attrib;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- filter: Filter Element ............................ -->

<!ENTITY % SVG.filter.extra.content "" >

<!ENTITY % SVG.filter.element "INCLUDE" >
<![%SVG.filter.element;[
<!ENTITY % SVG.filter.content
    "( %SVG.Description.class; | %SVG.animate.qname; | %SVG.set.qname;
        %SVG.FilterPrimitive.class; %SVG.filter.extra.content; )*"
>
<!ELEMENT %SVG.filter.qname; %SVG.filter.content; >
<!-- end of SVG.filter.element -->]]>

<!ENTITY % SVG.filter.attlist "INCLUDE" >
<![%SVG.filter.attlist;[
<!ATTLIST %SVG.filter.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
```

```
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    filterRes %NumberOptionalNumber.datatype; #IMPLIED
    filterUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    primitiveUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
<!-- end of SVG.filter.attlist -->]]>

<!-- feBlend: Filter Effect Blend Element ............. -->

<!ENTITY % SVG.feBlend.extra.content "" >

<!ENTITY % SVG.feBlend.element "INCLUDE" >
<![%SVG.feBlend.element;[
<!ENTITY % SVG.feBlend.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feBlend.extra.content; )*"
>
<!ELEMENT %SVG.feBlend.qname; %SVG.feBlend.content; >
<!-- end of SVG.feBlend.element -->]]>

<!ENTITY % SVG.feBlend.attlist "INCLUDE" >
<![%SVG.feBlend.attlist;[
<!ATTLIST %SVG.feBlend.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    mode ( normal | multiply | screen | darken | lighten ) 'normal'
>
<!-- end of SVG.feBlend.attlist -->]]>

<!-- feColorMatrix: Filter Effect Color Matrix Element . -->

<!ENTITY % SVG.feColorMatrix.extra.content "" >

<!ENTITY % SVG.feColorMatrix.element "INCLUDE" >
<![%SVG.feColorMatrix.element;[
<!ENTITY % SVG.feColorMatrix.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feColorMatrix.extra.content; )*"
>
<!ELEMENT %SVG.feColorMatrix.qname; %SVG.feColorMatrix.content; >
<!-- end of SVG.feColorMatrix.element -->]]>

<!ENTITY % SVG.feColorMatrix.attlist "INCLUDE" >
<![%SVG.feColorMatrix.attlist;[
<!ATTLIST %SVG.feColorMatrix.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    type ( matrix | saturate | hueRotate | luminanceToAlpha ) 'matrix'
    values CDATA #IMPLIED
>
<!-- end of SVG.feColorMatrix.attlist -->]]>

<!-- feComponentTransfer: Filter Effect Component Transfer Element -->

<!ENTITY % SVG.feComponentTransfer.extra.content "" >

<!ENTITY % SVG.feComponentTransfer.element "INCLUDE" >
<![%SVG.feComponentTransfer.element;[
<!ENTITY % SVG.feComponentTransfer.content
    "( %SVG.feFuncR.qname;?, %SVG.feFuncG.qname;?, %SVG.feFuncB.qname;?,
       %SVG.feFuncA.qname;? %SVG.feComponentTransfer.extra.content; )"
>
<!ELEMENT %SVG.feComponentTransfer.qname; %SVG.feComponentTransfer.content; >
<!-- end of SVG.feComponentTransfer.element -->]]>

<!ENTITY % SVG.feComponentTransfer.attlist "INCLUDE" >
<![%SVG.feComponentTransfer.attlist;[
<!ATTLIST %SVG.feComponentTransfer.qname;
```

```
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>
<!-- end of SVG.feComponentTransfer.attlist -->]]>

<!-- feComposite: Filter Effect Composite Element ..... -->

<!ENTITY % SVG.feComposite.extra.content "" >

<!ENTITY % SVG.feComposite.element "INCLUDE" >
<![%SVG.feComposite.element;[
<!ENTITY % SVG.feComposite.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feComposite.extra.content; )*"
>
<!ELEMENT %SVG.feComposite.qname; %SVG.feComposite.content; >
<!-- end of SVG.feComposite.element -->]]>

<!ENTITY % SVG.feComposite.attlist "INCLUDE" >
<![%SVG.feComposite.attlist;[
<!ATTLIST %SVG.feComposite.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    operator ( over | in | out | atop | xor | arithmetic ) 'over'
    k1 %Number.datatype; #IMPLIED
    k2 %Number.datatype; #IMPLIED
    k3 %Number.datatype; #IMPLIED
    k4 %Number.datatype; #IMPLIED
>
<!-- end of SVG.feComposite.attlist -->]]>

<!-- feConvolveMatrix: Filter Effect Convolve Matrix Element -->

<!ENTITY % SVG.feConvolveMatrix.extra.content "" >

<!ENTITY % SVG.feConvolveMatrix.element "INCLUDE" >
<![%SVG.feConvolveMatrix.element;[
<!ENTITY % SVG.feConvolveMatrix.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feConvolveMatrix.extra.content; )*"
>
<!ELEMENT %SVG.feConvolveMatrix.qname; %SVG.feConvolveMatrix.content; >
<!-- end of SVG.feConvolveMatrix.element -->]]>

<!ENTITY % SVG.feConvolveMatrix.attlist "INCLUDE" >
<![%SVG.feConvolveMatrix.attlist;[
<!ATTLIST %SVG.feConvolveMatrix.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    order %NumberOptionalNumber.datatype; #IMPLIED
    kernelMatrix CDATA #REQUIRED
    divisor %Number.datatype; #IMPLIED
    bias %Number.datatype; #IMPLIED
    targetX %Integer.datatype; #IMPLIED
    targetY %Integer.datatype; #IMPLIED
    edgeMode ( duplicate | wrap | none ) 'duplicate'
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
    preserveAlpha %Boolean.datatype; #IMPLIED
>
<!-- end of SVG.feConvolveMatrix.attlist -->]]>

<!-- feDiffuseLighting: Filter Effect Diffuse Lighting Element -->

<!ENTITY % SVG.feDiffuseLighting.extra.content "" >

<!ENTITY % SVG.feDiffuseLighting.element "INCLUDE" >
<![%SVG.feDiffuseLighting.element;[
<!ENTITY % SVG.feDiffuseLighting.content
    "(( %SVG.feDistantLight.qname; | %SVG.fePointLight.qname;
      | %SVG.feSpotLight.qname; ), ( %SVG.animate.qname; | %SVG.set.qname;
      | %SVG.animateColor.qname; %SVG.feDiffuseLighting.extra.content; )*)"
```

```
>
<!ELEMENT %SVG.feDiffuseLighting.qname; %SVG.feDiffuseLighting.content; >
<!-- end of SVG.feDiffuseLighting.element -->]]>

<!ENTITY % SVG.feDiffuseLighting.attlist "INCLUDE" >
<![%SVG.feDiffuseLighting.attlist;[
<!ATTLIST %SVG.feDiffuseLighting.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    surfaceScale %Number.datatype; #IMPLIED
    diffuseConstant %Number.datatype; #IMPLIED
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
>
<!-- end of SVG.feDiffuseLighting.attlist -->]]>

<!-- feDisplacementMap: Filter Effect Displacement Map Element -->

<!ENTITY % SVG.feDisplacementMap.extra.content "" >

<!ENTITY % SVG.feDisplacementMap.element "INCLUDE" >
<![%SVG.feDisplacementMap.element;[
<!ENTITY % SVG.feDisplacementMap.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feDisplacementMap.extra.content; )*"
>
<!ELEMENT %SVG.feDisplacementMap.qname; %SVG.feDisplacementMap.content; >
<!-- end of SVG.feDisplacementMap.element -->]]>

<!ENTITY % SVG.feDisplacementMap.attlist "INCLUDE" >
<![%SVG.feDisplacementMap.attlist;[
<!ATTLIST %SVG.feDisplacementMap.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    scale %Number.datatype; #IMPLIED
    xChannelSelector ( R | G | B | A ) 'A'
    yChannelSelector ( R | G | B | A ) 'A'
>
<!-- end of SVG.feDisplacementMap.attlist -->]]>

<!-- feFlood: Filter Effect Flood Element ............. -->

<!ENTITY % SVG.feFlood.extra.content "" >

<!ENTITY % SVG.feFlood.element "INCLUDE" >
<![%SVG.feFlood.element;[
<!ENTITY % SVG.feFlood.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
       %SVG.feFlood.extra.content; )*"
>
<!ELEMENT %SVG.feFlood.qname; %SVG.feFlood.content; >
<!-- end of SVG.feFlood.element -->]]>

<!ENTITY % SVG.feFlood.attlist "INCLUDE" >
<![%SVG.feFlood.attlist;[
<!ATTLIST %SVG.feFlood.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
>
<!-- end of SVG.feFlood.attlist -->]]>

<!-- feGaussianBlur: Filter Effect Gaussian Blur Element -->

<!ENTITY % SVG.feGaussianBlur.extra.content "" >

<!ENTITY % SVG.feGaussianBlur.element "INCLUDE" >
<![%SVG.feGaussianBlur.element;[
<!ENTITY % SVG.feGaussianBlur.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feGaussianBlur.extra.content; )*"
>
```

```
<!ELEMENT %SVG.feGaussianBlur.qname; %SVG.feGaussianBlur.content; >
<!-- end of SVG.feGaussianBlur.element -->]]>

<!ENTITY % SVG.feGaussianBlur.attlist "INCLUDE" >
<![%SVG.feGaussianBlur.attlist;[
<!ATTLIST %SVG.feGaussianBlur.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    stdDeviation %NumberOptionalNumber.datatype; #IMPLIED
>
<!-- end of SVG.feGaussianBlur.attlist -->]]>

<!-- feImage: Filter Effect Image Element .............. -->

<!ENTITY % SVG.feImage.extra.content "" >

<!ENTITY % SVG.feImage.element "INCLUDE" >
<![%SVG.feImage.element;[
<!ENTITY % SVG.feImage.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateTransform.qname;
       %SVG.feImage.extra.content; )*"
>
<!ELEMENT %SVG.feImage.qname; %SVG.feImage.content; >
<!-- end of SVG.feImage.element -->]]>

<!ENTITY % SVG.feImage.attlist "INCLUDE" >
<![%SVG.feImage.attlist;[
<!ATTLIST %SVG.feImage.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
<!-- end of SVG.feImage.attlist -->]]>

<!-- feMerge: Filter Effect Merge Element .............. -->

<!ENTITY % SVG.feMerge.extra.content "" >

<!ENTITY % SVG.feMerge.element "INCLUDE" >
<![%SVG.feMerge.element;[
<!ENTITY % SVG.feMerge.content
    "( %SVG.feMergeNode.qname; %SVG.feMerge.extra.content; )*"
>
<!ELEMENT %SVG.feMerge.qname; %SVG.feMerge.content; >
<!-- end of SVG.feMerge.element -->]]>

<!ENTITY % SVG.feMerge.attlist "INCLUDE" >
<![%SVG.feMerge.attlist;[
<!ATTLIST %SVG.feMerge.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
>
<!-- end of SVG.feMerge.attlist -->]]>

<!-- feMergeNode: Filter Effect Merge Node Element ..... -->

<!ENTITY % SVG.feMergeNode.extra.content "" >

<!ENTITY % SVG.feMergeNode.element "INCLUDE" >
<![%SVG.feMergeNode.element;[
<!ENTITY % SVG.feMergeNode.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feMergeNode.extra.content; )*"
>
<!ELEMENT %SVG.feMergeNode.qname; %SVG.feMergeNode.content; >
<!-- end of SVG.feMergeNode.element -->]]>

<!ENTITY % SVG.feMergeNode.attlist "INCLUDE" >
<![%SVG.feMergeNode.attlist;[
<!ATTLIST %SVG.feMergeNode.qname;
```

```
    %SVG.Core.attrib;
    in CDATA #IMPLIED
>
<!-- end of SVG.feMergeNode.attlist -->]]>

<!-- feMorphology: Filter Effect Morphology Element .... -->

<!ENTITY % SVG.feMorphology.extra.content "" >

<!ENTITY % SVG.feMorphology.element "INCLUDE" >
<![%SVG.feMorphology.element;[
<!ENTITY % SVG.feMorphology.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feMorphology.extra.content; )*"
>
<!ELEMENT %SVG.feMorphology.qname; %SVG.feMorphology.content; >
<!-- end of SVG.feMorphology.element -->]]>

<!ENTITY % SVG.feMorphology.attlist "INCLUDE" >
<![%SVG.feMorphology.attlist;[
<!ATTLIST %SVG.feMorphology.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    operator ( erode | dilate ) 'erode'
    radius %NumberOptionalNumber.datatype; #IMPLIED
>
<!-- end of SVG.feMorphology.attlist -->]]>

<!-- feOffset: Filter Effect Offset Element ............ -->

<!ENTITY % SVG.feOffset.extra.content "" >

<!ENTITY % SVG.feOffset.element "INCLUDE" >
<![%SVG.feOffset.element;[
<!ENTITY % SVG.feOffset.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feOffset.extra.content; )*"
>
<!ELEMENT %SVG.feOffset.qname; %SVG.feOffset.content; >
<!-- end of SVG.feOffset.element -->]]>

<!ENTITY % SVG.feOffset.attlist "INCLUDE" >
<![%SVG.feOffset.attlist;[
<!ATTLIST %SVG.feOffset.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    dx %Number.datatype; #IMPLIED
    dy %Number.datatype; #IMPLIED
>
<!-- end of SVG.feOffset.attlist -->]]>

<!-- feSpecularLighting: Filter Effect Specular Lighting Element -->

<!ENTITY % SVG.feSpecularLighting.extra.content "" >

<!ENTITY % SVG.feSpecularLighting.element "INCLUDE" >
<![%SVG.feSpecularLighting.element;[
<!ENTITY % SVG.feSpecularLighting.content
    "(( %SVG.feDistantLight.qname; | %SVG.fePointLight.qname;
      | %SVG.feSpotLight.qname; ), ( %SVG.animate.qname; | %SVG.set.qname;
      | %SVG.animateColor.qname; %SVG.feSpecularLighting.extra.content; )*)"
>
<!ELEMENT %SVG.feSpecularLighting.qname; %SVG.feSpecularLighting.content; >
<!-- end of SVG.feSpecularLighting.element -->]]>

<!ENTITY % SVG.feSpecularLighting.attlist "INCLUDE" >
<![%SVG.feSpecularLighting.attlist;[
<!ATTLIST %SVG.feSpecularLighting.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    surfaceScale %Number.datatype; #IMPLIED
    specularConstant %Number.datatype; #IMPLIED
```

```
    specularExponent %Number.datatype; #IMPLIED
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
>
<!-- end of SVG.feSpecularLighting.attlist -->]]>

<!-- feTile: Filter Effect Tile Element ................ -->

<!ENTITY % SVG.feTile.extra.content "" >

<!ENTITY % SVG.feTile.element "INCLUDE" >
<![%SVG.feTile.element;[
<!ENTITY % SVG.feTile.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feTile.extra.content; )*"
>
<!ELEMENT %SVG.feTile.qname; %SVG.feTile.content; >
<!-- end of SVG.feTile.element -->]]>

<!ENTITY % SVG.feTile.attlist "INCLUDE" >
<![%SVG.feTile.attlist;[
<!ATTLIST %SVG.feTile.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>
<!-- end of SVG.feTile.attlist -->]]>

<!-- feTurbulence: Filter Effect Turbulence Element .... -->

<!ENTITY % SVG.feTurbulence.extra.content "" >

<!ENTITY % SVG.feTurbulence.element "INCLUDE" >
<![%SVG.feTurbulence.element;[
<!ENTITY % SVG.feTurbulence.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feTurbulence.extra.content; )*"
>
<!ELEMENT %SVG.feTurbulence.qname; %SVG.feTurbulence.content; >
<!-- end of SVG.feTurbulence.element -->]]>

<!ENTITY % SVG.feTurbulence.attlist "INCLUDE" >
<![%SVG.feTurbulence.attlist;[
<!ATTLIST %SVG.feTurbulence.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
    baseFrequency %NumberOptionalNumber.datatype; #IMPLIED
    numOctaves %Integer.datatype; #IMPLIED
    seed %Number.datatype; #IMPLIED
    stitchTiles ( stitch | noStitch ) 'noStitch'
    type ( fractalNoise | turbulence ) 'turbulence'
>
<!-- end of SVG.feTurbulence.attlist -->]]>

<!-- feDistantLight: Filter Effect Distant Light Element -->

<!ENTITY % SVG.feDistantLight.extra.content "" >

<!ENTITY % SVG.feDistantLight.element "INCLUDE" >
<![%SVG.feDistantLight.element;[
<!ENTITY % SVG.feDistantLight.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feDistantLight.extra.content; )*"
>
<!ELEMENT %SVG.feDistantLight.qname; %SVG.feDistantLight.content; >
<!-- end of SVG.feDistantLight.element -->]]>

<!ENTITY % SVG.feDistantLight.attlist "INCLUDE" >
<![%SVG.feDistantLight.attlist;[
<!ATTLIST %SVG.feDistantLight.qname;
    %SVG.Core.attrib;
    azimuth %Number.datatype; #IMPLIED
    elevation %Number.datatype; #IMPLIED
>
<!-- end of SVG.feDistantLight.attlist -->]]>
```

```
<!-- fePointLight: Filter Effect Point Light Element ... -->

<!ENTITY % SVG.fePointLight.extra.content "" >

<!ENTITY % SVG.fePointLight.element "INCLUDE" >
<![%SVG.fePointLight.element;[
<!ENTITY % SVG.fePointLight.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.fePointLight.extra.content; )*"
>
<!ELEMENT %SVG.fePointLight.qname; %SVG.fePointLight.content; >
<!-- end of SVG.fePointLight.element -->]]>

<!ENTITY % SVG.fePointLight.attlist "INCLUDE" >
<![%SVG.fePointLight.attlist;[
<!ATTLIST %SVG.fePointLight.qname;
    %SVG.Core.attrib;
    x %Number.datatype; #IMPLIED
    y %Number.datatype; #IMPLIED
    z %Number.datatype; #IMPLIED
>
<!-- end of SVG.fePointLight.attlist -->]]>

<!-- feSpotLight: Filter Effect Spot Light Element ..... -->

<!ENTITY % SVG.feSpotLight.extra.content "" >

<!ENTITY % SVG.feSpotLight.element "INCLUDE" >
<![%SVG.feSpotLight.element;[
<!ENTITY % SVG.feSpotLight.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feSpotLight.extra.content; )*"
>
<!ELEMENT %SVG.feSpotLight.qname; %SVG.feSpotLight.content; >
<!-- end of SVG.feSpotLight.element -->]]>

<!ENTITY % SVG.feSpotLight.attlist "INCLUDE" >
<![%SVG.feSpotLight.attlist;[
<!ATTLIST %SVG.feSpotLight.qname;
    %SVG.Core.attrib;
    x %Number.datatype; #IMPLIED
    y %Number.datatype; #IMPLIED
    z %Number.datatype; #IMPLIED
    pointsAtX %Number.datatype; #IMPLIED
    pointsAtY %Number.datatype; #IMPLIED
    pointsAtZ %Number.datatype; #IMPLIED
    specularExponent %Number.datatype; #IMPLIED
    limitingConeAngle %Number.datatype; #IMPLIED
>
<!-- end of SVG.feSpotLight.attlist -->]]>

<!-- feFuncR: Filter Effect Function Red Element ....... -->

<!ENTITY % SVG.feFuncR.extra.content "" >

<!ENTITY % SVG.feFuncR.element "INCLUDE" >
<![%SVG.feFuncR.element;[
<!ENTITY % SVG.feFuncR.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncR.extra.content; )*"
>
<!ELEMENT %SVG.feFuncR.qname; %SVG.feFuncR.content; >
<!-- end of SVG.feFuncR.element -->]]>

<!ENTITY % SVG.feFuncR.attlist "INCLUDE" >
<![%SVG.feFuncR.attlist;[
<!ATTLIST %SVG.feFuncR.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncR.attlist -->]]>

<!-- feFuncG: Filter Effect Function Green Element ..... -->
```

```
<!ENTITY % SVG.feFuncG.extra.content "" >

<!ENTITY % SVG.feFuncG.element "INCLUDE" >
<![%SVG.feFuncG.element;[
<!ENTITY % SVG.feFuncG.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncG.extra.content; )*"
>
<!ELEMENT %SVG.feFuncG.qname; %SVG.feFuncG.content; >
<!-- end of SVG.feFuncG.element -->]]>

<!ENTITY % SVG.feFuncG.attlist "INCLUDE" >
<![%SVG.feFuncG.attlist;[
<!ATTLIST %SVG.feFuncG.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncG.attlist -->]]>

<!-- feFuncB: Filter Effect Function Blue Element ..... -->

<!ENTITY % SVG.feFuncB.extra.content "" >

<!ENTITY % SVG.feFuncB.element "INCLUDE" >
<![%SVG.feFuncB.element;[
<!ENTITY % SVG.feFuncB.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncB.extra.content; )*"
>
<!ELEMENT %SVG.feFuncB.qname; %SVG.feFuncB.content; >
<!-- end of SVG.feFuncB.element -->]]>

<!ENTITY % SVG.feFuncB.attlist "INCLUDE" >
<![%SVG.feFuncB.attlist;[
<!ATTLIST %SVG.feFuncB.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncB.attlist -->]]>

<!-- feFuncA: Filter Effect Function Alpha Element ..... -->

<!ENTITY % SVG.feFuncA.extra.content "" >

<!ENTITY % SVG.feFuncA.element "INCLUDE" >
<![%SVG.feFuncA.element;[
<!ENTITY % SVG.feFuncA.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncA.extra.content; )*"
>
<!ELEMENT %SVG.feFuncA.qname; %SVG.feFuncA.content; >
<!-- end of SVG.feFuncA.element -->]]>

<!ENTITY % SVG.feFuncA.attlist "INCLUDE" >
<![%SVG.feFuncA.attlist;[
<!ATTLIST %SVG.feFuncA.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncA.attlist -->]]>
```

```
<!-- end of svg-filter.mod -->
```

A.3.33 Basic Filter Module

The Basic Filter Module defines the Filter.class and FilterPrimitive.class element collections and the Filter.attrib, FilterColor.attrib, FilterPrimitive.attrib and FilterPrimitiveWithIn.attrib attribute collections.

Collection name	Elements in collection
Filter.class	filter
FilterPrimitive.class	feBlend, feFlood, feColorMatrix, feComponentTransfer, feComposite, feGaussianBlur, feImage, feMerge, feOffset, feTile

Collection name	Attributes in collection
Filter.attrib	filter
FilterColor.attrib	color-interpolation-filters
FilterPrimitive.attrib	x, y, width, height, result
FilterPrimitiveWithIn.attrib	FilterPrimitive.attrib, in

```
<!-- ....................................................................... -->
<!-- SVG 1.1 Basic Filter Module ........................................... -->
<!-- file: svg-basic-filter.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-filter.mod,v 1.4 2009/07/16 07:04:54 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Basic Filter//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-filter.mod"

     ....................................................................... -->

<!-- Basic Filter

        filter, feBlend, feColorMatrix, feComponentTransfer, feComposite,
        feFlood, feGaussianBlur, feImage, feMerge, feMergeNode, feOffset,
        feTile, feFuncR, feFuncG, feFuncB, feFuncA

     This module declares markup to provide support for filter effect.
-->

<!-- 'filter' property/attribute value (e.g., 'none', <uri>) -->
<!ENTITY % FilterValue.datatype "CDATA" >

<!-- list of <number>s, but at least one and at most two -->
<!ENTITY % NumberOptionalNumber.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.filter.qname "filter" >
<!ENTITY % SVG.feBlend.qname "feBlend" >
<!ENTITY % SVG.feColorMatrix.qname "feColorMatrix" >
<!ENTITY % SVG.feComponentTransfer.qname "feComponentTransfer" >
<!ENTITY % SVG.feComposite.qname "feComposite" >
<!ENTITY % SVG.feFlood.qname "feFlood" >
<!ENTITY % SVG.feGaussianBlur.qname "feGaussianBlur" >
```

```
<!ENTITY % SVG.feImage.qname "feImage" >
<!ENTITY % SVG.feMerge.qname "feMerge" >
<!ENTITY % SVG.feMergeNode.qname "feMergeNode" >
<!ENTITY % SVG.feOffset.qname "feOffset" >
<!ENTITY % SVG.feTile.qname "feTile" >
<!ENTITY % SVG.feFuncR.qname "feFuncR" >
<!ENTITY % SVG.feFuncG.qname "feFuncG" >
<!ENTITY % SVG.feFuncB.qname "feFuncB" >
<!ENTITY % SVG.feFuncA.qname "feFuncA" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.XLinkEmbed.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Filter.class ................................ -->

<!ENTITY % SVG.Filter.extra.class "" >

<!ENTITY % SVG.Filter.class
    "| %SVG.filter.qname; %SVG.Filter.extra.class;"
>

<!-- SVG.FilterPrimitive.class ........................ -->

<!ENTITY % SVG.FilterPrimitive.extra.class "" >

<!ENTITY % SVG.FilterPrimitive.class
    "| %SVG.feBlend.qname; | %SVG.feColorMatrix.qname;
     | %SVG.feComponentTransfer.qname; | %SVG.feComposite.qname;
     | %SVG.feFlood.qname; | %SVG.feGaussianBlur.qname; | %SVG.feImage.qname;
     | %SVG.feMerge.qname; | %SVG.feOffset.qname; | %SVG.feTile.qname;
       %SVG.FilterPrimitive.extra.class;"
>

<!-- SVG.Filter.attrib ................................ -->

<!ENTITY % SVG.Filter.extra.attrib "" >

<!ENTITY % SVG.Filter.attrib
    "filter %FilterValue.datatype; #IMPLIED
     %SVG.Filter.extra.attrib;"
>

<!-- SVG.FilterColor.attrib ........................... -->

<!ENTITY % SVG.FilterColor.extra.attrib "" >

<!ENTITY % SVG.FilterColor.attrib
    "color-interpolation-filters ( auto | sRGB | linearRGB | inherit )
                                 #IMPLIED
     %SVG.FilterColor.extra.attrib;"
>

<!-- SVG.FilterPrimitive.attrib ....................... -->

<!ENTITY % SVG.FilterPrimitive.extra.attrib "" >

<!ENTITY % SVG.FilterPrimitive.attrib
```

```
    "x %Coordinate.datatype; #IMPLIED
     y %Coordinate.datatype; #IMPLIED
     width %Length.datatype; #IMPLIED
     height %Length.datatype; #IMPLIED
     result CDATA #IMPLIED
     %SVG.FilterPrimitive.extra.attrib;"
>

<!-- SVG.FilterPrimitiveWithIn.attrib .................. -->

<!ENTITY % SVG.FilterPrimitiveWithIn.extra.attrib "" >

<!ENTITY % SVG.FilterPrimitiveWithIn.attrib
    "%SVG.FilterPrimitive.attrib;
     in CDATA #IMPLIED
     %SVG.FilterPrimitiveWithIn.extra.attrib;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- filter: Filter Element ............................ -->

<!ENTITY % SVG.filter.extra.content "" >

<!ENTITY % SVG.filter.element "INCLUDE" >
<![%SVG.filter.element;[
<!ENTITY % SVG.filter.content
    "( %SVG.Description.class; | %SVG.animate.qname; | %SVG.set.qname;
       %SVG.FilterPrimitive.class; %SVG.filter.extra.content; )*"
>
<!ELEMENT %SVG.filter.qname; %SVG.filter.content; >
<!-- end of SVG.filter.element -->]]>

<!ENTITY % SVG.filter.attlist "INCLUDE" >
<![%SVG.filter.attlist;[
<!ATTLIST %SVG.filter.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    filterRes %NumberOptionalNumber.datatype; #IMPLIED
    filterUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    primitiveUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
<!-- end of SVG.filter.attlist -->]]>
```

```
<!-- feBlend: Filter Effect Blend Element .............. -->

<!ENTITY % SVG.feBlend.extra.content "" >

<!ENTITY % SVG.feBlend.element "INCLUDE" >
<![%SVG.feBlend.element;[
<!ENTITY % SVG.feBlend.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feBlend.extra.content; )*"
>
<!ELEMENT %SVG.feBlend.qname; %SVG.feBlend.content; >
<!-- end of SVG.feBlend.element -->]]>

<!ENTITY % SVG.feBlend.attlist "INCLUDE" >
<![%SVG.feBlend.attlist;[
<!ATTLIST %SVG.feBlend.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    mode ( normal | multiply | screen | darken | lighten ) 'normal'
>
<!-- end of SVG.feBlend.attlist -->]]>

<!-- feColorMatrix: Filter Effect Color Matrix Element . -->

<!ENTITY % SVG.feColorMatrix.extra.content "" >

<!ENTITY % SVG.feColorMatrix.element "INCLUDE" >
<![%SVG.feColorMatrix.element;[
<!ENTITY % SVG.feColorMatrix.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feColorMatrix.extra.content; )*"
>
<!ELEMENT %SVG.feColorMatrix.qname; %SVG.feColorMatrix.content; >
<!-- end of SVG.feColorMatrix.element -->]]>

<!ENTITY % SVG.feColorMatrix.attlist "INCLUDE" >
<![%SVG.feColorMatrix.attlist;[
<!ATTLIST %SVG.feColorMatrix.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    type ( matrix | saturate | hueRotate | luminanceToAlpha ) 'matrix'
    values CDATA #IMPLIED
>
<!-- end of SVG.feColorMatrix.attlist -->]]>

<!-- feComponentTransfer: Filter Effect Component Transfer Element -->

<!ENTITY % SVG.feComponentTransfer.extra.content "" >

<!ENTITY % SVG.feComponentTransfer.element "INCLUDE" >
<![%SVG.feComponentTransfer.element;[
<!ENTITY % SVG.feComponentTransfer.content
    "( %SVG.feFuncR.qname;?, %SVG.feFuncG.qname;?, %SVG.feFuncB.qname;?,
       %SVG.feFuncA.qname;? %SVG.feComponentTransfer.extra.content; )"
>
<!ELEMENT %SVG.feComponentTransfer.qname; %SVG.feComponentTransfer.content; >
<!-- end of SVG.feComponentTransfer.element -->]]>

<!ENTITY % SVG.feComponentTransfer.attlist "INCLUDE" >
<![%SVG.feComponentTransfer.attlist;[
<!ATTLIST %SVG.feComponentTransfer.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>
<!-- end of SVG.feComponentTransfer.attlist -->]]>

<!-- feComposite: Filter Effect Composite Element ...... -->

<!ENTITY % SVG.feComposite.extra.content "" >

<!ENTITY % SVG.feComposite.element "INCLUDE" >
```

```
<![%SVG.feComposite.element;[
<!ENTITY % SVG.feComposite.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feComposite.extra.content; )*"
>
<!ELEMENT %SVG.feComposite.qname; %SVG.feComposite.content; >
<!-- end of SVG.feComposite.element -->]]>

<!ENTITY % SVG.feComposite.attlist "INCLUDE" >
<![%SVG.feComposite.attlist;[
<!ATTLIST %SVG.feComposite.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    operator ( over | in | out | atop | xor | arithmetic ) 'over'
    k1 %Number.datatype; #IMPLIED
    k2 %Number.datatype; #IMPLIED
    k3 %Number.datatype; #IMPLIED
    k4 %Number.datatype; #IMPLIED
>
<!-- end of SVG.feComposite.attlist -->]]>

<!-- feFlood: Filter Effect Flood Element ..............  -->

<!ENTITY % SVG.feFlood.extra.content "" >

<!ENTITY % SVG.feFlood.element "INCLUDE" >
<![%SVG.feFlood.element;[
<!ENTITY % SVG.feFlood.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
       %SVG.feFlood.extra.content; )*"
>
<!ELEMENT %SVG.feFlood.qname; %SVG.feFlood.content; >
<!-- end of SVG.feFlood.element -->]]>

<!ENTITY % SVG.feFlood.attlist "INCLUDE" >
<![%SVG.feFlood.attlist;[
<!ATTLIST %SVG.feFlood.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Color.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitive.attrib;
    flood-color %SVGColor.datatype; #IMPLIED
    flood-opacity %OpacityValue.datatype; #IMPLIED
>
<!-- end of SVG.feFlood.attlist -->]]>

<!-- feGaussianBlur: Filter Effect Gaussian Blur Element -->

<!ENTITY % SVG.feGaussianBlur.extra.content "" >

<!ENTITY % SVG.feGaussianBlur.element "INCLUDE" >
<![%SVG.feGaussianBlur.element;[
<!ENTITY % SVG.feGaussianBlur.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feGaussianBlur.extra.content; )*"
>
<!ELEMENT %SVG.feGaussianBlur.qname; %SVG.feGaussianBlur.content; >
<!-- end of SVG.feGaussianBlur.element -->]]>

<!ENTITY % SVG.feGaussianBlur.attlist "INCLUDE" >
<![%SVG.feGaussianBlur.attlist;[
<!ATTLIST %SVG.feGaussianBlur.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    stdDeviation %NumberOptionalNumber.datatype; #IMPLIED
>
<!-- end of SVG.feGaussianBlur.attlist -->]]>

<!-- feImage: Filter Effect Image Element ..............  -->

<!ENTITY % SVG.feImage.extra.content "" >

<!ENTITY % SVG.feImage.element "INCLUDE" >
```

```
<![%SVG.feImage.element;[
<!ENTITY % SVG.feImage.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateTransform.qname;
       %SVG.feImage.extra.content; )*"
>
<!ELEMENT %SVG.feImage.qname; %SVG.feImage.content; >
<!-- end of SVG.feImage.element -->]]>

<!ENTITY % SVG.feImage.attlist "INCLUDE" >
<![%SVG.feImage.attlist;[
<!ATTLIST %SVG.feImage.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
<!-- end of SVG.feImage.attlist -->]]>

<!-- feMerge: Filter Effect Merge Element .............. -->

<!ENTITY % SVG.feMerge.extra.content "" >

<!ENTITY % SVG.feMerge.element "INCLUDE" >
<![%SVG.feMerge.element;[
<!ENTITY % SVG.feMerge.content
    "( %SVG.feMergeNode.qname; %SVG.feMerge.extra.content; )*"
>
<!ELEMENT %SVG.feMerge.qname; %SVG.feMerge.content; >
<!-- end of SVG.feMerge.element -->]]>

<!ENTITY % SVG.feMerge.attlist "INCLUDE" >
<![%SVG.feMerge.attlist;[
<!ATTLIST %SVG.feMerge.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitive.attrib;
>
<!-- end of SVG.feMerge.attlist -->]]>

<!-- feMergeNode: Filter Effect Merge Node Element ..... -->

<!ENTITY % SVG.feMergeNode.extra.content "" >

<!ENTITY % SVG.feMergeNode.element "INCLUDE" >
<![%SVG.feMergeNode.element;[
<!ENTITY % SVG.feMergeNode.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feMergeNode.extra.content; )*"
>
<!ELEMENT %SVG.feMergeNode.qname; %SVG.feMergeNode.content; >
<!-- end of SVG.feMergeNode.element -->]]>

<!ENTITY % SVG.feMergeNode.attlist "INCLUDE" >
<![%SVG.feMergeNode.attlist;[
<!ATTLIST %SVG.feMergeNode.qname;
    %SVG.Core.attrib;
    in CDATA #IMPLIED
>
<!-- end of SVG.feMergeNode.attlist -->]]>

<!-- feOffset: Filter Effect Offset Element ............ -->

<!ENTITY % SVG.feOffset.extra.content "" >

<!ENTITY % SVG.feOffset.element "INCLUDE" >
<![%SVG.feOffset.element;[
<!ENTITY % SVG.feOffset.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feOffset.extra.content; )*"
>
<!ELEMENT %SVG.feOffset.qname; %SVG.feOffset.content; >
<!-- end of SVG.feOffset.element -->]]>

<!ENTITY % SVG.feOffset.attlist "INCLUDE" >
<![%SVG.feOffset.attlist;[
<!ATTLIST %SVG.feOffset.qname;
```

```
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    dx %Number.datatype; #IMPLIED
    dy %Number.datatype; #IMPLIED
>
<!-- end of SVG.feOffset.attlist -->]]>

<!-- feTile: Filter Effect Tile Element ................ -->

<!ENTITY % SVG.feTile.extra.content "" >

<!ENTITY % SVG.feTile.element "INCLUDE" >
<![%SVG.feTile.element;[
<!ENTITY % SVG.feTile.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feTile.extra.content; )*"
>
<!ELEMENT %SVG.feTile.qname; %SVG.feTile.content; >
<!-- end of SVG.feTile.element -->]]>

<!ENTITY % SVG.feTile.attlist "INCLUDE" >
<![%SVG.feTile.attlist;[
<!ATTLIST %SVG.feTile.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>
<!-- end of SVG.feTile.attlist -->]]>

<!-- feFuncR: Filter Effect Function Red Element ....... -->

<!ENTITY % SVG.feFuncR.extra.content "" >

<!ENTITY % SVG.feFuncR.element "INCLUDE" >
<![%SVG.feFuncR.element;[
<!ENTITY % SVG.feFuncR.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncR.extra.content; )*"
>
<!ELEMENT %SVG.feFuncR.qname; %SVG.feFuncR.content; >
<!-- end of SVG.feFuncR.element -->]]>

<!ENTITY % SVG.feFuncR.attlist "INCLUDE" >
<![%SVG.feFuncR.attlist;[
<!ATTLIST %SVG.feFuncR.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncR.attlist -->]]>

<!-- feFuncG: Filter Effect Function Green Element ..... -->

<!ENTITY % SVG.feFuncG.extra.content "" >

<!ENTITY % SVG.feFuncG.element "INCLUDE" >
<![%SVG.feFuncG.element;[
<!ENTITY % SVG.feFuncG.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncG.extra.content; )*"
>
<!ELEMENT %SVG.feFuncG.qname; %SVG.feFuncG.content; >
<!-- end of SVG.feFuncG.element -->]]>

<!ENTITY % SVG.feFuncG.attlist "INCLUDE" >
<![%SVG.feFuncG.attlist;[
<!ATTLIST %SVG.feFuncG.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
```

```
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncG.attlist -->]]>

<!-- feFuncB: Filter Effect Function Blue Element ..... -->

<!ENTITY % SVG.feFuncB.extra.content "" >

<!ENTITY % SVG.feFuncB.element "INCLUDE" >
<![%SVG.feFuncB.element;[
<!ENTITY % SVG.feFuncB.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncB.extra.content; )*"
>
<!ELEMENT %SVG.feFuncB.qname; %SVG.feFuncB.content; >
<!-- end of SVG.feFuncB.element -->]]>

<!ENTITY % SVG.feFuncB.attlist "INCLUDE" >
<![%SVG.feFuncB.attlist;[
<!ATTLIST %SVG.feFuncB.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncB.attlist -->]]>

<!-- feFuncA: Filter Effect Function Alpha Element ..... -->

<!ENTITY % SVG.feFuncA.extra.content "" >

<!ENTITY % SVG.feFuncA.element "INCLUDE" >
<![%SVG.feFuncA.element;[
<!ENTITY % SVG.feFuncA.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncA.extra.content; )*"
>
<!ELEMENT %SVG.feFuncA.qname; %SVG.feFuncA.content; >
<!-- end of SVG.feFuncA.element -->]]>

<!ENTITY % SVG.feFuncA.attlist "INCLUDE" >
<![%SVG.feFuncA.attlist;[
<!ATTLIST %SVG.feFuncA.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncA.attlist -->]]>

<!-- end of svg-basic-filter.mod -->
```

A.3.34 Cursor Module

The Cursor Module defines the Cursor.class element collection and the Cursor.attrib attribute collection.

Collection name	Elements in collection
Cursor.class	cursor

Collection name	Elements in collection
Cursor.attrib	cursor

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Cursor Module ................................................ -->
<!-- file: svg-cursor.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-cursor.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Cursor//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-cursor.mod"

     ..................................................................... -->

<!-- Cursor

        cursor

     This module declares markup to provide support for cursor.
-->

<!-- 'cursor' property/attribute value (e.g., 'crosshair', <uri>) -->
<!ENTITY % CursorValue.datatype "CDATA" >

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.cursor.qname "cursor" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.XLinkRequired.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Cursor.class ................................. -->

<!ENTITY % SVG.Cursor.extra.class "" >

<!ENTITY % SVG.Cursor.class
    "| %SVG.cursor.qname; %SVG.Cursor.extra.class;"
>

<!-- SVG.Cursor.attrib ................................ -->

<!ENTITY % SVG.Cursor.extra.attrib "" >

<!ENTITY % SVG.Cursor.attrib
    "cursor %CursorValue.datatype; #IMPLIED
     %SVG.Cursor.extra.attrib;"
>

<!-- cursor: Cursor Element ........................... -->

<!ENTITY % SVG.cursor.extra.content "" >

<!ENTITY % SVG.cursor.element "INCLUDE" >
<![%SVG.cursor.element;[
<!ENTITY % SVG.cursor.content
    "( %SVG.Description.class; %SVG.cursor.extra.content; )*"
>
<!ELEMENT %SVG.cursor.qname; %SVG.cursor.content; >
<!-- end of SVG.cursor.element -->]]>

<!ENTITY % SVG.cursor.attlist "INCLUDE" >
<![%SVG.cursor.attlist;[
<!ATTLIST %SVG.cursor.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.XLinkRequired.attrib;
```

```
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
>
<!-- end of SVG.cursor.attlist -->]]>

<!-- end of svg-cursor.mod -->
```

A.3.35 Hyperlinking Module

The Hyperlinking Module defines the Hyperlink.class element collection.

Collection name	Elements in collection
Hyperlink.class	a

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Hyperlinking Module ........................................ -->
<!-- file: svg-hyperlink.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-hyperlink.mod,v 1.2 2009/05/30 04:21:38 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Hyperlinking//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-hyperlink.mod"

     ..................................................................... -->

<!-- Hyperlinking

        a

     This module declares markup to provide support for hyper linking.
-->

<!-- link to this target -->
<!ENTITY % LinkTarget.datatype "NMTOKEN" >

<!-- Qualified Names (Default) ......................... -->

<!ENTITY % SVG.a.qname "a" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkReplace.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Hyperlink.class ............................... -->
```

```
<!ENTITY % SVG.Hyperlink.extra.class "" >

<!ENTITY % SVG.Hyperlink.class
    "| %SVG.a.qname; %SVG.Hyperlink.extra.class;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- a: Anchor Element ................................ -->

<!ENTITY % SVG.a.extra.content "" >

<!ENTITY % SVG.a.element "INCLUDE" >
<![%SVG.a.element;[
<!ENTITY % SVG.a.content
    "( #PCDATA | %SVG.Description.class; | %SVG.Animation.class;
       %SVG.Structure.class; %SVG.Conditional.class; %SVG.Image.class;
       %SVG.Style.class; %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.a.extra.content; )*"
>
<!ELEMENT %SVG.a.qname; %SVG.a.content; >
<!-- end of SVG.a.element -->]]>

<!ENTITY % SVG.a.attlist "INCLUDE" >
<![%SVG.a.attlist;[
<!ATTLIST %SVG.a.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
    %SVG.XLinkReplace.attrib;
    %SVG.External.attrib;
    transform %TransformList.datatype; #IMPLIED
    target %LinkTarget.datatype; #IMPLIED
>
<!-- end of SVG.a.attlist -->]]>

<!-- end of svg-hyperlink.mod -->
```

A.3.36 View Module

The View Module defines the View.class element collection.

Collection name	Elements in collection
View.class	view

```
<!-- ..................................................................... -->
<!-- SVG 1.1 View Module .................................................. -->
<!-- file: svg-view.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-view.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 View//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-view.mod"

     ..................................................................... -->

<!-- View

        view

     This module declares markup to provide support for view.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.view.qname "view" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.View.class ................................... -->

<!ENTITY % SVG.View.extra.class "" >

<!ENTITY % SVG.View.class
    "| %SVG.view.qname; %SVG.View.extra.class;"
>

<!-- view: View Element ............................... -->

<!ENTITY % SVG.view.extra.content "" >

<!ENTITY % SVG.view.element "INCLUDE" >
<![%SVG.view.element;[
<!ENTITY % SVG.view.content
    "( %SVG.Description.class; %SVG.view.extra.content; )*"
>
<!ELEMENT %SVG.view.qname; %SVG.view.content; >
<!-- end of SVG.view.element -->]]>

<!ENTITY % SVG.view.attlist "INCLUDE" >
<![%SVG.view.attlist;[
<!ATTLIST %SVG.view.qname;
    %SVG.Core.attrib;
    %SVG.External.attrib;
    viewBox %ViewBoxSpec.datatype; #IMPLIED
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
    zoomAndPan ( disable | magnify ) 'magnify'
    viewTarget CDATA #IMPLIED
>
<!-- end of SVG.view.attlist -->]]>

<!-- end of svg-view.mod -->
```

A.3.37 Scripting Module

The Script Module defines the Script.class element collection.

Collection name	Elements in collection
Script.class	script

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Scripting Module ............................................ -->
<!-- file: svg-script.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-script.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Scripting//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-script.mod"

     ..................................................................... -->

<!-- Scripting

        script

     This module declares markup to provide support for scripting.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.script.qname "script" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Script.class ................................. -->

<!ENTITY % SVG.Script.extra.class "" >

<!ENTITY % SVG.Script.class
    "| %SVG.script.qname; %SVG.Script.extra.class;"
>

<!-- script: Script Element ........................... -->

<!ENTITY % SVG.script.extra.content "" >

<!ENTITY % SVG.script.element "INCLUDE" >
<![%SVG.script.element;[
<!ENTITY % SVG.script.content
    "( #PCDATA %SVG.script.extra.content; )*"
>
<!ELEMENT %SVG.script.qname; %SVG.script.content; >
<!-- end of SVG.script.element -->]]>

<!ENTITY % SVG.script.attlist "INCLUDE" >
<![%SVG.script.attlist;[
<!ATTLIST %SVG.script.qname;
    %SVG.Core.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    type %ContentType.datatype; #REQUIRED
>
<!-- end of SVG.script.attlist -->]]>

<!-- end of svg-script.mod -->
```

A.3.38 Animation Module

The Animation Module defines the Animation.class element collection and the Animation.attrib, AnimationAttribute.attrib, AnimationTiming.attrib, AnimationValue.attrib and AnimationAddtion.attrib attribute collections.

Collection name	Elements in collection
Animation.class	animate, animateColor, animateTransform, animateMotion, set

Collection name	Attributes in collection
Animation.attrib	XLink.attrib
AnimationAttribute.attrib	attributeName, attributeType
AnimationTiming.attrib	begin, dur, end, min, max, restart, repeatCount, repeatDur, fill
AnimationValue.attrib	calcMode, values, keyTimes, keySplines, from, to, by
AnimationAddition.attrib	additive, accumulate

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Animation Module ............................................ -->
<!-- file: svg-animation.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-animation.mod,v 1.3 2009/06/11 17:20:55 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Animation//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-animation.mod"

     ..................................................................... -->

<!-- Animation

        animate, set, animateMotion, animateColor, animateTransform, mpath

     This module declares markup to provide support for animation.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.animate.qname "animate" >
<!ENTITY % SVG.set.qname "set" >
<!ENTITY % SVG.animateMotion.qname "animateMotion" >
<!ENTITY % SVG.animateColor.qname "animateColor" >
<!ENTITY % SVG.animateTransform.qname "animateTransform" >
<!ENTITY % SVG.mpath.qname "mpath" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.AnimationEvents.attrib "" >
<!ENTITY % SVG.XLink.attrib "" >
<!ENTITY % SVG.XLinkRequired.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Animation.class .............................. -->

<!ENTITY % SVG.Animation.extra.class "" >
```

```
<!ENTITY % SVG.Animation.class
    "%SVG.animate.qname; | %SVG.set.qname; | %SVG.animateMotion.qname; |
     %SVG.animateColor.qname; | %SVG.animateTransform.qname;
     %SVG.Animation.extra.class;"
>

<!-- SVG.Animation.attrib ............................ -->

<!ENTITY % SVG.Animation.extra.attrib "" >

<!ENTITY % SVG.Animation.attrib
    "%SVG.XLink.attrib;
     %SVG.Animation.extra.attrib;"
>

<!-- SVG.AnimationAttribute.attrib ..................... -->

<!ENTITY % SVG.AnimationAttribute.extra.attrib "" >

<!ENTITY % SVG.AnimationAttribute.attrib
    "attributeName  CDATA  #REQUIRED
     attributeType  CDATA  #IMPLIED
     %SVG.AnimationAttribute.extra.attrib;"
>

<!-- SVG.AnimationTiming.attrib ....................... -->

<!ENTITY % SVG.AnimationTiming.extra.attrib "" >

<!ENTITY % SVG.AnimationTiming.attrib
    "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'
     %SVG.AnimationTiming.extra.attrib;"
>

<!-- SVG.AnimationValue.attrib ........................ -->

<!ENTITY % SVG.AnimationValue.extra.attrib "" >

<!ENTITY % SVG.AnimationValue.attrib
    "calcMode ( discrete | linear | paced | spline ) 'linear'
     values CDATA #IMPLIED
     keyTimes CDATA #IMPLIED
     keySplines CDATA #IMPLIED
     from CDATA #IMPLIED
     to CDATA #IMPLIED
     by CDATA #IMPLIED
     %SVG.AnimationValue.extra.attrib;"
>

<!-- SVG.AnimationAddtion.attrib ...................... -->

<!ENTITY % SVG.AnimationAddtion.extra.attrib "" >

<!ENTITY % SVG.AnimationAddtion.attrib
    "additive ( replace | sum ) 'replace'
     accumulate ( none | sum ) 'none'
     %SVG.AnimationAddtion.extra.attrib;"
>

<!-- animate: Animate Element ......................... -->

<!ENTITY % SVG.animate.extra.content "" >

<!ENTITY % SVG.animate.element "INCLUDE" >
<![%SVG.animate.element;[
<!ENTITY % SVG.animate.content
    "( %SVG.Description.class; %SVG.animate.extra.content; )*"
>
<!ELEMENT %SVG.animate.qname; %SVG.animate.content; >
```

```
<!-- end of SVG.animate.element -->]]>

<!ENTITY % SVG.animate.attlist "INCLUDE" >
<![%SVG.animate.attlist;[
<!ATTLIST %SVG.animate.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.AnimationEvents.attrib;
    %SVG.External.attrib;
    %SVG.Animation.attrib;
    %SVG.AnimationAttribute.attrib;
    %SVG.AnimationTiming.attrib;
    %SVG.AnimationValue.attrib;
    %SVG.AnimationAddtion.attrib;
>
<!-- end of SVG.animate.attlist -->]]>

<!-- set: Set Element ................................ -->

<!ENTITY % SVG.set.extra.content "" >

<!ENTITY % SVG.set.element "INCLUDE" >
<![%SVG.set.element;[
<!ENTITY % SVG.set.content
    "( %SVG.Description.class; %SVG.set.extra.content; )*"
>
<!ELEMENT %SVG.set.qname; %SVG.set.content; >
<!-- end of SVG.set.element -->]]>

<!ENTITY % SVG.set.attlist "INCLUDE" >
<![%SVG.set.attlist;[
<!ATTLIST %SVG.set.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.AnimationEvents.attrib;
    %SVG.External.attrib;
    %SVG.Animation.attrib;
    %SVG.AnimationAttribute.attrib;
    %SVG.AnimationTiming.attrib;
    to CDATA #IMPLIED
>
<!-- end of SVG.set.attlist -->]]>

<!-- animateMotion: Animate Motion Element ............. -->

<!ENTITY % SVG.animateMotion.extra.content "" >

<!ENTITY % SVG.animateMotion.element "INCLUDE" >
<![%SVG.animateMotion.element;[
<!ENTITY % SVG.animateMotion.content
    "( ( %SVG.mpath.qname;,
         ( %SVG.Description.class; %SVG.animateMotion.extra.content; )* )
       | ( ( %SVG.Description.class; %SVG.animateMotion.extra.content; )+,
           %SVG.mpath.qname;,
           ( ( %SVG.Description.class; %SVG.animateMotion.extra.content; )* )? ) )"
>
<!ELEMENT %SVG.animateMotion.qname; %SVG.animateMotion.content; >
<!-- end of SVG.animateMotion.element -->]]>

<!ENTITY % SVG.animateMotion.attlist "INCLUDE" >
<![%SVG.animateMotion.attlist;[
<!ATTLIST %SVG.animateMotion.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.AnimationEvents.attrib;
    %SVG.External.attrib;
    %SVG.Animation.attrib;
    %SVG.AnimationTiming.attrib;
    %SVG.AnimationAddtion.attrib;
    calcMode ( discrete | linear | paced | spline ) 'paced'
    values CDATA #IMPLIED
    keyTimes CDATA #IMPLIED
    keySplines CDATA #IMPLIED
    from CDATA #IMPLIED
    to CDATA #IMPLIED
    by CDATA #IMPLIED
    path CDATA #IMPLIED
```

```
    keyPoints CDATA #IMPLIED
    rotate CDATA #IMPLIED
    origin CDATA #IMPLIED
>
<!-- end of SVG.animateMotion.attlist -->]]>

<!-- animateColor: Animate Color Element ............... -->

<!ENTITY % SVG.animateColor.extra.content "" >

<!ENTITY % SVG.animateColor.element "INCLUDE" >
<![%SVG.animateColor.element;[
<!ENTITY % SVG.animateColor.content
    "( %SVG.Description.class; %SVG.animateColor.extra.content; )*"
>
<!ELEMENT %SVG.animateColor.qname; %SVG.animateColor.content; >
<!-- end of SVG.animateColor.element -->]]>

<!ENTITY % SVG.animateColor.attlist "INCLUDE" >
<![%SVG.animateColor.attlist;[
<!ATTLIST %SVG.animateColor.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.AnimationEvents.attrib;
    %SVG.External.attrib;
    %SVG.Animation.attrib;
    %SVG.AnimationAttribute.attrib;
    %SVG.AnimationTiming.attrib;
    %SVG.AnimationValue.attrib;
    %SVG.AnimationAddtion.attrib;
>
<!-- end of SVG.animateColor.attlist -->]]>

<!-- animateTransform: Animate Transform Element ....... -->

<!ENTITY % SVG.animateTransform.extra.content "" >

<!ENTITY % SVG.animateTransform.element "INCLUDE" >
<![%SVG.animateTransform.element;[
<!ENTITY % SVG.animateTransform.content
    "( %SVG.Description.class; %SVG.animateTransform.extra.content; )*"
>
<!ELEMENT %SVG.animateTransform.qname; %SVG.animateTransform.content; >
<!-- end of SVG.animateTransform.element -->]]>

<!ENTITY % SVG.animateTransform.attlist "INCLUDE" >
<![%SVG.animateTransform.attlist;[
<!ATTLIST %SVG.animateTransform.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.AnimationEvents.attrib;
    %SVG.External.attrib;
    %SVG.Animation.attrib;
    %SVG.AnimationAttribute.attrib;
    %SVG.AnimationTiming.attrib;
    %SVG.AnimationValue.attrib;
    %SVG.AnimationAddtion.attrib;
    type ( translate | scale | rotate | skewX | skewY ) 'translate'
>
<!-- end of SVG.animateTransform.attlist -->]]>

<!-- mpath: Motion Path Element ........................ -->

<!ENTITY % SVG.mpath.extra.content "" >

<!ENTITY % SVG.mpath.element "INCLUDE" >
<![%SVG.mpath.element;[
<!ENTITY % SVG.mpath.content
    "( %SVG.Description.class; %SVG.mpath.extra.content; )*"
>
<!ELEMENT %SVG.mpath.qname; %SVG.mpath.content; >
<!-- end of SVG.mpath.element -->]]>

<!ENTITY % SVG.mpath.attlist "INCLUDE" >
<![%SVG.mpath.attlist;[
<!ATTLIST %SVG.mpath.qname;
    %SVG.Core.attrib;
```

```
    %SVG.XLinkRequired.attrib;
    %SVG.External.attrib;
>
<!-- end of SVG.mpath.attlist -->]]>

<!-- end of svg-animation.mod -->
```

A.3.39 Font Module

The Font Module defines the Font.class element collection.

Collection name	Elements in collection
Font.class	font, font-face

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Font Module .................................................. -->
<!-- file: svg-font.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-font.mod,v 1.5 2009/06/11 17:20:55 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Font//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-font.mod"

     ..................................................................... -->

<!-- Font

        font, font-face, glyph, missing-glyph, hkern, vkern, font-face-src,
        font-face-uri, font-face-format, font-face-name

     This module declares markup to provide support for template.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.font.qname "font" >
<!ENTITY % SVG.font-face.qname "font-face" >
<!ENTITY % SVG.glyph.qname "glyph" >
<!ENTITY % SVG.missing-glyph.qname "missing-glyph" >
<!ENTITY % SVG.hkern.qname "hkern" >
<!ENTITY % SVG.vkern.qname "vkern" >
<!ENTITY % SVG.font-face-src.qname "font-face-src" >
<!ENTITY % SVG.font-face-uri.qname "font-face-uri" >
<!ENTITY % SVG.font-face-format.qname "font-face-format" >
<!ENTITY % SVG.font-face-name.qname "font-face-name" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
```

```
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkRequired.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Font.class ................................... -->

<!ENTITY % SVG.Font.extra.class "" >

<!ENTITY % SVG.Font.class
    "| %SVG.font.qname; | %SVG.font-face.qname; %SVG.Font.extra.class;"
>

<!-- SVG.Presentation.attrib .......................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- font: Font Element ............................... -->

<!ENTITY % SVG.font.extra.content "" >

<!ENTITY % SVG.font.element "INCLUDE" >
<![%SVG.font.element;[
<!ENTITY % SVG.font.content
    "( %SVG.Description.class; | %SVG.font-face.qname;
     | %SVG.missing-glyph.qname; | %SVG.glyph.qname; | %SVG.hkern.qname;
     | %SVG.vkern.qname; %SVG.font.extra.content; )*"
>
<!ELEMENT %SVG.font.qname; %SVG.font.content; >
<!-- end of SVG.font.element -->]]>

<!ENTITY % SVG.font.attlist "INCLUDE" >
<![%SVG.font.attlist;[
<!ATTLIST %SVG.font.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.External.attrib;
    horiz-origin-x %Number.datatype; #IMPLIED
    horiz-origin-y %Number.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #REQUIRED
    vert-origin-x %Number.datatype; #IMPLIED
    vert-origin-y %Number.datatype; #IMPLIED
    vert-adv-y %Number.datatype; #IMPLIED
>
<!-- end of SVG.font.attlist -->]]>

<!-- font-face: Font Face Element ..................... -->

<!ENTITY % SVG.font-face.extra.content "" >

<!ENTITY % SVG.font-face.element "INCLUDE" >
<![%SVG.font-face.element;[
```

```
<!ENTITY % SVG.font-face.content
    "( ( %SVG.font-face-src.qname;,
         ( %SVG.Description.class; %SVG.font-face.extra.content; )* )
       | ( ( %SVG.Description.class; %SVG.font-face.extra.content; )+,
           %SVG.font-face-src.qname;,
           ( ( %SVG.Description.class; %SVG.font-face.extra.content; )* )? ) )"
>
<!ELEMENT %SVG.font-face.qname; %SVG.font-face.content; >
<!-- end of SVG.font-face.element -->]]>

<!ENTITY % SVG.font-face.attlist "INCLUDE" >
<![%SVG.font-face.attlist;[
<!ATTLIST %SVG.font-face.qname;
    %SVG.Core.attrib;
    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.datatype; #IMPLIED
    panose-1 CDATA #IMPLIED
    stemv %Number.datatype; #IMPLIED
    stemh %Number.datatype; #IMPLIED
    slope %Number.datatype; #IMPLIED
    cap-height %Number.datatype; #IMPLIED
    x-height %Number.datatype; #IMPLIED
    accent-height %Number.datatype; #IMPLIED
    ascent %Number.datatype; #IMPLIED
    descent %Number.datatype; #IMPLIED
    widths CDATA #IMPLIED
    bbox CDATA #IMPLIED
    ideographic %Number.datatype; #IMPLIED
    alphabetic %Number.datatype; #IMPLIED
    mathematical %Number.datatype; #IMPLIED
    hanging %Number.datatype; #IMPLIED
    v-ideographic %Number.datatype; #IMPLIED
    v-alphabetic %Number.datatype; #IMPLIED
    v-mathematical %Number.datatype; #IMPLIED
    v-hanging %Number.datatype; #IMPLIED
    underline-position %Number.datatype; #IMPLIED
    underline-thickness %Number.datatype; #IMPLIED
    strikethrough-position %Number.datatype; #IMPLIED
    strikethrough-thickness %Number.datatype; #IMPLIED
    overline-position %Number.datatype; #IMPLIED
    overline-thickness %Number.datatype; #IMPLIED
>
<!-- end of SVG.font-face.attlist -->]]>

<!-- glyph: Glyph Element ............................. -->

<!ENTITY % SVG.glyph.extra.content "" >

<!ENTITY % SVG.glyph.element "INCLUDE" >
<![%SVG.glyph.element;[
<!ENTITY % SVG.glyph.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.glyph.extra.content; )*"
>
<!ELEMENT %SVG.glyph.qname; %SVG.glyph.content; >
<!-- end of SVG.glyph.element -->]]>

<!ENTITY % SVG.glyph.attlist "INCLUDE" >
<![%SVG.glyph.attlist;[
<!ATTLIST %SVG.glyph.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    unicode CDATA #IMPLIED
    glyph-name CDATA #IMPLIED
    d %PathData.datatype; #IMPLIED
```

```
    orientation CDATA #IMPLIED
    arabic-form CDATA #IMPLIED
    lang %LanguageCodes.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #IMPLIED
    vert-origin-x %Number.datatype; #IMPLIED
    vert-origin-y %Number.datatype; #IMPLIED
    vert-adv-y %Number.datatype; #IMPLIED
>
<!-- end of SVG.glyph.attlist -->]]>

<!-- missing-glyph: Missing Glyph Element ............... -->

<!ENTITY % SVG.missing-glyph.extra.content "" >

<!ENTITY % SVG.missing-glyph.element "INCLUDE" >
<![%SVG.missing-glyph.element;[
<!ENTITY % SVG.missing-glyph.content
    "( %SVG.Description.class; | %SVG.Animation.class; %SVG.Structure.class;
       %SVG.Conditional.class; %SVG.Image.class; %SVG.Style.class;
       %SVG.Shape.class; %SVG.Text.class; %SVG.Marker.class;
       %SVG.ColorProfile.class; %SVG.Gradient.class; %SVG.Pattern.class;
       %SVG.Clip.class; %SVG.Mask.class; %SVG.Filter.class; %SVG.Cursor.class;
       %SVG.Hyperlink.class; %SVG.View.class; %SVG.Script.class;
       %SVG.Font.class; %SVG.Extensibility.class; %SVG.missing-glyph.extra.content; )*"
>
<!ELEMENT %SVG.missing-glyph.qname; %SVG.missing-glyph.content; >
<!-- end of SVG.missing-glyph.element -->]]>

<!ENTITY % SVG.missing-glyph.attlist "INCLUDE" >
<![%SVG.missing-glyph.attlist;[
<!ATTLIST %SVG.missing-glyph.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    d %PathData.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #IMPLIED
    vert-origin-x %Number.datatype; #IMPLIED
    vert-origin-y %Number.datatype; #IMPLIED
    vert-adv-y %Number.datatype; #IMPLIED
>
<!-- end of SVG.missing-glyph.attlist -->]]>

<!-- hkern: Horizontal Kerning Element .................. -->

<!ENTITY % SVG.hkern.element "INCLUDE" >
<![%SVG.hkern.element;[
<!ENTITY % SVG.hkern.content "EMPTY" >
<!ELEMENT %SVG.hkern.qname; %SVG.hkern.content; >
<!-- end of SVG.hkern.element -->]]>

<!ENTITY % SVG.hkern.attlist "INCLUDE" >
<![%SVG.hkern.attlist;[
<!ATTLIST %SVG.hkern.qname;
    %SVG.Core.attrib;
    u1 CDATA #IMPLIED
    g1 CDATA #IMPLIED
    u2 CDATA #IMPLIED
    g2 CDATA #IMPLIED
    k %Number.datatype; #REQUIRED
>
<!-- end of SVG.hkern.attlist -->]]>

<!-- vkern: Vertical Kerning Element .................... -->

<!ENTITY % SVG.vkern.element "INCLUDE" >
<![%SVG.vkern.element;[
<!ENTITY % SVG.vkern.content "EMPTY" >
<!ELEMENT %SVG.vkern.qname; %SVG.vkern.content; >
<!-- end of SVG.vkern.element -->]]>

<!ENTITY % SVG.vkern.attlist "INCLUDE" >
<![%SVG.vkern.attlist;[
<!ATTLIST %SVG.vkern.qname;
    %SVG.Core.attrib;
    u1 CDATA #IMPLIED
    g1 CDATA #IMPLIED
    u2 CDATA #IMPLIED
```

```
    g2 CDATA #IMPLIED
    k %Number.datatype; #REQUIRED
>
<!-- end of SVG.vkern.attlist -->]]>

<!-- font-face-src: Font Face Source Element ........... -->

<!ENTITY % SVG.font-face-src.extra.content "" >

<!ENTITY % SVG.font-face-src.element "INCLUDE" >
<![%SVG.font-face-src.element;[
<!ENTITY % SVG.font-face-src.content
    "( %SVG.font-face-uri.qname; | %SVG.font-face-name.qname;
       %SVG.font-face-src.extra.content; )+"
>
<!ELEMENT %SVG.font-face-src.qname; %SVG.font-face-src.content; >
<!-- end of SVG.font-face-src.element -->]]>

<!ENTITY % SVG.font-face-src.attlist "INCLUDE" >
<![%SVG.font-face-src.attlist;[
<!ATTLIST %SVG.font-face-src.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.font-face-src.attlist -->]]>

<!-- font-face-uri: Font Face URI Element .............. -->

<!ENTITY % SVG.font-face-uri.extra.content "" >

<!ENTITY % SVG.font-face-uri.element "INCLUDE" >
<![%SVG.font-face-uri.element;[
<!ENTITY % SVG.font-face-uri.content
    "( %SVG.font-face-format.qname; %SVG.font-face-uri.extra.content; )*"
>
<!ELEMENT %SVG.font-face-uri.qname; %SVG.font-face-uri.content; >
<!-- end of SVG.font-face-uri.element -->]]>

<!ENTITY % SVG.font-face-uri.attlist "INCLUDE" >
<![%SVG.font-face-uri.attlist;[
<!ATTLIST %SVG.font-face-uri.qname;
    %SVG.Core.attrib;
    %SVG.XLinkRequired.attrib;
>
<!-- end of SVG.font-face-uri.attlist -->]]>

<!-- font-face-format: Font Face Format Element ........ -->

<!ENTITY % SVG.font-face-format.element "INCLUDE" >
<![%SVG.font-face-format.element;[
<!ENTITY % SVG.font-face-format.content "EMPTY" >
<!ELEMENT %SVG.font-face-format.qname; %SVG.font-face-format.content; >
<!-- end of SVG.font-face-format.element -->]]>

<!ENTITY % SVG.font-face-format.attlist "INCLUDE" >
<![%SVG.font-face-format.attlist;[
<!ATTLIST %SVG.font-face-format.qname;
    %SVG.Core.attrib;
    string CDATA #IMPLIED
>
<!-- end of SVG.font-face-format.attlist -->]]>

<!-- font-face-name: Font Face Name Element ............ -->

<!ENTITY % SVG.font-face-name.element "INCLUDE" >
<![%SVG.font-face-name.element;[
<!ENTITY % SVG.font-face-name.content "EMPTY" >
<!ELEMENT %SVG.font-face-name.qname; %SVG.font-face-name.content; >
<!-- end of SVG.font-face-name.element -->]]>

<!ENTITY % SVG.font-face-name.attlist "INCLUDE" >
<![%SVG.font-face-name.attlist;[
<!ATTLIST %SVG.font-face-name.qname;
    %SVG.Core.attrib;
    name CDATA #IMPLIED
>
<!-- end of SVG.font-face-name.attlist -->]]>
```

```
<!-- end of svg-font.mod -->
```

A.3.40 Basic Font Module

The Basic Font Module defines the Font.class element collection.

Collection name	Elements in collection
Font.class	font, font-face

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Basic Font Module ............................................ -->
<!-- file: svg-basic-font.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-basic-font.mod,v 1.3 2009/06/11 17:20:55 cmccorma Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Basic Font//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-basic-font.mod"

     ..................................................................... -->

<!-- Basic Font

        font, font-face, glyph, missing-glyph, hkern, font-face-src,
        font-face-name

     This module declares markup to provide support for template.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.font.qname "font" >
<!ENTITY % SVG.font-face.qname "font-face" >
<!ENTITY % SVG.glyph.qname "glyph" >
<!ENTITY % SVG.missing-glyph.qname "missing-glyph" >
<!ENTITY % SVG.hkern.qname "hkern" >
<!ENTITY % SVG.font-face-src.qname "font-face-src" >
<!ENTITY % SVG.font-face-name.qname "font-face-name" >

<!-- Attribute Collections (Default) ................... -->

<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Container.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.XLinkRequired.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Font.class ................................... -->
```

```
<!ENTITY % SVG.Font.extra.class "" >

<!ENTITY % SVG.Font.class
    "| %SVG.font.qname; | %SVG.font-face.qname; %SVG.Font.extra.class;"
>

<!-- font: Font Element ............................... -->

<!ENTITY % SVG.font.extra.content "" >

<!ENTITY % SVG.font.element "INCLUDE" >
<![%SVG.font.element;[
<!ENTITY % SVG.font.content
    "( %SVG.Description.class; | %SVG.font-face.qname;
     | %SVG.missing-glyph.qname; | %SVG.glyph.qname; | %SVG.hkern.qname;
       %SVG.font.extra.content; )*"
>
<!ELEMENT %SVG.font.qname; %SVG.font.content; >
<!-- end of SVG.font.element -->]]>

<!ENTITY % SVG.font.attlist "INCLUDE" >
<![%SVG.font.attlist;[
<!ATTLIST %SVG.font.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.External.attrib;
    horiz-origin-x %Number.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #REQUIRED
>
<!-- end of SVG.font.attlist -->]]>

<!-- font-face: Font Face Element ..................... -->

<!ENTITY % SVG.font-face.extra.content "" >

<!ENTITY % SVG.font-face.element "INCLUDE" >
<![%SVG.font-face.element;[
<!ENTITY % SVG.font-face.content
    "( ( %SVG.font-face-src.qname;,
         ( %SVG.Description.class; %SVG.font-face.extra.content; )* )
       | ( ( %SVG.Description.class; %SVG.font-face.extra.content; )+,
           %SVG.font-face-src.qname;,
           ( ( %SVG.Description.class; %SVG.font-face.extra.content; )* )? ) )"
>
<!ELEMENT %SVG.font-face.qname; %SVG.font-face.content; >
<!-- end of SVG.font-face.element -->]]>

<!ENTITY % SVG.font-face.attlist "INCLUDE" >
<![%SVG.font-face.attlist;[
<!ATTLIST %SVG.font-face.qname;
    %SVG.Core.attrib;
    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.datatype; #IMPLIED
    panose-1 CDATA #IMPLIED
    stemv %Number.datatype; #IMPLIED
    stemh %Number.datatype; #IMPLIED
    slope %Number.datatype; #IMPLIED
    cap-height %Number.datatype; #IMPLIED
    x-height %Number.datatype; #IMPLIED
    accent-height %Number.datatype; #IMPLIED
    ascent %Number.datatype; #IMPLIED
    descent %Number.datatype; #IMPLIED
    widths CDATA #IMPLIED
    bbox CDATA #IMPLIED
    ideographic %Number.datatype; #IMPLIED
    alphabetic %Number.datatype; #IMPLIED
    mathematical %Number.datatype; #IMPLIED
    hanging %Number.datatype; #IMPLIED
    underline-position %Number.datatype; #IMPLIED
    underline-thickness %Number.datatype; #IMPLIED
    strikethrough-position %Number.datatype; #IMPLIED
```

```
    strikethrough-thickness %Number.datatype; #IMPLIED
    overline-position %Number.datatype; #IMPLIED
    overline-thickness %Number.datatype; #IMPLIED
>
<!-- end of SVG.font-face.attlist -->]]>

<!-- glyph: Glyph Element ............................. -->

<!ENTITY % SVG.glyph.extra.content "" >

<!ENTITY % SVG.glyph.element "INCLUDE" >
<![%SVG.glyph.element;[
<!ENTITY % SVG.glyph.content
    "( %SVG.Description.class; %SVG.glyph.extra.content; )*"
>
<!ELEMENT %SVG.glyph.qname; %SVG.glyph.content; >
<!-- end of SVG.glyph.element -->]]>

<!ENTITY % SVG.glyph.attlist "INCLUDE" >
<![%SVG.glyph.attlist;[
<!ATTLIST %SVG.glyph.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    unicode CDATA #IMPLIED
    glyph-name CDATA #IMPLIED
    d %PathData.datatype; #IMPLIED
    arabic-form CDATA #IMPLIED
    lang %LanguageCodes.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #IMPLIED
>
<!-- end of SVG.glyph.attlist -->]]>

<!-- missing-glyph: Missing Glyph Element .............. -->

<!ENTITY % SVG.missing-glyph.extra.content "" >

<!ENTITY % SVG.missing-glyph.element "INCLUDE" >
<![%SVG.missing-glyph.element;[
<!ENTITY % SVG.missing-glyph.content
    "( %SVG.Description.class; %SVG.missing-glyph.extra.content; )*"
>
<!ELEMENT %SVG.missing-glyph.qname; %SVG.missing-glyph.content; >
<!-- end of SVG.missing-glyph.element -->]]>

<!ENTITY % SVG.missing-glyph.attlist "INCLUDE" >
<![%SVG.missing-glyph.attlist;[
<!ATTLIST %SVG.missing-glyph.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    d %PathData.datatype; #IMPLIED
    horiz-adv-x %Number.datatype; #IMPLIED
>
<!-- end of SVG.missing-glyph.attlist -->]]>

<!-- hkern: Horizontal Kerning Element ................. -->

<!ENTITY % SVG.hkern.element "INCLUDE" >
<![%SVG.hkern.element;[
<!ENTITY % SVG.hkern.content "EMPTY" >
<!ELEMENT %SVG.hkern.qname; %SVG.hkern.content; >
<!-- end of SVG.hkern.element -->]]>

<!ENTITY % SVG.hkern.attlist "INCLUDE" >
<![%SVG.hkern.attlist;[
<!ATTLIST %SVG.hkern.qname;
    %SVG.Core.attrib;
    u1 CDATA #IMPLIED
    g1 CDATA #IMPLIED
    u2 CDATA #IMPLIED
    g2 CDATA #IMPLIED
    k %Number.datatype; #REQUIRED
>
<!-- end of SVG.hkern.attlist -->]]>

<!-- font-face-src: Font Face Source Element ........... -->

<!ENTITY % SVG.font-face-src.extra.content "" >
```

```
<!ENTITY % SVG.font-face-src.element "INCLUDE" >
<![%SVG.font-face-src.element;[
<!ENTITY % SVG.font-face-src.content
    "( %SVG.font-face-name.qname; %SVG.font-face-src.extra.content; )+"
>
<!ELEMENT %SVG.font-face-src.qname; %SVG.font-face-src.content; >
<!-- end of SVG.font-face-src.element -->]]>

<!ENTITY % SVG.font-face-src.attlist "INCLUDE" >
<![%SVG.font-face-src.attlist;[
<!ATTLIST %SVG.font-face-src.qname;
    %SVG.Core.attrib;
>
<!-- end of SVG.font-face-src.attlist -->]]>

<!-- font-face-name: Font Face Name Element ............ -->

<!ENTITY % SVG.font-face-name.element "INCLUDE" >
<![%SVG.font-face-name.element;[
<!ENTITY % SVG.font-face-name.content "EMPTY" >
<!ELEMENT %SVG.font-face-name.qname; %SVG.font-face-name.content; >
<!-- end of SVG.font-face-name.element -->]]>

<!ENTITY % SVG.font-face-name.attlist "INCLUDE" >
<![%SVG.font-face-name.attlist;[
<!ATTLIST %SVG.font-face-name.qname;
    %SVG.Core.attrib;
    name CDATA #IMPLIED
>
<!-- end of SVG.font-face-name.attlist -->]]>

<!-- end of svg-basic-font.mod -->
```

A.3.41 Extensibility Module

The Extensibility Module defines the Extensibility.class element collection.

Collection name	Elements in collection
Extensibility.class	foreignObject

```
<!-- ..................................................................... -->
<!-- SVG 1.1 Extensibility Module ........................................ -->
<!-- file: svg-extensibility.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg-extensibility.mod,v 1.1 2008/12/11 06:05:55 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ELEMENTS SVG 1.1 Extensibility//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg-extensibility.mod"

     ..................................................................... -->

<!-- Extensibility

        foreignObject

     This module declares markup to provide support for extensibility.
-->

<!-- Qualified Names (Default) ........................ -->

<!ENTITY % SVG.foreignObject.qname "foreignObject" >

<!-- Attribute Collections (Default) ................... -->
```

```
<!ENTITY % SVG.Core.attrib "" >
<!ENTITY % SVG.Conditional.attrib "" >
<!ENTITY % SVG.Style.attrib "" >
<!ENTITY % SVG.Viewport.attrib "" >
<!ENTITY % SVG.Text.attrib "" >
<!ENTITY % SVG.TextContent.attrib "" >
<!ENTITY % SVG.Font.attrib "" >
<!ENTITY % SVG.Paint.attrib "" >
<!ENTITY % SVG.Color.attrib "" >
<!ENTITY % SVG.Opacity.attrib "" >
<!ENTITY % SVG.Graphics.attrib "" >
<!ENTITY % SVG.Marker.attrib "" >
<!ENTITY % SVG.Gradient.attrib "" >
<!ENTITY % SVG.Clip.attrib "" >
<!ENTITY % SVG.Mask.attrib "" >
<!ENTITY % SVG.Filter.attrib "" >
<!ENTITY % SVG.FilterColor.attrib "" >
<!ENTITY % SVG.GraphicalEvents.attrib "" >
<!ENTITY % SVG.Cursor.attrib "" >
<!ENTITY % SVG.External.attrib "" >

<!-- SVG.Extensibility.class ........................... -->

<!ENTITY % SVG.Extensibility.extra.class "" >

<!ENTITY % SVG.Extensibility.class
    "| %SVG.foreignObject.qname; %SVG.Extensibility.extra.class;"
>

<!-- SVG.Presentation.attrib ........................... -->

<!ENTITY % SVG.Presentation.extra.attrib "" >

<!ENTITY % SVG.Presentation.attrib
    "%SVG.Container.attrib;
     %SVG.Viewport.attrib;
     %SVG.Text.attrib;
     %SVG.TextContent.attrib;
     %SVG.Font.attrib;
     %SVG.Paint.attrib;
     %SVG.Color.attrib;
     %SVG.Opacity.attrib;
     %SVG.Graphics.attrib;
     %SVG.Marker.attrib;
     %SVG.ColorProfile.attrib;
     %SVG.Gradient.attrib;
     %SVG.Clip.attrib;
     %SVG.Mask.attrib;
     %SVG.Filter.attrib;
     %SVG.FilterColor.attrib;
     %SVG.Cursor.attrib;
     flood-color %SVGColor.datatype; #IMPLIED
     flood-opacity %OpacityValue.datatype; #IMPLIED
     lighting-color %SVGColor.datatype; #IMPLIED
     %SVG.Presentation.extra.attrib;"
>

<!-- foreignObject: Foreign Object Element ............. -->

<!ENTITY % SVG.foreignObject.extra.content "" >

<!ENTITY % SVG.foreignObject.element "INCLUDE" >
<![%SVG.foreignObject.element;[
<!ENTITY % SVG.foreignObject.content
    "( #PCDATA %SVG.foreignObject.extra.content; )*"
>
<!ELEMENT %SVG.foreignObject.qname; %SVG.foreignObject.content; >
<!-- end of SVG.foreignObject.element -->]]>

<!ENTITY % SVG.foreignObject.attlist "INCLUDE" >
<![%SVG.foreignObject.attlist;[
<!ATTLIST %SVG.foreignObject.qname;
    %SVG.Core.attrib;
    %SVG.Conditional.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.GraphicalEvents.attrib;
```

```
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #REQUIRED
    height %Length.datatype; #REQUIRED
    transform %TransformList.datatype; #IMPLIED
>
<!-- end of SVG.foreignObject.attlist -->]]>

<!-- end of svg-extensibility.mod -->
```

A.4 SVG 1.1 Document Type Definition

A.4.1 SVG 1.1 DTD Driver

This section contains the DTD driver for the SVG 1.1 document type implementation as an XML DTD. It relies upon SVG 1.1 module implementations defined in Section A.3.

```
<!-- ..................................................................... -->
<!-- SVG 1.1 DTD .......................................................... -->
<!-- file: svg11.dtd
-->

<!-- SVG 1.1 DTD

     This is SVG, a language for describing two-dimensional graphics in XML.

     The Scalable Vector Graphics (SVG)
     Copyright 2001, 2002 World Wide Web Consortium
        (Massachusetts Institute of Technology, Institut National de
         Recherche en Informatique et en Automatique, Keio University).
         All Rights Reserved.

     Permission to use, copy, modify and distribute the SVG DTD and its
     accompanying documentation for any purpose and without fee is hereby
     granted in perpetuity, provided that the above copyright notice and
     this paragraph appear in all copies.  The copyright holders make no
     representation about the suitability of the DTD for any purpose.

     It is provided "as is" without expressed or implied warranty.

        Author:   Jun Fujisawa <fujisawa.jun@canon.co.jp>
        Revision: $Id: svg11.dtd,v 1.1 2008/12/11 06:05:57 agrasso Exp $

-->
<!-- This is the driver file for version 1.1 of the SVG DTD.

     This DTD is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//DTD SVG 1.1//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"
-->
<!ENTITY % SVG.version "-//W3C//DTD SVG 1.1//EN" >

<!-- Use this URI to identify the default namespace:

        "http://www.w3.org/2000/svg"

     See the Qualified Names module for information
     on the use of namespace prefixes in the DTD.
-->
<!ENTITY % NS.prefixed "IGNORE" >
<!ENTITY % SVG.prefix "" >

<!-- reserved for future use with document profiles -->
<!ENTITY % SVG.profile "" >

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!-- Pre-Framework Redeclaration Placeholder ..................... -->
```

```
<!ENTITY % svg-prefw-redecl.module "IGNORE" >
<![%svg-prefw-redecl.module;[
%svg-prefw-redecl.mod;]]>

<!-- Document Model Module ..................................... -->
<!ENTITY % svg-model.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Document Model//EN"
           "svg11-model.mod" >

<!-- Attribute Collection Module ............................... -->
<!ENTITY % svg-attribs.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Attribute Collection//EN"
           "svg11-attribs.mod" >

<!-- Modular Framework Module .................................. -->
<!ENTITY % svg-framework.module "INCLUDE" >
<![%svg-framework.module;[
<!ENTITY % svg-framework.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Modular Framework//EN"
           "svg-framework.mod" >
%svg-framework.mod;]]>

<!-- Post-Framework Redeclaration Placeholder ................... -->
<!ENTITY % svg-postfw-redecl.module "IGNORE" >
<![%svg-postfw-redecl.module;[
%svg-postfw-redecl.mod;]]>

<!-- :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!-- Core Attribute Module ..................................... -->
<!ENTITY % svg-core-attrib.module "INCLUDE" >
<![%svg-core-attrib.module;[
<!ENTITY % svg-core-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Core Attribute//EN"
           "svg-core-attrib.mod" >
%svg-core-attrib.mod;]]>

<!-- Container Attribute Module ................................ -->
<!ENTITY % svg-container-attrib.module "INCLUDE" >
<![%svg-container-attrib.module;[
<!ENTITY % svg-container-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Container Attribute//EN"
           "svg-container-attrib.mod" >
%svg-container-attrib.mod;]]>

<!-- Viewport Attribute Module ................................. -->
<!ENTITY % svg-viewport-attrib.module "INCLUDE" >
<![%svg-viewport-attrib.module;[
<!ENTITY % svg-viewport-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Viewport Attribute//EN"
           "svg-viewport-attrib.mod" >
%svg-viewport-attrib.mod;]]>

<!-- Paint Attribute Module .................................... -->
<!ENTITY % svg-paint-attrib.module "INCLUDE" >
<![%svg-paint-attrib.module;[
<!ENTITY % svg-paint-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Paint Attribute//EN"
           "svg-paint-attrib.mod" >
%svg-paint-attrib.mod;]]>

<!-- Paint Opacity Attribute Module ............................ -->
<!ENTITY % svg-opacity-attrib.module "INCLUDE" >
<![%svg-opacity-attrib.module;[
<!ENTITY % svg-opacity-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Paint Opacity Attribute//EN"
           "svg-opacity-attrib.mod" >
%svg-opacity-attrib.mod;]]>

<!-- Graphics Attribute Module ................................. -->
<!ENTITY % svg-graphics-attrib.module "INCLUDE" >
<![%svg-graphics-attrib.module;[
<!ENTITY % svg-graphics-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Graphics Attribute//EN"
           "svg-graphics-attrib.mod" >
%svg-graphics-attrib.mod;]]>
```

```
<!-- Document Events Attribute Module ........................... -->
<!ENTITY % svg-docevents-attrib.module "INCLUDE" >
<![%svg-docevents-attrib.module;[
<!ENTITY % svg-docevents-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Document Events Attribute//EN"
           "svg-docevents-attrib.mod" >
%svg-docevents-attrib.mod;]]>

<!-- Graphical Element Events Attribute Module .................. -->
<!ENTITY % svg-graphevents-attrib.module "INCLUDE" >
<![%svg-graphevents-attrib.module;[
<!ENTITY % svg-graphevents-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Graphical Element Events Attribute//EN"
           "svg-graphevents-attrib.mod" >
%svg-graphevents-attrib.mod;]]>

<!-- Animation Events Attribute Module .......................... -->
<!ENTITY % svg-animevents-attrib.module "INCLUDE" >
<![%svg-animevents-attrib.module;[
<!ENTITY % svg-animevents-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Animation Events Attribute//EN"
           "svg-animevents-attrib.mod" >
%svg-animevents-attrib.mod;]]>

<!-- XLink Attribute Module ..................................... -->
<!ENTITY % svg-xlink-attrib.module "INCLUDE" >
<![%svg-xlink-attrib.module;[
<!ENTITY % svg-xlink-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 XLink Attribute//EN"
           "svg-xlink-attrib.mod" >
%svg-xlink-attrib.mod;]]>

<!-- External Resources Attribute Module ........................ -->
<!ENTITY % svg-extresources-attrib.module "INCLUDE" >
<![%svg-extresources-attrib.module;[
<!ENTITY % svg-extresources-attrib.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 External Resources Attribute//EN"
           "svg-extresources-attrib.mod" >
%svg-extresources-attrib.mod;]]>

<!-- :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!-- Structure Module ........................................... -->
<!ENTITY % svg-structure.module "INCLUDE" >
<![%svg-structure.module;[
<!ENTITY % svg-structure.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Structure//EN"
           "svg-structure.mod" >
%svg-structure.mod;]]>

<!-- Conditional Processing Module .............................. -->
<!ENTITY % svg-conditional.module "INCLUDE" >
<![%svg-conditional.module;[
<!ENTITY % svg-conditional.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Conditional Processing//EN"
           "svg-conditional.mod" >
%svg-conditional.mod;]]>

<!-- Image Module ............................................... -->
<!ENTITY % svg-image.module "INCLUDE" >
<![%svg-image.module;[
<!ENTITY % svg-image.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Image//EN"
           "svg-image.mod" >
%svg-image.mod;]]>

<!-- Style Module ............................................... -->
<!ENTITY % svg-style.module "INCLUDE" >
<![%svg-style.module;[
<!ENTITY % svg-style.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Style//EN"
           "svg-style.mod" >
%svg-style.mod;]]>

<!-- Shape Module ............................................... -->
<!ENTITY % svg-shape.module "INCLUDE" >
<![%svg-shape.module;[
```

```
<!ENTITY % svg-shape.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Shape//EN"
           "svg-shape.mod" >
%svg-shape.mod;]]>

<!-- Text Module ..................................................... -->
<!ENTITY % svg-text.module "INCLUDE" >
<![%svg-text.module;[
<!ENTITY % svg-text.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Text//EN"
           "svg-text.mod" >
%svg-text.mod;]]>

<!-- Marker Module ................................................... -->
<!ENTITY % svg-marker.module "INCLUDE" >
<![%svg-marker.module;[
<!ENTITY % svg-marker.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Marker//EN"
           "svg-marker.mod" >
%svg-marker.mod;]]>

<!-- Color Profile Module ............................................ -->
<!ENTITY % svg-profile.module "INCLUDE" >
<![%svg-profile.module;[
<!ENTITY % svg-profile.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Color Profile//EN"
           "svg-profile.mod" >
%svg-profile.mod;]]>

<!-- Gradient Module ................................................. -->
<!ENTITY % svg-gradient.module "INCLUDE" >
<![%svg-gradient.module;[
<!ENTITY % svg-gradient.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Gradient//EN"
           "svg-gradient.mod" >
%svg-gradient.mod;]]>

<!-- Pattern Module .................................................. -->
<!ENTITY % svg-pattern.module "INCLUDE" >
<![%svg-pattern.module;[
<!ENTITY % svg-pattern.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Pattern//EN"
           "svg-pattern.mod" >
%svg-pattern.mod;]]>

<!-- Clip Module ..................................................... -->
<!ENTITY % svg-clip.module "INCLUDE" >
<![%svg-clip.module;[
<!ENTITY % svg-clip.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Clip//EN"
           "svg-clip.mod" >
%svg-clip.mod;]]>

<!-- Mask Module ..................................................... -->
<!ENTITY % svg-mask.module "INCLUDE" >
<![%svg-mask.module;[
<!ENTITY % svg-mask.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Mask//EN"
           "svg-mask.mod" >
%svg-mask.mod;]]>

<!-- Filter Module ................................................... -->
<!ENTITY % svg-filter.module "INCLUDE" >
<![%svg-filter.module;[
<!ENTITY % svg-filter.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Filter//EN"
           "svg-filter.mod" >
%svg-filter.mod;]]>

<!-- Cursor Module ................................................... -->
<!ENTITY % svg-cursor.module "INCLUDE" >
<![%svg-cursor.module;[
<!ENTITY % svg-cursor.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Cursor//EN"
           "svg-cursor.mod" >
%svg-cursor.mod;]]>
```

```
<!-- Hyperlinking Module ..................................... -->
<!ENTITY % svg-hyperlink.module "INCLUDE" >
<![%svg-hyperlink.module;[
<!ENTITY % svg-hyperlink.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Hyperlinking//EN"
           "svg-hyperlink.mod" >
%svg-hyperlink.mod;]]>

<!-- View Module ............................................. -->
<!ENTITY % svg-view.module "INCLUDE" >
<![%svg-view.module;[
<!ENTITY % svg-view.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 View//EN"
           "svg-view.mod" >
%svg-view.mod;]]>

<!-- Scripting Module ........................................ -->
<!ENTITY % svg-script.module "INCLUDE" >
<![%svg-script.module;[
<!ENTITY % svg-script.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Scripting//EN"
           "svg-script.mod" >
%svg-script.mod;]]>

<!-- Animation Module ........................................ -->
<!ENTITY % svg-animation.module "INCLUDE" >
<![%svg-animation.module;[
<!ENTITY % svg-animation.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Animation//EN"
           "svg-animation.mod" >
%svg-animation.mod;]]>

<!-- Font Module ............................................. -->
<!ENTITY % svg-font.module "INCLUDE" >
<![%svg-font.module;[
<!ENTITY % svg-font.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Font//EN"
           "svg-font.mod" >
%svg-font.mod;]]>

<!-- Extensibility Module .................................... -->
<!ENTITY % svg-extensibility.module "INCLUDE" >
<![%svg-extensibility.module;[
<!ENTITY % svg-extensibility.mod
    PUBLIC "-//W3C//ELEMENTS SVG 1.1 Extensibility//EN"
           "svg-extensibility.mod" >
%svg-extensibility.mod;]]>

<!-- end of SVG 1.1 DTD ...................................................... -->
<!-- ......................................................................... -->
```

A.4.2 SVG 1.1 Document Model

A SVG Family Document Type (such as SVG 1.1) must define the content model that it uses. This is done through a separate content model module that is instantiated by the SVG Modular Framework. The content model module and the SVG 1.1 DTD Driver (above) work together to customize the module implementations to the document type's specific requirements. The content model module for SVG 1.1 is defined below:

```
<!-- ......................................................................... -->
<!-- SVG 1.1 Document Model Module ........................................ -->
<!-- file: svg11-model.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg11-model.mod,v 1.1 2008/12/11 06:05:57 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Document Model//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-model.mod"
```

```
     ..................................................................... -->

<!-- SVG 1.1 Document Model

     This module describes the groupings of elements that make up
     common content models for SVG elements.
-->

<!-- module: svg-structure.mod ....................... -->

<!ENTITY % SVG.Description.extra.class "" >
<!ENTITY % SVG.Description.class
    "%SVG.desc.qname; | %SVG.title.qname; | %SVG.metadata.qname;
     %SVG.Description.extra.class;"
>

<!ENTITY % SVG.Use.extra.class "" >
<!ENTITY % SVG.Use.class
    "| %SVG.use.qname; %SVG.Use.extra.class;"
>

<!ENTITY % SVG.Structure.extra.class "" >
<!ENTITY % SVG.Structure.class
    "| %SVG.svg.qname; | %SVG.g.qname; | %SVG.defs.qname; | %SVG.symbol.qname;
       %SVG.Use.class; %SVG.Structure.extra.class;"
>

<!-- module: svg-conditional.mod ..................... -->

<!ENTITY % SVG.Conditional.extra.class "" >
<!ENTITY % SVG.Conditional.class
    "| %SVG.switch.qname; %SVG.Conditional.extra.class;"
>

<!-- module: svg-image.mod ........................... -->

<!ENTITY % SVG.Image.extra.class "" >
<!ENTITY % SVG.Image.class
    "| %SVG.image.qname; %SVG.Image.extra.class;"
>

<!-- module: svg-style.mod ........................... -->

<!ENTITY % SVG.Style.extra.class "" >
<!ENTITY % SVG.Style.class
    "| %SVG.style.qname; %SVG.Style.extra.class;"
>

<!-- module: svg-shape.mod ........................... -->

<!ENTITY % SVG.Shape.extra.class "" >
<!ENTITY % SVG.Shape.class
    "| %SVG.path.qname; | %SVG.rect.qname; | %SVG.circle.qname;
     | %SVG.line.qname; | %SVG.ellipse.qname; | %SVG.polyline.qname;
     | %SVG.polygon.qname; %SVG.Shape.extra.class;"
>

<!-- module: svg-text.mod ............................ -->

<!ENTITY % SVG.Text.extra.class "" >
<!ENTITY % SVG.Text.class
    "| %SVG.text.qname; | %SVG.altGlyphDef.qname; %SVG.Text.extra.class;"
>

<!ENTITY % SVG.TextContent.extra.class "" >
<!ENTITY % SVG.TextContent.class
    "| %SVG.tspan.qname; | %SVG.tref.qname; | %SVG.textPath.qname;
     | %SVG.altGlyph.qname; %SVG.TextContent.extra.class;"
>

<!-- module: svg-marker.mod .......................... -->

<!ENTITY % SVG.Marker.extra.class "" >
<!ENTITY % SVG.Marker.class
    "| %SVG.marker.qname; %SVG.Marker.extra.class;"
>
```

```
<!-- module: svg-profile.mod ........................... -->

<!ENTITY % SVG.ColorProfile.extra.class "" >
<!ENTITY % SVG.ColorProfile.class
    "| %SVG.color-profile.qname; %SVG.ColorProfile.extra.class;"
>

<!-- module: svg-gradient.mod ........................... -->

<!ENTITY % SVG.Gradient.extra.class "" >
<!ENTITY % SVG.Gradient.class
    "| %SVG.linearGradient.qname; | %SVG.radialGradient.qname;
       %SVG.Gradient.extra.class;"
>

<!-- module: svg-pattern.mod ........................... -->

<!ENTITY % SVG.Pattern.extra.class "" >
<!ENTITY % SVG.Pattern.class
    "| %SVG.pattern.qname; %SVG.Pattern.extra.class;"
>

<!-- module: svg-clip.mod ............................. -->

<!ENTITY % SVG.Clip.extra.class "" >
<!ENTITY % SVG.Clip.class
    "| %SVG.clipPath.qname; %SVG.Clip.extra.class;"
>

<!-- module: svg-mask.mod ............................. -->

<!ENTITY % SVG.Mask.extra.class "" >
<!ENTITY % SVG.Mask.class
    "| %SVG.mask.qname; %SVG.Mask.extra.class;"
>

<!-- module: svg-filter.mod ............................ -->

<!ENTITY % SVG.Filter.extra.class "" >
<!ENTITY % SVG.Filter.class
    "| %SVG.filter.qname; %SVG.Filter.extra.class;"
>

<!ENTITY % SVG.FilterPrimitive.extra.class "" >
<!ENTITY % SVG.FilterPrimitive.class
    "| %SVG.feBlend.qname; | %SVG.feColorMatrix.qname;
     | %SVG.feComponentTransfer.qname; | %SVG.feComposite.qname;
     | %SVG.feConvolveMatrix.qname; | %SVG.feDiffuseLighting.qname;
     | %SVG.feDisplacementMap.qname; | %SVG.feFlood.qname;
     | %SVG.feGaussianBlur.qname; | %SVG.feImage.qname; | %SVG.feMerge.qname;
     | %SVG.feMorphology.qname; | %SVG.feOffset.qname;
     | %SVG.feSpecularLighting.qname; | %SVG.feTile.qname;
     | %SVG.feTurbulence.qname; %SVG.FilterPrimitive.extra.class;"
>

<!-- module: svg-cursor.mod ............................ -->

<!ENTITY % SVG.Cursor.extra.class "" >
<!ENTITY % SVG.Cursor.class
    "| %SVG.cursor.qname; %SVG.Cursor.extra.class;"
>

<!-- module: svg-hyperlink.mod ......................... -->

<!ENTITY % SVG.Hyperlink.extra.class "" >
<!ENTITY % SVG.Hyperlink.class
    "| %SVG.a.qname; %SVG.Hyperlink.extra.class;"
>

<!-- module: svg-view.mod .............................. -->

<!ENTITY % SVG.View.extra.class "" >
<!ENTITY % SVG.View.class
    "| %SVG.view.qname; %SVG.View.extra.class;"
>

<!-- module: svg-script.mod ............................ -->
```

```
<!ENTITY % SVG.Script.extra.class "" >
<!ENTITY % SVG.Script.class
    "| %SVG.script.qname; %SVG.Script.extra.class;"
>

<!-- module: svg-animation.mod ....................... -->

<!ENTITY % SVG.Animation.extra.class "" >
<!ENTITY % SVG.Animation.class
    "%SVG.animate.qname; | %SVG.set.qname; | %SVG.animateMotion.qname; |
     %SVG.animateColor.qname; | %SVG.animateTransform.qname;
     %SVG.Animation.extra.class;"
>

<!-- module: svg-font.mod ............................ -->

<!ENTITY % SVG.Font.extra.class "" >
<!ENTITY % SVG.Font.class
    "| %SVG.font.qname; | %SVG.font-face.qname; %SVG.Font.extra.class;"
>

<!-- module: svg-extensibility.mod ................... -->

<!ENTITY % SVG.Extensibility.extra.class "" >
<!ENTITY % SVG.Extensibility.class
    "| %SVG.foreignObject.qname; %SVG.Extensibility.extra.class;"
>

<!-- end of svg11-model.mod -->
```

A.4.3 SVG 1.1 Attribute Collection

This section contains the attribute collection for SVG 1.1. The attribute collection module and the SVG 1.1 DTD Driver work together to customize the module implementations to the document type's specific requirements.

```
<!-- ................................................................... -->
<!-- SVG 1.1 Attribute Collection Module ............................... -->
<!-- file: svg11-attribs.mod

     This is SVG, a language for describing two-dimensional graphics in XML.
     Copyright 2001, 2002 W3C (MIT, INRIA, Keio), All Rights Reserved.
     Revision: $Id: svg11-attribs.mod,v 1.1 2008/12/11 06:05:56 agrasso Exp $

     This DTD module is identified by the PUBLIC and SYSTEM identifiers:

        PUBLIC "-//W3C//ENTITIES SVG 1.1 Attribute Collection//EN"
        SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-attribs.mod"

     ................................................................... -->

<!-- SVG 1.1 Attribute Collection

     This module defines the set of common attributes that can be present
     on many SVG elements.
-->

<!-- module: svg-conditional.mod ..................... -->

<!ENTITY % ExtensionList.datatype "CDATA" >
<!ENTITY % FeatureList.datatype "CDATA" >

<!ENTITY % SVG.Conditional.extra.attrib "" >
<!ENTITY % SVG.Conditional.attrib
    "requiredFeatures %FeatureList.datatype; #IMPLIED
     requiredExtensions %ExtensionList.datatype; #IMPLIED
     systemLanguage %LanguageCodes.datatype; #IMPLIED
     %SVG.Conditional.extra.attrib;"
>

<!-- module: svg-style.mod ........................... -->

<!ENTITY % ClassList.datatype "CDATA" >
```

```
<!ENTITY % StyleSheet.datatype "CDATA" >

<!ENTITY % SVG.Style.extra.attrib "" >
<!ENTITY % SVG.Style.attrib
    "style %StyleSheet.datatype; #IMPLIED
     class %ClassList.datatype; #IMPLIED
     %SVG.Style.extra.attrib;"
>

<!-- module: svg-text.mod ............................. -->

<!ENTITY % BaselineShiftValue.datatype "CDATA" >
<!ENTITY % FontFamilyValue.datatype "CDATA" >
<!ENTITY % FontSizeValue.datatype "CDATA" >
<!ENTITY % FontSizeAdjustValue.datatype "CDATA" >
<!ENTITY % GlyphOrientationHorizontalValue.datatype "CDATA" >
<!ENTITY % GlyphOrientationVerticalValue.datatype "CDATA" >
<!ENTITY % KerningValue.datatype "CDATA" >
<!ENTITY % SpacingValue.datatype "CDATA" >
<!ENTITY % TextDecorationValue.datatype "CDATA" >

<!ENTITY % SVG.Text.extra.attrib "" >
<!ENTITY % SVG.Text.attrib
    "writing-mode ( lr-tb | rl-tb | tb-rl | lr | rl | tb | inherit ) #IMPLIED
     %SVG.Text.extra.attrib;"
>

<!ENTITY % SVG.TextContent.extra.attrib "" >
<!ENTITY % SVG.TextContent.attrib
    "alignment-baseline ( auto | baseline | before-edge | text-before-edge |
                          middle | central | after-edge | text-after-edge |
                          ideographic | alphabetic | hanging | mathematical |
                          inherit ) #IMPLIED
     baseline-shift %BaselineShiftValue.datatype; #IMPLIED
     direction ( ltr | rtl | inherit ) #IMPLIED
     dominant-baseline ( auto | use-script | no-change | reset-size |
                         ideographic | alphabetic | hanging | mathematical |
                         central | middle | text-after-edge | text-before-edge |
                         inherit ) #IMPLIED
     glyph-orientation-horizontal %GlyphOrientationHorizontalValue.datatype;
                                  #IMPLIED
     glyph-orientation-vertical %GlyphOrientationVerticalValue.datatype;
                                #IMPLIED
     kerning %KerningValue.datatype; #IMPLIED
     letter-spacing %SpacingValue.datatype; #IMPLIED
     text-anchor ( start | middle | end | inherit ) #IMPLIED
     text-decoration %TextDecorationValue.datatype; #IMPLIED
     unicode-bidi ( normal | embed | bidi-override | inherit ) #IMPLIED
     word-spacing %SpacingValue.datatype; #IMPLIED
     %SVG.TextContent.extra.attrib;"
>

<!ENTITY % SVG.Font.extra.attrib "" >
<!ENTITY % SVG.Font.attrib
    "font-family %FontFamilyValue.datatype; #IMPLIED
     font-size %FontSizeValue.datatype; #IMPLIED
     font-size-adjust %FontSizeAdjustValue.datatype; #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
     %SVG.Font.extra.attrib;"
>

<!-- module: svg-marker.mod ........................... -->

<!ENTITY % MarkerValue.datatype "CDATA" >

<!ENTITY % SVG.Marker.extra.attrib "" >
<!ENTITY % SVG.Marker.attrib
    "marker-start %MarkerValue.datatype; #IMPLIED
     marker-mid %MarkerValue.datatype; #IMPLIED
     marker-end %MarkerValue.datatype; #IMPLIED
```

```
     %SVG.Marker.extra.attrib;"
>

<!-- module: svg-profile.mod ........................... -->

<!ENTITY % SVG.ColorProfile.extra.attrib "" >
<!ENTITY % SVG.ColorProfile.attrib
    "color-profile CDATA #IMPLIED
     %SVG.ColorProfile.extra.attrib;"
>

<!-- module: svg-gradient.mod ......................... -->

<!ENTITY % NumberOrPercentage.datatype "CDATA" >

<!ENTITY % SVG.Gradient.extra.attrib "" >
<!ENTITY % SVG.Gradient.attrib
    "stop-color %SVGColor.datatype; #IMPLIED
     stop-opacity %OpacityValue.datatype; #IMPLIED
     %SVG.Gradient.extra.attrib;"
>

<!-- module: svg-clip.mod ............................. -->

<!ENTITY % ClipPathValue.datatype "CDATA" >

<!ENTITY % SVG.Clip.extra.attrib "" >
<!ENTITY % SVG.Clip.attrib
    "clip-path %ClipPathValue.datatype; #IMPLIED
     clip-rule %ClipFillRule.datatype; #IMPLIED
     %SVG.Clip.extra.attrib;"
>

<!-- module: svg-mask.mod ............................. -->

<!ENTITY % MaskValue.datatype "CDATA" >

<!ENTITY % SVG.Mask.extra.attrib "" >
<!ENTITY % SVG.Mask.attrib
    "mask %MaskValue.datatype; #IMPLIED
     %SVG.Mask.extra.attrib;"
>

<!-- module: svg-filter.mod ........................... -->

<!ENTITY % FilterValue.datatype "CDATA" >
<!ENTITY % NumberOptionalNumber.datatype "CDATA" >

<!ENTITY % SVG.Filter.extra.attrib "" >
<!ENTITY % SVG.Filter.attrib
    "filter %FilterValue.datatype; #IMPLIED
     %SVG.Filter.extra.attrib;"
>

<!ENTITY % SVG.FilterColor.extra.attrib "" >
<!ENTITY % SVG.FilterColor.attrib
    "color-interpolation-filters ( auto | sRGB | linearRGB | inherit )
                                 #IMPLIED
     %SVG.FilterColor.extra.attrib;"
>

<!-- module: svg-cursor.mod ........................... -->

<!ENTITY % CursorValue.datatype "CDATA" >

<!ENTITY % SVG.Cursor.extra.attrib "" >
<!ENTITY % SVG.Cursor.attrib
    "cursor %CursorValue.datatype; #IMPLIED
     %SVG.Cursor.extra.attrib;"
>

<!-- end of svg11-attribs.mod -->
```

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 Core 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 builds upon and is compatible with DOM Level 2. In particular:

- The SVG DOM requires complete support for DOM Level 2 Core [DOM2]
- Wherever appropriate, the SVG DOM is modeled after and maintains consistency with the Document Object Model HTML ([DOM1], chapter 2).
- The SVG DOM requires complete support for DOM Level 2 Views [DOM2VIEWS].
- The SVG DOM requires support for relevant aspects of DOM Level 2 Events [DOM2EVENTS]. (For the specific features from DOM 2 Events that are required, see see Relationship with DOM Level 2 Events.)
- For implementations that support CSS, the SVG DOM requires complete support for DOM Level 2 Style Sheets ([DOM2STYLE], chapter 1) and relevant aspects of DOM Level 2 CSS ([DOM2STYLE], chapter 2). (For the specific features from DOM 2 CSS that are required, see Relationship with DOM Level 2 CSS.)

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.

All SVG DOM objects that directly correspond to an attribute, e.g. the SVGAnimatedLength ry in an SVGRectElement, are live. This means that any changes made to the attribute are immediately reflected in the corresponding SVG DOM object.

B.1.1 SVG DOM object initialization

The SVG DOM allows attributes to be accessed even though they haven't been specified explicitly in the document markup. When this happens an appropriate object is created, initialized and returned. This newly constructed object does not affect rendering until it is modified for the first time. After the first modification the object becomes live, such that any modifications made to the corresponding attribute are immediately reflected in the object.

For example, if `rectElement.x.baseVal` is accessed and the 'x' attribute was not specified in the document, the returned SVG DOM object would represent the value **0 user units**.

For cases where an attribute has a default value the returned SVG DOM object that must reflect that value, and for all other cases the object is initialized as described below. If a particular SVG DOM interface is not listed below that means that the object initialization shall be done using the values for the objects that the interface contains, e.g **DOMString** in the case of SVGAnimatedString, or four **floats** in the case of SVGRect.

SVGTextContentElement.textLength

Initialized with the return-value of getComputedTextLength on the same element.

DOMString
Initialized as the empty string ("").
float
long
short
Initialized as **0**.
boolean
Initialized as **false**.
SVGLength
Initialized as **0 user units** (SVG_LENGTHTYPE_NUMBER).
SVGLengthList
SVGNumberList
SVGPointList
SVGStringList
SVGTransformList
Initialized as the empty list.
SVGAngle
Initialized as **0 in unspecified units** (SVG_ANGLETYPE_UNSPECIFIED).
SVGZoomAndPan
Initialized as **0** (SVG_ZOOMANDPAN_UNKNOWN).
SVGPreserveAspectRatio
Initialized as **'xMidYMid meet'**.

B.2 Elements in the SVG DOM

Every Element object that corresponds to an SVG element (that is, an element with namespace URI "http://www.w3.org/2000/svg" and a local name that is one of the elements defined in this specification) must also implement the DOM interface identified in element definition. For example, in The 'rect' element, the SVGRectElement interface is identified. This means that every Element object whose namespace URI is "http://www.w3.org/2000/svg" and whose local name is "rect" must also implement SVGRectElement.

B.3 Naming conventions

The SVG DOM follows similar naming conventions to the Document Object Model HTML ([DOM1], chapter 2).

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.4 Exception SVGException

This exception is raised when a specific SVG operation is impossible to perform.

```
exception SVGException {
  unsigned short code;
};

// SVGException code
const unsigned short SVG_WRONG_TYPE_ERR = 0;
const unsigned short SVG_INVALID_VALUE_ERR = 1;
const unsigned short SVG_MATRIX_NOT_INVERTABLE = 2;
```

Constants in group "SVGException code":

- **SVG_WRONG_TYPE_ERR** (unsigned short)

  Raised when an object of the wrong type is passed to an operation.

  Note that no operation is defined to raise an SVGException with this code in SVG 1.1 Second Edition. The constant remains defined here for consistency with SVG 1.1 First Edition.

- **SVG_INVALID_VALUE_ERR** (unsigned short)

  Raised when an invalid value is passed to an operation or assigned to an attribute.

- **SVG_MATRIX_NOT_INVERTABLE** (unsigned short)

  Raised when an attempt is made to invert a matrix that is not invertible.

  Note the unusual spelling of this constant, which is necessary for compatibility with existing content.

Exception members:

code (unsigned short)
: A code identifying the reason why the requested operation could not be performed. The value of this member will be one of the constants in the *SVGException code* group.

B.5 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 DOM Level 2 Core [DOM2] 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 appendix Feature Strings for a list of features in the SVG language), the **version** number for the **hasFeature** method call is "1.1". For features that correspond to other languages, refer to the relevant other specifications to determine the appropriate version number for the given feature.

B.6 Relationship with DOM Level 2 Events

The SVG DOM supports all of the interfaces defined in, and the following event types from, DOM Level 2 Events [DOM2EVENTS]:

- These User Interface events ([DOM2EVENTS], section 1.6.1):
  - DOMFocusIn
  - DOMFocusOut
  - DOMActivate
- These mouse events ([DOM2EVENTS], section 1.6.2):
  - click
  - mousedown
  - mouseup
  - mouseover
  - mousemove
  - mouseout

  clientX and *clientY* parameters for mouse events represent the mouse coordinates at which the event occurred relative to the DOM Implementation's client area. *relatedTarget* is used to identify a secondary EventTarget related to a UI event. Currently this attribute is used with the mouseover event to indicate the EventTarget which the pointing device exited and with the mouseout event to indicate the EventTarget which the pointing device entered.
- These mutation events ([DOM2EVENTS], section 1.6.4):
  - 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)

  Note that the SVGLoad event does not fire until the document is fully loaded and is therefore subject to the processing of any **'externalResourcesRequired'** attributes.
- The SVG DOM defines an additional custom event interface:
  - SVGZoom (definition can be found in the description of the SVGZoomEvent interface)

- The following event types are triggered due to state changes in animations. (The definitions for these events can be found in the description of the TimeEvent interface.)
  - beginEvent
  - endEvent
  - repeatEvent

While event listeners can be registered using an `addEventListener` call on any element in the DOM, the use of event attributes on elements where those attributes are disallowed will not result in their being invoked if the relevant event is dispatched to the element. For example, if the 'onclick' attribute were specified on a 'title' element, its contents would never be run in response to a **click** event:

```
<svg xmlns="http://www.w3.org/2000/svg">
  <title onclick="alert('Hello')">Invalid event attribute</title>
  <script>
    // Find the 'title' element.
    var title = document.getElementsByTagNameNS("http://www.w3.org/2000/svg", "title")[0];

    // Create and initialize a 'click' event.
    var event = document.createEvent("MouseEvent");
    event.initMouseEvent("click", true, false, this, 1, 0, 0, 0, 0, false,
                         false, false, false, 0, null);

    // Dispatch the event to the 'title' element.  Since onclick="" is not
    // allowed on 'title', the alert will not show.
    title.dispatchEvent(event);
  </script>
</svg>
```

See the Attribute Index for details on which elements a given event attribute is allowed to be specified on.

Implementors may view the setting of event attributes as the creation and registration of an EventListener on the EventTarget. Such event listeners are invoked only for the "bubbling" and "at target" phases, as if false were specified for the `useCapture` argument to `addEventListener`. This EventListener behaves in the same manner as any other which may be registered on the EventTarget.

If the attribute representing the event listener is changed, this may be viewed as the removal of the previously registered EventListener and the registration of a new one. Futhermore, no specification is made as to the order in which event attributes will receive the event with regards to the other EventListeners on the EventTarget.

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 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 becomes 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 write:

```
<rect onactivate="MyActivateHandler(evt)" .../>
```

which will pass the Event object **evt** into function `MyActivateHandler`.

B.7 Relationship with DOM Level 2 CSS

B.7.1 Introduction

The section describes the facilities from DOM Level 2 CSS ([DOM2STYLE], chapter 2) that are part of the SVG DOM.

B.7.2 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 DOM Level 2 CSS ([DOM2STYLE], chapter 2), 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, which must be supported on all implementations of the SVG DOM.

- Interface RGBColor
- Interface CSSValue

B.7.3 User agents that support styling with CSS

User agents that support Styling with CSS, the SVG DOM, and aural styling ([CSS2], chapter 19) must support all of the interfaces defined in DOM Level 2 CSS ([DOM2STYLE], chapter 2) which apply to aural properties.

For visual media ([CSS2], section 7.3.1), user agents must support all of the required interfaces defined in DOM Level 2 CSS. All of the interfaces that are optional for DOM Level 2 CSS are also optional for user agents implementing the SVG DOM.

B.7.4 Extended interfaces

Note: the getPresentationAttribute method and the interfaces that extend CSSValue are deprecated, and may be dropped from future versions of the SVG specification.

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.

DOM Level 2 CSS defines a set of extended interfaces ([DOM2STYLE], section 2.3) for use in conjunction with interface CSSValue. The table below specifies the type of CSSValue used to represent each SVG property that applies to visual media ([CSS2], section 7.3.1). 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 cssValueType 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 cssValueType of the CSSValue is CSS_VALUE_LIST), the derived interface is CSSValueList. For all other properties (the cssValueType 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	Mandatory? (Extended interfaces only)
'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.8 Read only nodes in the DOM

Some operations and attributes in the SVG DOM are defined to raise an exception when an attempt is made to modify a node in the DOM that is read only. Such read only nodes are not related to attributes declared in IDL with the `readonly` keyword. Rather, they are nodes that cannot be modified by virtue of being defined as readonly nodes by DOM Level 2 Core ([DOM2], Glossary appendix). Specifically, Entity and EntityReference nodes and their descendants are read only ([DOM2], section 1.3).

B.9 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 is normative.

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/2011/PR-SVG11-20110609/svg.idl

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.

```
module smil {

interface ElementTimeControl {
  void beginElement();
  void beginElementAt(in float offset);
  void endElement();
  void endElementAt(in float offset);
};

interface TimeEvent : Event {

  readonly attribute AbstractView view;
  readonly attribute long detail;

  void initTimeEvent(in DOMString typeArg, in AbstractView viewArg, in long detailArg);
};

};

module svg {

exception SVGException {
  unsigned short code;
};

// SVGException code
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 setraises(DOMException);
           attribute DOMString xmlbase setraises(DOMException);
  readonly attribute SVGSVGElement ownerSVGElement;
  readonly attribute SVGElement viewportElement;
};

interface SVGAnimatedBoolean {
           attribute boolean baseVal setraises(DOMException);
  readonly attribute boolean animVal;
};

interface SVGAnimatedString {
           attribute DOMString baseVal setraises(DOMException);
  readonly attribute DOMString animVal;
};

interface SVGStringList {

  readonly attribute unsigned long numberOfItems;
```

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

interface SVGAnimatedEnumeration {
           attribute unsigned short baseVal setraises(DOMException);
  readonly attribute unsigned short animVal;
};

interface SVGAnimatedInteger {
           attribute long baseVal setraises(DOMException);
  readonly attribute long animVal;
};

interface SVGNumber {
  attribute float value setraises(DOMException);
};

interface SVGAnimatedNumber {
           attribute float baseVal setraises(DOMException);
  readonly attribute float animVal;
};

interface SVGNumberList {

  readonly attribute unsigned long numberOfItems;

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

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 setraises(DOMException);
           attribute float valueInSpecifiedUnits setraises(DOMException);
           attribute DOMString valueAsString setraises(DOMException);

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

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);
  SVGLength getItem(in unsigned long index) raises(DOMException);
  SVGLength insertItemBefore(in SVGLength newItem, in unsigned long index) raises(DOMException);
  SVGLength replaceItem(in SVGLength newItem, in unsigned long index) raises(DOMException);
  SVGLength removeItem(in unsigned long index) raises(DOMException);
  SVGLength appendItem(in SVGLength newItem) raises(DOMException);
};

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 setraises(DOMException);
           attribute float valueInSpecifiedUnits setraises(DOMException);
           attribute DOMString valueAsString setraises(DOMException);

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

interface SVGAnimatedAngle {
  readonly attribute SVGAngle baseVal;
  readonly attribute SVGAngle animVal;
};

interface SVGColor : 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 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 setraises(DOMException);
  readonly attribute SVGNumberList colors;
};

interface SVGRect {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float width setraises(DOMException);
  attribute float height setraises(DOMException);
};

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 CSSStyleDeclaration style;

  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 setraises(DOMException);
  attribute DOMString xmlspace setraises(DOMException);
};

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 setraises(DOMException);
};

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 : CSSRule {
  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,
                        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,
                          DocumentEvent,
                          ViewCSS,
                          DocumentCSS {

  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
           attribute DOMString contentScriptType setraises(DOMException);
           attribute DOMString contentStyleType setraises(DOMException);
  readonly attribute SVGRect viewport;
  readonly attribute float pixelUnitToMillimeterX;
  readonly attribute float pixelUnitToMillimeterY;
  readonly attribute float screenPixelToMillimeterX;
  readonly attribute float screenPixelToMillimeterY;
  readonly attribute boolean useCurrentView;
  readonly attribute SVGViewSpec currentView;
           attribute float currentScale;
  readonly attribute SVGPoint currentTranslate;

  unsigned long suspendRedraw(in unsigned long maxWaitMilliseconds);
  void unsuspendRedraw(in unsigned long suspendHandleID);
  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 {
};

interface SVGDefsElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable {
};

interface SVGDescElement : SVGElement,
                           SVGLangSpace,
                           SVGStylable {
};

interface SVGTitleElement : SVGElement,
                            SVGLangSpace,
                            SVGStylable {
};

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

interface SVGUseElement : SVGElement,
                          SVGURIReference,
                          SVGTests,
                          SVGLangSpace,
                          SVGExternalResourcesRequired,
                          SVGStylable,
                          SVGTransformable {
  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 : 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 {
  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 {
};

interface GetSVGDocument {
  SVGDocument getSVGDocument();
};

interface SVGStyleElement : SVGElement,
                            SVGLangSpace {
  attribute DOMString type setraises(DOMException);
  attribute DOMString media setraises(DOMException);
  attribute DOMString title setraises(DOMException);
};

interface SVGPoint {

  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);

  SVGPoint matrixTransform(in SVGMatrix matrix);
};

interface SVGPointList {

  readonly attribute unsigned long numberOfItems;

  void clear() raises(DOMException);
  SVGPoint initialize(in SVGPoint newItem) raises(DOMException);
  SVGPoint getItem(in unsigned long index) raises(DOMException);
  SVGPoint insertItemBefore(in SVGPoint newItem, in unsigned long index) raises(DOMException);
  SVGPoint replaceItem(in SVGPoint newItem, in unsigned long index) raises(DOMException);
  SVGPoint removeItem(in unsigned long index) raises(DOMException);
  SVGPoint appendItem(in SVGPoint newItem) raises(DOMException);
};

interface SVGMatrix {

  attribute float a setraises(DOMException);
  attribute float b setraises(DOMException);
  attribute float c setraises(DOMException);
  attribute float d setraises(DOMException);
  attribute float e setraises(DOMException);
  attribute float f setraises(DOMException);

  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) raises(DOMException);
  void setTranslate(in float tx, in float ty) raises(DOMException);
  void setScale(in float sx, in float sy) raises(DOMException);
  void setRotate(in float angle, in float cx, in float cy) raises(DOMException);
  void setSkewX(in float angle) raises(DOMException);
  void setSkewY(in float angle) raises(DOMException);
};

interface SVGTransformList {

  readonly attribute unsigned long numberOfItems;

  void clear() raises(DOMException);
  SVGTransform initialize(in SVGTransform newItem) raises(DOMException);
  SVGTransform getItem(in unsigned long index) raises(DOMException);
  SVGTransform insertItemBefore(in SVGTransform newItem, in unsigned long index) raises(DOMException);
  SVGTransform replaceItem(in SVGTransform newItem, in unsigned long index) raises(DOMException);
  SVGTransform removeItem(in unsigned long index) raises(DOMException);
  SVGTransform appendItem(in SVGTransform newItem) raises(DOMException);
  SVGTransform createSVGTransformFromMatrix(in SVGMatrix matrix);
  SVGTransform consolidate() raises(DOMException);
};

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 setraises(DOMException);
  attribute unsigned short meetOrSlice setraises(DOMException);
};

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 setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegMovetoRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegLinetoAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegLinetoRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegCurvetoCubicAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};

interface SVGPathSegCurvetoCubicRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};

interface SVGPathSegCurvetoQuadraticAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
};

interface SVGPathSegCurvetoQuadraticRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x1 setraises(DOMException);
  attribute float y1 setraises(DOMException);
};
```

```
interface SVGPathSegArcAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float r1 setraises(DOMException);
  attribute float r2 setraises(DOMException);
  attribute float angle setraises(DOMException);
  attribute boolean largeArcFlag setraises(DOMException);
  attribute boolean sweepFlag setraises(DOMException);
};

interface SVGPathSegArcRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float r1 setraises(DOMException);
  attribute float r2 setraises(DOMException);
  attribute float angle setraises(DOMException);
  attribute boolean largeArcFlag setraises(DOMException);
  attribute boolean sweepFlag setraises(DOMException);
};

interface SVGPathSegLinetoHorizontalAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
};

interface SVGPathSegLinetoHorizontalRel : SVGPathSeg {
  attribute float x setraises(DOMException);
};

interface SVGPathSegLinetoVerticalAbs : SVGPathSeg {
  attribute float y setraises(DOMException);
};

interface SVGPathSegLinetoVerticalRel : SVGPathSeg {
  attribute float y setraises(DOMException);
};

interface SVGPathSegCurvetoCubicSmoothAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};

interface SVGPathSegCurvetoCubicSmoothRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float x2 setraises(DOMException);
  attribute float y2 setraises(DOMException);
};

interface SVGPathSegCurvetoQuadraticSmoothAbs : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegCurvetoQuadraticSmoothRel : SVGPathSeg {
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
};

interface SVGPathSegList {

  readonly attribute unsigned long numberOfItems;

  void clear() raises(DOMException);
  SVGPathSeg initialize(in SVGPathSeg newItem) raises(DOMException);
  SVGPathSeg getItem(in unsigned long index) raises(DOMException);
  SVGPathSeg insertItemBefore(in SVGPathSeg newItem, in unsigned long index) raises(DOMException);
  SVGPathSeg replaceItem(in SVGPathSeg newItem, in unsigned long index) raises(DOMException);
  SVGPathSeg removeItem(in unsigned long index) raises(DOMException);
  SVGPathSeg appendItem(in SVGPathSeg newItem) raises(DOMException);
};

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,
                           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 {
  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 {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength r;
};

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

```
                            SVGStylable,
                            SVGTransformable {
  readonly attribute SVGAnimatedLength cx;
  readonly attribute SVGAnimatedLength cy;
  readonly attribute SVGAnimatedLength rx;
  readonly attribute SVGAnimatedLength ry;
};

interface SVGLineElement : SVGElement,
                           SVGTests,
                           SVGLangSpace,
                           SVGExternalResourcesRequired,
                           SVGStylable,
                           SVGTransformable {
  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,
                               SVGAnimatedPoints {
};

interface SVGPolygonElement : SVGElement,
                              SVGTests,
                              SVGLangSpace,
                              SVGExternalResourcesRequired,
                              SVGStylable,
                              SVGTransformable,
                              SVGAnimatedPoints {
};

interface SVGTextContentElement : SVGElement,
                                  SVGTests,
                                  SVGLangSpace,
                                  SVGExternalResourcesRequired,
                                  SVGStylable {

  // 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 setraises(DOMException);
  attribute DOMString format setraises(DOMException);
};

interface SVGAltGlyphDefElement : SVGElement {
};

interface SVGAltGlyphItemElement : SVGElement {
};

interface SVGGlyphRefElement : SVGElement,
                               SVGURIReference,
                               SVGStylable {
  attribute DOMString glyphRef setraises(DOMException);
  attribute DOMString format setraises(DOMException);
  attribute float x setraises(DOMException);
  attribute float y setraises(DOMException);
  attribute float dx setraises(DOMException);
  attribute float dy setraises(DOMException);
};

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() raises(DOMException);
  void setOrientToAngle(in SVGAngle angle) raises(DOMException);
};

interface SVGColorProfileElement : SVGElement,
                                   SVGURIReference,
                                   SVGRenderingIntent {
  attribute DOMString local;
  attribute DOMString name;
  attribute unsigned short renderingIntent;
};

interface SVGColorProfileRule : SVGCSSRule,
                                SVGRenderingIntent {
  attribute DOMString src setraises(DOMException);
  attribute DOMString name setraises(DOMException);
  attribute unsigned short renderingIntent setraises(DOMException);
};

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) raises(DOMException);
};

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 SVGAnimatedString in1;
  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 SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
  readonly attribute SVGAnimatedBoolean preserveAlpha;
};

interface SVGFEDiffuseLightingElement : SVGElement,
                                        SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

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 {
};

interface SVGFEGaussianBlurElement : SVGElement,
                                     SVGFilterPrimitiveStandardAttributes {

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;

  void setStdDeviation(in float stdDeviationX, in float stdDeviationY) raises(DOMException);
};

interface SVGFEImageElement : SVGElement,
                              SVGURIReference,
                              SVGLangSpace,
                              SVGExternalResourcesRequired,
                              SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};

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 SVGAnimatedNumber radiusX;
  readonly attribute SVGAnimatedNumber 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;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

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 {
  readonly attribute SVGAnimatedString target;
};

interface SVGViewElement : SVGElement,
                           SVGExternalResourcesRequired,
                           SVGFitToViewBox,
                           SVGZoomAndPan {
  readonly attribute SVGStringList viewTarget;
};

interface SVGScriptElement : SVGElement,
                             SVGURIReference,
                             SVGExternalResourcesRequired {
  attribute DOMString type setraises(DOMException);
};

interface SVGZoomEvent : 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,
                                ElementTimeControl {

  readonly attribute SVGElement targetElement;

  float getStartTime() raises(DOMException);
  float getCurrentTime();
  float getSimpleDuration() raises(DOMException);
};

interface SVGAnimateElement : SVGAnimationElement,
                              SVGStylable {
};

interface SVGSetElement : SVGAnimationElement {
```

```
};

interface SVGAnimateMotionElement : SVGAnimationElement {
};

interface SVGMPathElement : SVGElement,
                            SVGURIReference,
                            SVGExternalResourcesRequired {
};

interface SVGAnimateColorElement : SVGAnimationElement,
                                   SVGStylable {
};

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 SVGMetadataElement : SVGElement {
};

interface SVGForeignObjectElement : SVGElement,
                                    SVGTests,
                                    SVGLangSpace,
                                    SVGExternalResourcesRequired,
                                    SVGStylable,
                                    SVGTransformable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
};

};
```

Appendix D: Java Language Binding

Contents

This appendix is informative, not normative.

D.1 The Java language binding

The Java language binding for the SVG Document Object Model definitions is available at:

```
http://www.w3.org/TR/2011/PR-SVG11-20110609/java-binding.zip
```

Note that this language binding is not normative. The IDL Definitions are the normative parts of the SVG DOM.

D.2 Using SVG with the Java language

When scripting SVG with a language such as ECMAScript, it is possible to embed script code directly in the SVG content using the 'script' element and the event attributes (e.g., 'onload' or 'onclick'). For programming languages with a binary delivery format, such as the Java language, it is not possible to embed the code into the 'script' element or within the event attributes. Therefore there is a need to specify how such languages can bind to and handle events in the SVG document. The following technique describes how this should be done when using the Java language and what is expected of the User Agent that supports dynamic SVG content through the Java language.

- The 'script' element should reference a jar file containing the compiled code to handle the events fired by the document. For example:

  ```
  <script type="application/java-archive" xlink:href="myJavaHandlers.jar"/>
  ```

- The manifest in the referenced jar file contains an entry, SVG-Handler-Class, defining the class responsible for initializing the event listeners on the SVG document. The entry should be a fully qualified class name. For example:

  ```
  Manifest-Version: 1.1
  SVG-Handler-Class: org.example.svg.SVGHandler
  ```

- The class referenced by SVG-Handler-Class implements the EventListenerInitializer interface defined as:

  ```
  public interface EventListenerInitializer {
  ```

```
  void initializeEventListeners(SVGDocument doc);
}
```

- The method initializeEventListeners() will be called by the User Agent before the onload event is triggered.

The event binding of other binary programming languages is not specified.

Appendix E: ECMAScript Language Binding

Contents

This appendix is normative.

This appendix describes how to expose the SVG DOM to an ECMAScript language environment [ECMA-262].

For descriptions of how interfaces and exceptions defined in other specifications are to be exposed to an ECMAScript language environment, consult the relevant specification:

- For DOM Level 2 Core interfaces and exceptions, see the DOM Level 2 Core ECMAScript Language Binding appendix ([DOM2], appendix E).
- For DOM Level 2 Events interfaces, see the DOM Level 2 Events ECMAScript Language Binding appendix ([DOM2EVENTS], appendix C).
- For DOM Level 2 Views interfaces, see the DOM Level 2 Views ECMAScript Language Binding appendix ([DOM2VIEWS], appendix C).
- For DOM Level 2 Style interfaces, see the DOM Level 2 Style ECMAScript Language Binding appendix ([DOM2STYLE], appendix C).

E.1 Exceptions

The SVG DOM defines a single exception, SVGException, which is exposed to an ECMAScript language environment as follows.

The ECMAScript global object has a property named `"SVGException"` whose value is an object with the following properties:

- A read only property named `"SVG_WRONG_TYPE_ERR"`, whose value is the **Number** value 1.
- A read only property named `"SVG_INVALID_VALUE_ERR"`, whose value is the **Number** value 2.
- A read only property named `"SVG_MATRIX_NOT_INVERTABLE"`, whose value is the **Number** value 3.

A host object that is an SVGException also has these three properties on itself, or somewhere in its prototype chain. Every such host object also has a read only property named `"code"` whose value is a **Number**, corresponding to the code exception member on SVGException.

E.2 Constants

For every interface defined in the SVG DOM that has one or more constants defined on it, there exists a property on the ECMAScript global object whose name is the name of the interface, and whose value is an object with a read only property for each of the constants. The name of each of these read only properties is the name of the corresponding constant, and the value is a **Number** with the same value as that of the constant.

A host object that implements an interface with constants defined on it has, on itself or somewhere in its prototype chain, these properties that correspond to the constants.

E.3 Types

The following table gives the corresponding ECMAScript type for the IDL primitive types used in the SVG DOM.

IDL type	ECMAScript type
boolean	**Boolean**
float	**Number**
long	**Number**
unsigned short	**Number**
unsigned long	**Number**
DOMString	**String**

When an ECMAScript **Number** is assigned to a property that corresponds to an attribute of an IDL integer type (long, unsigned short or unsigned long), or it is passed as an argument passed to an operation for which the argument type is an IDL integer type, then behavior is undefined if the **Number** value is not an integer within the range of that type.

For an interface type, a host object that implements the given interface is used.

E.4 Objects

A host object that implements a given interface has properties on itself, or in its prototype chain, that correspond to the operations and attributes defined on that interface and all its superinterfaces.

A property that corresponds to an attribute is read only if the attribute is read only, and has a name that is the same as the name of the attribute. When getting the property, a value of a type according to the Types section above is returned. When setting the property, if it is not read only, then behavior is defined only if a value of a type according to the Types section is assigned to it.

For example, if *a* is a host object that implements the SVGLength interface, then evaluating the statement:

```
a.valueAsString = "10";
```

has defined behavior, but evaluating the statement:

```
a.valueAsString = 10;
```

does not.

A property that corresponds to an operation has a name that is the same as the name of the operation, and has a value that is a **Function** object. The value returned from the **Function** is of a type according to the table in the Types section above. When calling the **Function**, behavior is only defined if the correct number of arguments is passed, and the type of each argument is the type according to the Types table. Also, behavior is only defined for invoking the **Function** with a **this** value that is equal to the object from which the **Function** was obtained.

For example, if *L1* and *L2* are two distinct host objects that implement the SVGPointList interface and *p* is a host object that implements the SVGPoint interface, then evaluating the following statement has defined behavior:

```
L1.insertItemBefore(p, 0);
```

Evaluating any of the following statements, however, does not:

```
L1.insertItemBefore(p, '0');
L1.insertItemBefore(p, -1);
L1.insertItemBefore(p, 0.5);
L1.insertItemBefore(p);
L1.insertItemBefore(p, 0, 0);
L1.insertItemBefore({ x: 10, y: 20 }, 0);
L1.insertItemBefore.call([], p, 0);
L1.insertItemBefore.call(L2, p, 0);
```

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 path 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.
    - As mentioned in Stroke Properties, linecaps must be painted for zero length subpaths when '**stroke-linecap**' has a value of **round** or **square**.
- 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).

φ 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 formulas 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 φ 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).

φ 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

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_x \cos\theta \\ r_y \sin\theta \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \qquad \text{(F.6.3.1)}$$

(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).

θ is the angle from the x-axis of the current coordinate system to the x-axis of the ellipse.

θ ranges from:

- θ_1 which is the start angle of the elliptical arc prior to the stretch and rotate operations.
- θ_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 θ_1, θ_2 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 parameter-*

ization. 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 the following variables:

$c_x \quad c_y \quad r_x \quad r_y \quad \varphi \quad \theta_1 \quad \Delta\theta$

the task is to find:

$x_1 \quad y_1 \quad x_2 \quad y_2 \quad f_A \quad f_S$

This can be achieved using the following formulas:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_x \cos\theta_1 \\ r_y \sin\theta_1 \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{F.6.4.1}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_x \cos(\theta_1 + \Delta\theta) \\ r_y \sin(\theta_1 + \Delta\theta) \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{F.6.4.2}$$

$$f_A = \begin{cases} 1 & \text{if } |\Delta\theta| > 180° \\ 0 & \text{if } |\Delta\theta| \leq 180° \end{cases} \tag{F.6.4.3}$$

$$f_S = \begin{cases} 1 & \text{if } \Delta\theta > 0° \\ 0 & \text{if } \Delta\theta < 0° \end{cases} \tag{F.6.4.4}$$

F.6.5 Conversion from endpoint to center parameterization

Given the following variables:

$x_1 \quad y_1 \quad x_2 \quad y_2 \quad f_A \quad f_S \quad r_x \quad r_y \quad \varphi$

the task is to find:

$c_x \quad c_y \quad \theta_1 \quad \Delta\theta$

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')

$$\begin{pmatrix} x_1' \\ y_1' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} \frac{x_1 - x_2}{2} \\ \frac{y_1 - y_2}{2} \end{pmatrix} \quad \text{(F.6.5.1)}$$

- *Step 2: Compute* (c_x', c_y')

$$\begin{pmatrix} c_x' \\ c_y' \end{pmatrix} = \pm \sqrt{\frac{r_x^2 r_y^2 - r_x^2 (y_1')^2 - r_y^2 (x_1')^2}{r_x^2 (y_1')^2 + r_y^2 (x_1')^2}} \begin{pmatrix} \frac{r_x y_1'}{r_y} \\ -\frac{r_y x_1'}{r_x} \end{pmatrix} \quad \text{(F.6.5.2)}$$

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')

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} c_x' \\ c_y' \end{pmatrix} + \begin{pmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{pmatrix} \quad \text{(F.6.5.3)}$$

- *Step 4: Compute* θ_1 and $\Delta\theta$

  In general, the angle between two vectors (u_x, u_y) and (v_x, v_y) can be computed as

$$\angle\left(\vec{u}, \vec{v}\right) = \pm \arccos \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \ \|\vec{v}\|} \quad \text{(F.6.5.4)}$$

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 θ_1 and $\Delta\theta$ as follows:

$$\theta_1 = \angle\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{x_1' - c_x'}{r_x} \\ \frac{y_1' - c_y'}{r_y} \end{pmatrix}\right) \quad \text{(F.6.5.5)}$$

$$\Delta\theta \equiv \angle\left(\begin{pmatrix} \frac{x_1' - c_x'}{r_x} \\ \frac{y_1' - c_y'}{r_y} \end{pmatrix}, \begin{pmatrix} \frac{-x_1' - c_x'}{r_x} \\ \frac{-y_1' - c_y'}{r_y} \end{pmatrix}\right) \bmod 360° \quad \text{(F.6.5.6)}$$

where θ_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 $f_S = 0$ and the right side of (F.6.5.6) is greater than 0, then subtract 360°, whereas if $f_S = 1$ and the right side of (F.6.5.6) is less than 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:

$$r_x \leftarrow |r_x| \qquad r_y \leftarrow |r_y| \tag{F.6.6.1}$$

- *Step 3: Ensure radii are large enough*

  Using the primed coordinate values of equation (F.6.5.1), compute

$$\Lambda = \frac{(x_1')^2}{r_x^2} + \frac{(y_1')^2}{r_y^2} \tag{F.6.6.2}$$

  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

$$r_x \leftarrow \sqrt{\Lambda}\, r_x \qquad r_y \leftarrow \sqrt{\Lambda}\, r_y \tag{F.6.6.3}$$

- *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.

Appendix G: Conformance Criteria

Contents

This appendix is normative.

G.1 Introduction

In order to ensure that SVG-family documents are maximally portable among SVG-family user agents, this specification rigidly defines conformance requirements for both, as well as for SVG-family document types. While the conformance definitions can be found in this appendix, they necessarily reference normative text within this document and within other related specifications. It is only possible to fully comprehend the conformance requirements of SVG through a complete reading of all normative references.

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) and also:

- is XML well-formed ([XML10], section 2.1),
- conforms to the *Namespaces in XML* specification [XML-NS],
- any CSS stylesheets conform to the core grammar of *Cascading Style Sheets, level 2* [CSS2],
- any `<?xml-stylesheet?>` processing instruction conforms to *Associating stylesheets with XML documents* [XML-SS],
- and the document fragment is determined to be valid as follows:
  1. Let *E* be the encoding of the XML document in which the SVG document fragment resides.
  2. Let *V* be the XML version of the document in which the SVG document fragment resides.
  3. Let *D* be an XML document constructed by concatenating:
     - an XML declaration identifying version *V* and encoding *E* (that is, `<?xml version="V" encoding="E"?>`),

- the DOCTYPE declaration `<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">`, and
- the SVG document fragment with any entities expanded.

4. Remove from *D* any subtree rooted by an element that is not in the SVG namespace.
5. Remove from *D* any attribute that is in a namespace that is not the XLink namespace or the Namespaces in XML namespace.
6. Remove the prefix from the name of any element in *D* that uses one.
7. Let *A* be the set of all attributes in *D* that are in the XLink namespace.
8. Set the attributes **xmlns="http://www.w3.org/2000/svg"** and **xmlns:xlink="http://www.w3.org/1999/xlink"** on *D*'s document element and remove any other attributes in the Namespaces in XML namespace from *D*.
9. Change the prefix of every attribute *A* to be `xlink`.
10. The document fragment is valid if *D* is a valid XML document ([XML10], section 2.8).

SVG document fragments can be included within parent XML documents using the XML namespace facilities described in Namespaces in XML [XML-XS]. Note, however, that since a Conforming SVG Document Fragment must have an 'svg' element as its root, the use of an individual non-'svg' element from the SVG namespace is disallowed. 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 SVG Document Fragment, the file could be modified as follows:

```
<?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>
```

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, and
- the SVG document fragment rooted at the document element is a Conforming SVG Document Fragment.

G.4 Conforming SVG Generators

A *Conforming SVG Generator* is a program which:

- always creates Conforming SVG Document Fragments and/or Conforming SVG Stand-Alone Files.
- does not create non-conforming SVG document fragments 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.5 Conforming SVG Servers

Conforming SVG Servers must meet all the requirements of a Conforming SVG Generator. In addition, Conforming SVG Servers using HTTP or other protocols that use Internet Media types must serve SVG stand-alone files with the media type `"image/svg+xml"`.

Also, if the SVG file is compressed with gzip or deflate, Conforming SVG Servers must indicate this with the appropriate header, according to what the protocol supports. Specifically, for content compressed by the server immediately prior to transfer, the server must use the "Transfer-Encoding: gzip" or "Transfer-Encoding: deflate" headers as appropriate, and for content stored in a compressed format on the server (e.g. with the file extension "svgz"), the server must use the "Content-Encoding: gzip" or "Content-Encoding: deflate" headers as appropriate.

Note: Compression of stored *content* (the "entity," in HTTP terms) is distinct from automatic compression of the *message body*, as defined in HTTP/1.1 TE/ Transfer Encoding ([RFC2616], sections 14.39 and 14.41).

G.6 Conforming SVG DOM Subtree

A DOM subtree rooted at a given element is a *Conforming SVG DOM Subtree* if, once serialized to XML, is a Con-

forming SVG Document Fragment. If the DOM subtree cannot be serialized to XML, such as when a Comment node's data contains the substring "--", then the subtree is not a Conforming SVG DOM Subtree.

G.7 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 XML 1.0 [XML10] and *Namespaces in XML* [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 "http://www.w3.org/TR/SVG11/feature#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 "http://www.w3.org/TR/SVG11/feature#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.8 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 "http://www.w3.org/TR/SVG11/feature#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 "http://www.w3.org/TR/SVG11/feature#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 and PNG image formats [JPEG] [PNG].
- 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 must correctly support gzip-encoded [RFC1952] and deflate-encoded [RFC1951] data streams, for any content type (including SVG, script files, images). SVG implementations that support HTTP must support these encodings according to the HTTP 1.1 specification [RFC2616]; in particular, the client must specify with an "Accept-Encoding:" request header [HTTP-ACCEPT-ENCODING] those encodings that it accepts, including at minimum gzip and deflate, and then decompress any gzip-encoded and deflate-encoded data streams that are downloaded from the server. When an SVG viewer retrieves compressed content (e.g., an .svgz file) over HTTP, if the "Content-Encoding" and "Transfer-Encoding" response headers are missing or specify a value that does not match the compression method that has been applied to the content, then the SVG viewer must not render the content and must treat the document as being in error.
- The viewer must support base64 encoded content using the "data:" URL scheme [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:" URL scheme.)
- 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 (px unit) to the mathematically correct result at the initial 1:1 zoom ratio. It is suggested that viewers attempt to keep a high degree of accuracy when zooming.
- 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 [ECMA-262].

The Web Accessibility Initiative 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 supports *Cascading Style Sheets, level 2* [CSS2], a *Conforming SVG Viewer* must support CSS styling of SVG content and must support all features from CSS2 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 ([CSS2], section 3.2).

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**').

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 (WCAG) 2.0* [WCAG2] 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.

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 *Character Model for the World Wide Web 1.0: Fundamentals* [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.

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 [RFC1952]:

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 ([CSS2], section 15.1) 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.

Appendix K: References

Contents

K.1 Normative references

[ATAG]
Authoring Tool Accessibility Guidelines 1.0, J. Treviranus, J. Richards, I. Jacobs, C. McCathieNevile, eds. World Wide Web Consortium, 03 February 2000.
This edition of ATAG 1.0 is http://www.w3.org/TR/2000/REC-ATAG10-20000203/.
The latest edition of ATAG 1.0 is available at http://www.w3.org/TR/ATAG10/.

[BCP47]
IETF BCP 47 Tags for Identifying Languages, A. Phillips and M. Davis, Editors, September 2009.
Available at http://www.rfc-editor.org/rfc/bcp/bcp47.txt.

[COLORIMETRY]
Colorimetry, Third Edition, Commission Internationale de l'Eclairage, CIE Publication 15:2004, ISBN 3-901-906-33-9.
Available at http://www.cie.co.at/publ/abst/15-2004.html.

[CSS2]
Cascading Style Sheets, level 2, B. Bos, H. W. Lie, C. Lilley, I. Jacobs, eds. World Wide Web Consortium, 11 April 2008.
This edition of CSS2 is http://www.w3.org/TR/2008/REC-CSS2-20080411/ and is no longer maintained.
The latest edition of CSS2 is available at http://www.w3.org/TR/CSS2/. The CSS Working Group encourages authors and implementors to reference CSS 2.1 (or its successor) instead of this document and, when features common to CSS2 and CSS 2.1 are defined differently to follow the definitions in CSS 2.1. A list of changes between CSS2 and CSS 2.1 may be helpful.

[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, eds. World Wide Web Consortium, 01 October 1998.
This edition of DOM Level 1 is http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/.
The latest edition of DOM Level 1 is 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, eds. World Wide Web Consortium, 13 November 2000.
This edition of DOM Level 2 Core is http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.
The latest edition of DOM Level 2 Core is available at http://www.w3.org/TR/DOM-Level-2-Core/.

[DOM2EVENTS]

Document Object Model (DOM) Level 2 Events Specification, T. Pixley, ed. World Wide Web Consortium, 13 November 2000.

This edition of DOM 2 Events is http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/.

The latest edition of DOM 2 Events is available at http://www.w3.org/TR/DOM-Level-2-Events/.

[DOM2STYLE]

Document Object Model (DOM) Level 2 Style Specification, C. Wilson, P. Le Hégaret, V. Apparao, eds. World Wide Web Consortium, 13 November 2000.

This edition of DOM Level 2 Style is http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113/.

The latest edition of DOM Level 2 Style is available at http://www.w3.org/TR/DOM-Level-2-Style/.

[DOM2VIEWS]

Document Object Model (DOM) Level 2 Views Specification, A. Le Hors, L. Cable, eds. World Wide Web Consortium, 13 November 2000.

This edition of DOM 2 Views is http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113/.

The latest edition of DOM 2 Views is available at http://www.w3.org/TR/DOM-Level-2-Views/.

[ECMA-262]

ECMAScript Language Specification, 5th Edition, M. Cowlishaw, ed. Ecma International, December 2009.

Available at http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[ICC42]

Specification ICC.1:2004-10, File Format for Color Profiles, Profile Version 4.2.0.0 with errata incorporated, 5/22/20006, International Color Consortium, 2006.

Available at http://www.color.org/ICC1v42_2006-05.pdf.

This specification is substantially identical to ISO 15076-1:2005.

The ICC list some approved revisions to ICC.1:2004-10.

[ISO8601]

Data elements and interchange formats - Information interchange - Representation of dates and times, International Organization for Standardization, 2004. Available at http://www.iso.org/iso/catalogue_detail?csnumber=40874.

[JPEG]

ISO/IEC 10918-1:1994/Cor 1:2005: Information Technology — Digital Compression And Coding Of Continuous-tone Still Images, International Organization for Standardization, September 2005.

Available at http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18902.

An older version is available at http://www.w3.org/Graphics/JPEG/itu-t81.pdf.

[GML]

OpenGIS Geography Markup Language (GML) Encoding Standard, version 3.2.1, C. Portele, ed. Open GIS Consortium, 27 August 2007.

Available at http://portal.opengeospatial.org/files/?artifact_id=20509.

[PNG]

Portable Network Graphics (PNG) Specification (Second Edition): Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification, ISO/IEC 15948:2003 (E), D. Duce, ed. World Wide Web Consortium, 10 November 2003.

This edition of PNG is http://www.w3.org/TR/2003/REC-PNG-20031110/.
The latest edition of PNG is available at http://www.w3.org/TR/PNG/.

[PORTERDUFF]
Compositing Digital Images, T. Porter and T. Duff. SIGGRAPH '84 Conference Proceedings, Association for Computing Machinery, Volume 18, Number 3, July 1984.

[RFC1951]
DEFLATE Compressed Data Format Specification version 1.3, P. Deutsch, May 1996.
Available at http://www.ietf.org/rfc/rfc1951.txt.

[RFC1952]
GZIP file format specification version 4.3, P. Deutsch, May 1996.
Available at http://www.ietf.org/rfc/rfc1952.txt.

[RFC2046]
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed and N. Borenstein, November 1996. (Note that this RFC obsoletes RFC 1521, RFC 1522 and RFC 1590.)
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.

[RFC2318]
The text/css Media Type, H. Lie, B. Bos, C. Lilley, March 1998.
Available at http://www.ietf.org/rfc/rfc2318.txt.

[RFC2397]
The "data" URL scheme, L. Masinter, August 1998.
Available at http://www.ietf.org/rfc/rfc2397.

[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. (Note that this RFC obsoletes RFC 2068.)
Available at http://www.ietf.org/rfc/rfc2616.

[RFC2732]
Format for Literal IPv6 Addresses in URL's, 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.

[RFC3629]
UTF-8, a transformation format of ISO 10646, F. Yergeau, November 2003. (Note that this RFC obsoletes RFC 2044 and RFC 2279.)
Available at http://www.ietf.org/rfc/rfc3629.txt.

[RFC3986]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, January 2005. (Note that RFC 3986 updates RFC 1738 and obsoletes RFC 2732, RFC 2396 and RFC 1808.)
Available at http://tools.ietf.org/html/rfc3986.

[RFC3987]
Internationalized Resource Identifiers (IRIs), M. Dürst, M. Suignard, January 2005.
Available at http://tools.ietf.org/html/rfc3987.

[RFC4329]
Scripting Media Types, B. Höhrmann, April 2006.
Available at http://www.ietf.org/rfc/rfc4329.txt.

[SMILANIM]
SMIL Animation, P. Schmitz, A. Cohen. World Wide Web Consortium, 04 September 2001.
This edition of SMIL Animation is http://www.w3.org/TR/2001/REC-smil-animation-20010904/.
The latest edition of SMIL Animation is available at http://www.w3.org/TR/smil-animation/.

[SRGB]
IEC 61966-2-1/Amd 1:2003 : Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB, International Electrotechnical Commission, 2003.
Available at http://webstore.iec.ch/webstore/webstore.nsf/artnum/025408 and at http://www.colour.org/tc8-05/Docs/colorspace/61966-2-1.pdf.
See also http://www.color.org/chardata/rgb/srgb.xalter for technical data and color profiles.

[UAX9]
Unicode Bidirectional Algorithm, The Unicode Standard Annex #9. The Unicode Consortium, 2010.
Available at http://www.unicode.org/reports/tr9/.

[UNICODE]
The Unicode Standard, Version 6.0.0, The Unicode Consortium, Mountain View, CA, 2011. ISBN 978-1-936213-01-6.
Available at http://www.unicode.org/versions/Unicode6.0.0.

[XLINK]
XML Linking Language (XLink) Version 1.1, S. DeRose, E. Maler, D. Orchard, N. Walsh, eds. World Wide Web Consortium, 06 May 2010.
This edition of XLink 1.1 is http://www.w3.org/TR/2010/REC-xlink11-20100506/.
The latest edition of XLink 1.1 is available at http://www.w3.org/TR/xlink11/.

[XML10]
Extensible Markup Language (XML) 1.0 (Fifth Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, eds. World Wide Web Consortium, 26 November 2008.
This edition of XML 1.0 is http://www.w3.org/TR/2008/REC-xml-20081126/.
The latest edition of XML 1.0 is available at http://www.w3.org/TR/xml/.

[XML-BASE]
XML Base (Second Edition), J. Marsh, R. Tobin, eds. World Wide Web Consortium, 28 January 2009.
This edition of XML Base is http://www.w3.org/TR/2009/REC-xmlbase-20090128/.
The latest edition of XML Base is available at http://www.w3.org/TR/xmlbase/.

[XML-NS]
Namespaces in XML 1.0 (Third Edition), T. Bray, D. Hollander, A. Layman, R. Tobin, H. Thompson, eds. World Wide Web Consortium, 8 December 2009.

This edition of Namespaces in XML is http://www.w3.org/TR/2009/REC-xml-names-20091208/.
The latest edition of Namespaces in XML is available at http://www.w3.org/TR/xml-names/.

[XML-SS]
Associating Style Sheets with XML documents, Version 1.0, J. Clark, ed. World Wide Web Consortium, 29 June 1999.
This edition of XML Stylsheet is http://www.w3.org/1999/06/REC-xml-stylesheet-19990629/.
The latest edition of XML Stylesheet is available at http://www.w3.org/TR/xml-stylesheet/.

[XSL]
Extensible Stylesheet Language (XSL) Version 1.1, A. Berglund, ed. World Wide Web Consortium, 05 December 2006.
This edition of XSL is http://www.w3.org/TR/2006/REC-xsl11-20061205/.
The latest edition of XSL is available at http://www.w3.org/TR/xsl/.

K.2 Informative references

[CHARMOD]
Character Model for the World Wide Web 1.0: Fundametnals, M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, eds. World Wide Web Consortium, 15 February 2005.
This edition of Charmod Fundamentals is http://www.w3.org/TR/2005/REC-charmod-20050215/.
The latest edition of Charmod Fundamentals is available at http://www.w3.org/TR/charmod/.

[DCORE]
Dublin Core Metadata Initiative.
Available at http://dublincore.org/.

[DOM3]
Document Object Model (DOM) Level 3 Core Specification, A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, eds. World Wide Web Consortium, 07 April 2004.
This edition of DOM 3 Core is http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.
The latest edition of DOM 3 Core is available at http://www.w3.org/TR/DOM-Level-3-Core/.

[FOLEY-VANDAM]
Computer Graphics: Principles and Practice, Second Edition, J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, R. L. Phillips. Addison-Wesley, 1995.

[HTML4]
HTML 4.01 Specification, D. Raggett, A. Le Hors, I. Jacobs. World Wide Web Consortium, 24 December 1999.
This edition of HTML 4 is http://www.w3.org/TR/1999/REC-html401-19991224/.
The latest edition of HTML 4 is available at http://www.w3.org/TR/html4/.

[MATHML]
Mathematical Markup Language (MathML) Version 2.0 (Second Edition), D. Carlisle, P. Ion, R. Miner, N. Poppelier, eds. World Wide Web Consortium, 21 October 2003.
This edition of MathML 2 is http://www.w3.org/TR/2003/REC-MathML2-20031021/.
The latest edition of MathML 2 is available at http://www.w3.org/TR/MathML2/.

[MIMETYPES]

MIME Media Types, Internet Assigned Numbers Authority.
Available at http://www.iana.org/assignments/media-types/.

[NVDL]

Information Technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language: ISO/IEC 19757-4:2006/Cor 1:2008, International Organization for Standardization, December 2005.
Available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c038615_ISO_IEC_19757-4_2006(E).zip.
See also http://nvdl.org/.

[OPENTYPE]

OpenType Specification Version 1.6. July 2009.
Available at http://www.microsoft.com/typography/otspec160/.
(Note that this is technically equivalent to ISO/IEC 14496-22:2009 (Second Edition) "Open Font Format", available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c052136_ISO_IEC_14496-22_2009(E).zip.)

[RDF-PRIMER]

RDF Primer, F. Manolas, E. Miller, eds. World Wide Web Consortium, 10 February 2004.
This edition of RDF Primer is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.
The latest edition of RDF Primer is available at http://www.w3.org/TR/rdf-primer/.

[SCHEMA2]

XML Schema Part 2: Datatypes Second Edition. P. Biron, A. Malhotra, eds. World Wide Web Consortium, 28 October 2004. (See also *Processing XML 1.1 documents with XML Schema 1.0 processors*.)
This edition of XML Schema Part 2 is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.
The latest edition of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2/.

[SELECTORS]

Selectors Level 3, T. Çelik, E. Etemad, D. Glazman, I. Hickson, P. Linss, J. Williams, eds. World Wide Web Consortium, 15 December 2009.
This edition of Selectors Level 3 is http://www.w3.org/TR/2009/PR-css3-selectors-20091215/.
The latest edition of Selectors Level 3 is available at http://www.w3.org/TR/css3-selectors/.

[SVG-ACCESS]

Accessibility Features of SVG, C. McCathieNevile, M. Koivunen, eds. World Wide Web Consortium, 07 August 2000.
This edition of Accessibility Features of SVG is http://www.w3.org/TR/2000/NOTE-SVG-access-20000807/.
The latest edition of Accessibility Features of SVG is available at http://www.w3.org/TR/SVG-access/.

[SVG-COMPOSITING]

SVG Compositing Specification, A. Grasso, ed. World Wide Web Consortium, 30 April 2009.
This edition of SVG Compositing is http://www.w3.org/TR/2009/WD-SVGCompositing-20090430/.
The latest edition of SVG Compositing is available at http://www.w3.org/TR/SVGCompositing/.

[SMIL]

Synchronized Multimedia Integration Language (SMIL 3.0), D. Bulterman *et al.*, eds. 01 December 2008.
This edition of SMIL is http://www.w3.org/TR/2008/REC-SMIL3-20081201/.
The latest edition of SMIL is available at http://www.w3.org/TR/smil/.

[SVG10]

Scalable Vector Graphics (SVG) 1.0, J. Ferraiolo, ed. 04 September 2001.
This edition of SVG 1.0 is http://www.w3.org/TR/2003/REC-SVG11-20030114/.
The latest edition of SVG 1.0 is available at http://www.w3.org/TR/SVG10/.

[UAAG]

User Agent Accessibility Guidelines 1.0, I. Jacobs, J. Gunderson, E. Hansen, eds. 17 December 2002.
This edition of UAAG is http://www.w3.org/TR/2002/REC-UAAG10-20021217/.
The latest edition of UAAG is available at http://www.w3.org/TR/UAAG10/.

[WCAG2]

Web Content Accessibility Guidelines (WCAG) 2.0, B. Caldwell, M. Cooper, L. Reid, G. Vanderheiden, eds. World Wide Web Consortium, 11 December 2008.
This edition of WCAG 2.0 is http://www.w3.org/TR/2008/REC-WCAG20-20081211/.
The latest edition of WCAG 2.0 is available at http://www.w3.org/TR/WCAG20/.

[WINDOW]

Window Object 1.0, I. Davis, M. Stachowiak, eds. World Wide Web Consortium, work in progress, 07 April 2006.
This edition of Window Object 1.0 is http://www.w3.org/TR/2006/WD-Window-20060407/.
The latest edition of Window Object 1.0 is available at http://www.w3.org/TR/Window/.

[XHTML]

XHTML™ 1.0: The Extensible HyperText Markup Language (Second Edition), S. Pemberton, ed. World Wide Web Consortium, 1 August 2002.
This edition of XHTML 1 is http://www.w3.org/TR/2002/REC-xhtml1-20020801/.
The latest edition of XHTML 1 is available at http://www.w3.org/TR/xhtml1/.

[XHTMLplusMathMLplusSVG]

An XHTML + MathML + SVG Profile, 石川 雅康 (ISHIKAWA Masayasu), ed. World Wide Web Consortium, work in progress, 09 August 2002.
This edition of XHTML + MathML + SVG is http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020809/.
The latest edition of XHTML + MathML + SVG is available at http://www.w3.org/TR/XHTMLplusMathMLplusSVG/.

[XSLT]

XSL Transformations (XSLT) Version 1.0, J. Clark, ed. World Wide Web Consortium, 16 November 1999.
This edition of XSLT 1.0 is http://www.w3.org/TR/1999/REC-xslt-19991116.
The latest edition of XSLT 1.0 is available at http://www.w3.org/TR/xslt.

[XSLT2]

XSL Transformations (XSLT) Version 2.0, M. Kay, ed. World Wide Web Consortium, 23 January 2007.
This edition of XSLT 2.0 is http://www.w3.org/TR/2007/REC-xslt20-20070123/.
The latest edition of XSLT 2.0 is available at http://www.w3.org/TR/xslt20/.

Appendix L: Element Index

This appendix is informative, not normative.

The following are the elements in the SVG language:

- ‘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’

Appendix M: Attribute Index

Contents

This appendix is informative, not normative.

M.1 Regular attributes

The following table lists all of the attributes defined in the SVG language, except for the presentation attributes, which are treated in the Presentation attributes section below. For each attribute, the elements on which the attribute may be specified is also given.

Attribute	Elements on which the attribute may be specified	Anim.
'accent-height'	'font-face'	
'accumulate'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'additive'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'alphabetic'	'font-face'	
'amplitude'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'arabic-form'	'glyph'	
'ascent'	'font-face'	
'attributeName'	'animate', 'animateColor', 'animateTransform', 'set'	
'attributeType'	'animate', 'animateColor', 'animateTransform', 'set'	
'azimuth'	'feDistantLight'	√
'baseFrequency'	'feTurbulence'	√
'baseProfile'	'svg'	
'bbox'	'font-face'	
'begin'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'bias'	'feConvolveMatrix'	√
'by'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'calcMode'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'cap-height'	'font-face'	
'class'	'a', 'altGlyph', 'circle', 'clipPath', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence', 'filter', 'font', 'foreignObject', 'g', 'glyph', 'glyphRef', 'image', 'line', 'linearGradient', 'marker', 'mask', 'missing-glyph', 'path', 'pattern', 'polygon', 'polyline', 'radialGradient', 'rect', 'stop', 'svg', 'switch', 'symbol', 'text', 'textPath', 'title', 'tref', 'tspan', 'use'	√
'clipPathUnits'	'clipPath'	√
'contentScriptType'	'svg'	
'contentStyleType'	'svg'	
'cx'	'circle'	√

'cx'	'ellipse'	√
'cx'	'radialGradient'	√
'cy'	'circle'	√
'cy'	'ellipse'	√
'cy'	'radialGradient'	√
'd'	'path'	√
'd'	'glyph', 'missing-glyph'	
'descent'	'font-face'	
'diffuseConstant'	'feDiffuseLighting'	√
'divisor'	'feConvolveMatrix'	√
'dur'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'dx'	'altGlyph'	√
'dx'	'feOffset'	√
'dx'	'glyphRef'	
'dx'	'text'	√
'dx'	'tref', 'tspan'	√
'dy'	'altGlyph'	√
'dy'	'feOffset'	√
'dy'	'glyphRef'	
'dy'	'text'	√
'dy'	'tref', 'tspan'	√
'edgeMode'	'feConvolveMatrix'	√
'elevation'	'feDistantLight'	√
'end'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'exponent'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'externalResourcesRequired'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'cursor', 'defs', 'ellipse', 'feImage', 'filter', 'font', 'foreignObject', 'g', 'image', 'line', 'linearGradient', 'marker', 'mask', 'mpath',	

	'path', 'pattern', 'polygon', 'polyline', 'radialGradient', 'rect', 'script', 'set', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use', 'view'	
'fill'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'filterRes'	'filter'	√
'filterUnits'	'filter'	√
'font-family'	'font-face'	
'font-size'	'font-face'	
'font-stretch'	'font-face'	
'font-style'	'font-face'	
'font-variant'	'font-face'	
'font-weight'	'font-face'	
'format'	'altGlyph'	
'format'	'glyphRef'	
'from'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'fx'	'radialGradient'	√
'fy'	'radialGradient'	√
'g1'	'hkern', 'vkern'	
'g2'	'hkern', 'vkern'	
'glyph-name'	'glyph'	
'glyphRef'	'altGlyph'	
'glyphRef'	'glyphRef'	
'gradientTransform'	'linearGradient'	√
'gradientTransform'	'radialGradient'	√
'gradientUnits'	'linearGradient'	√
'gradientUnits'	'radialGradient'	√
'hanging'	'font-face'	
'height'	'filter'	√

'height'	'foreignObject'	√
'height'	'image'	√
'height'	'pattern'	√
'height'	'rect'	√
'height'	'svg'	√
'height'	'use'	√
'height'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence'	√
'height'	'mask'	√
'horiz-adv-x'	'font'	
'horiz-adv-x'	'glyph', 'missing-glyph'	
'horiz-origin-x'	'font'	
'horiz-origin-y'	'font'	
'id'	'a', 'altGlyph', 'altGlyphDef', 'altGlyphItem', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'color-profile', 'cursor', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feDistantLight', 'feFlood', 'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', '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'	
'ideographic'	'font-face'	
'in'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feGaussianBlur', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile'	√
'in2'	'feBlend'	√
'in2'	'feComposite'	√
'in2'	'feDisplacementMap'	√
'intercept'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'k'	'hkern', 'vkern'	

'k1'	'feComposite'	√
'k2'	'feComposite'	√
'k3'	'feComposite'	√
'k4'	'feComposite'	√
'kernelMatrix'	'feConvolveMatrix'	√
'kernelUnitLength'	'feConvolveMatrix'	√
'kernelUnitLength'	'feDiffuseLighting'	√
'kernelUnitLength'	'feSpecularLighting'	√
'keyPoints'	'animateMotion'	
'keySplines'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'keyTimes'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'lang'	'glyph'	
'lengthAdjust'	'text', 'textPath', 'tref', 'tspan'	√
'limitingConeAngle'	'feSpotLight'	√
'local'	'color-profile'	
'markerHeight'	'marker'	√
'markerUnits'	'marker'	√
'markerWidth'	'marker'	√
'maskContentUnits'	'mask'	√
'maskUnits'	'mask'	√
'mathematical'	'font-face'	
'max'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'media'	'style'	
'method'	'textPath'	√
'min'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'mode'	'feBlend'	√
'name'	'color-profile'	

'name'	'font-face-name'	
'numOctaves'	'feTurbulence'	√
'offset'	'stop'	√
'offset'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'onabort'	'svg'	
'onactivate'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onbegin'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'onclick'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onend'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'onerror'	'svg'	
'onfocusin'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onfocusout'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onload'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onload'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'onmousedown'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onmousemove'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onmouseout'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onmouseover'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onmouseup'	'a', 'altGlyph', 'circle', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan', 'use'	
'onrepeat'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	

'onresize'	'svg'	
'onscroll'	'svg'	
'onunload'	'svg'	
'onzoom'	'svg'	
'operator'	'feComposite'	√
'operator'	'feMorphology'	√
'order'	'feConvolveMatrix'	√
'orient'	'marker'	√
'orientation'	'glyph'	
'origin'	'animateMotion'	
'overline-position'	'font-face'	
'overline-thickness'	'font-face'	
'panose-1'	'font-face'	
'path'	'animateMotion'	
'pathLength'	'path'	√
'patternContentUnits'	'pattern'	√
'patternTransform'	'pattern'	√
'patternUnits'	'pattern'	√
'points'	'polygon'	√
'points'	'polyline'	√
'pointsAtX'	'feSpotLight'	√
'pointsAtY'	'feSpotLight'	√
'pointsAtZ'	'feSpotLight'	√
'preserveAlpha'	'feConvolveMatrix'	√
'preserveAspectRatio'	'feImage', 'image', 'marker', 'pattern', 'svg', 'symbol', 'view'	√
'primitiveUnits'	'filter'	√
'r'	'circle'	√

'r'	'radialGradient'	√
'radius'	'feMorphology'	√
'refX'	'marker'	√
'refY'	'marker'	√
'rendering-intent'	'color-profile'	
'repeatCount'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'repeatDur'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'requiredExtensions'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'cursor', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'mask', 'path', 'pattern', 'polygon', 'polyline', 'rect', 'set', 'svg', 'switch', 'text', 'textPath', 'tref', 'tspan', 'use'	
'requiredFeatures'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'cursor', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'mask', 'path', 'pattern', 'polygon', 'polyline', 'rect', 'set', 'svg', 'switch', 'text', 'textPath', 'tref', 'tspan', 'use'	
'restart'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'result'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence'	√
'rotate'	'altGlyph'	√
'rotate'	'animateMotion'	
'rotate'	'text'	√
'rotate'	'tref', 'tspan'	√
'rx'	'ellipse'	√
'rx'	'rect'	√
'ry'	'ellipse'	√
'ry'	'rect'	√
'scale'	'feDisplacementMap'	√
'seed'	'feTurbulence'	√
'slope'	'font-face'	
'slope'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√

'spacing'	'textPath'	√
'specularConstant'	'feSpecularLighting'	√
'specularExponent'	'feSpecularLighting'	√
'specularExponent'	'feSpotLight'	√
'spreadMethod'	'linearGradient'	√
'spreadMethod'	'radialGradient'	√
'startOffset'	'textPath'	√
'stdDeviation'	'feGaussianBlur'	√
'stemh'	'font-face'	
'stemv'	'font-face'	
'stitchTiles'	'feTurbulence'	√
'strikethrough-position'	'font-face'	
'strikethrough-thickness'	'font-face'	
'string'	'font-face-format'	
'style'	'a', 'altGlyph', 'circle', 'clipPath', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence', 'filter', 'font', 'foreignObject', 'g', 'glyph', 'glyphRef', 'image', 'line', 'linearGradient', 'marker', 'mask', 'missing-glyph', 'path', 'pattern', 'polygon', 'polyline', 'radialGradient', 'rect', 'stop', 'svg', 'switch', 'symbol', 'text', 'textPath', 'title', 'tref', 'tspan', 'use'	
'surfaceScale'	'feDiffuseLighting'	√
'surfaceScale'	'feSpecularLighting'	√
'systemLanguage'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'cursor', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'mask', 'path', 'pattern', 'polygon', 'polyline', 'rect', 'set', 'svg', 'switch', 'text', 'textPath', 'tref', 'tspan', 'use'	
'tableValues'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'target'	'a'	√
'targetX'	'feConvolveMatrix'	√
'targetY'	'feConvolveMatrix'	√
'textLength'	'text'	√

'textLength'	'textPath', 'tref', 'tspan'	√
'title'	'style'	
'to'	'set'	
'to'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'transform'	'a', 'circle', 'clipPath', 'defs', 'ellipse', 'foreignObject', 'g', 'image', 'line', 'path', 'polygon', 'polyline', 'rect', 'switch', 'text', 'use'	√
'type'	'animateTransform'	
'type'	'feColorMatrix'	√
'type'	'feTurbulence'	√
'type'	'script'	
'type'	'style'	
'type'	'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR'	√
'u1'	'hkern', 'vkern'	
'u2'	'hkern', 'vkern'	
'underline-position'	'font-face'	
'underline-thickness'	'font-face'	
'unicode'	'glyph'	
'unicode-range'	'font-face'	
'units-per-em'	'font-face'	
'v-alphabetic'	'font-face'	
'v-hanging'	'font-face'	
'v-ideographic'	'font-face'	
'v-mathematical'	'font-face'	
'values'	'feColorMatrix'	√
'values'	'animate', 'animateColor', 'animateMotion', 'animateTransform'	
'version'	'svg'	
'vert-adv-y'	'font'	

'vert-adv-y'	'glyph', 'missing-glyph'	
'vert-origin-x'	'font'	
'vert-origin-x'	'glyph', 'missing-glyph'	
'vert-origin-y'	'font'	
'vert-origin-y'	'glyph', 'missing-glyph'	
'viewBox'	'marker', 'pattern', 'svg', 'symbol', 'view'	√
'viewTarget'	'view'	
'width'	'filter'	√
'width'	'foreignObject'	√
'width'	'image'	√
'width'	'pattern'	√
'width'	'rect'	√
'width'	'svg'	√
'width'	'use'	√
'width'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence'	√
'width'	'mask'	√
'widths'	'font-face'	
'x'	'altGlyph'	√
'x'	'cursor'	√
'x'	'fePointLight'	√
'x'	'feSpotLight'	√
'x'	'filter'	√
'x'	'foreignObject'	√
'x'	'glyphRef'	
'x'	'image'	√

'x'	'pattern'	√
'x'	'rect'	√
'x'	'svg'	√
'x'	'text'	√
'x'	'use'	√
'x'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence'	√
'x'	'mask'	√
'x'	'tref', 'tspan'	√
'x-height'	'font-face'	
'x1'	'line'	√
'x1'	'linearGradient'	√
'x2'	'line'	√
'x2'	'linearGradient'	√
'xChannelSelector'	'feDisplacementMap'	√
'xlink:actuate'	'a'	
'xlink:actuate'	'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'mpath', 'pattern', 'script', 'set', 'use'	
'xlink:arcrole'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'linearGradient', 'mpath', 'pattern', 'radialGradient', 'script', 'set', 'textPath', 'tref', 'use'	
'xlink:href'	'a'	√
'xlink:href'	'altGlyph'	
'xlink:href'	'color-profile'	
'xlink:href'	'cursor'	√
'xlink:href'	'feImage'	√
'xlink:href'	'filter'	√

'xlink:href'	'font-face-uri'	
'xlink:href'	'glyphRef'	
'xlink:href'	'image'	√
'xlink:href'	'linearGradient'	√
'xlink:href'	'mpath'	
'xlink:href'	'pattern'	√
'xlink:href'	'radialGradient'	√
'xlink:href'	'script'	
'xlink:href'	'textPath'	√
'xlink:href'	'use'	√
'xlink:href'	'animate', 'animateColor', 'animateMotion', 'animateTransform', 'set'	
'xlink:href'	'tref'	√
'xlink:role'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'linearGradient', 'mpath', 'pattern', 'radialGradient', 'script', 'set', 'textPath', 'tref', 'use'	
'xlink:show'	'a'	
'xlink:show'	'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'mpath', 'pattern', 'script', 'set', 'use'	
'xlink:title'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'linearGradient', 'mpath', 'pattern', 'radialGradient', 'script', 'set', 'textPath', 'tref', 'use'	
'xlink:type'	'a', 'altGlyph', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'color-profile', 'cursor', 'feImage', 'filter', 'font-face-uri', 'glyphRef', 'image', 'linearGradient', 'mpath', 'pattern', 'radialGradient', 'script', 'set', 'textPath', 'tref', 'use'	
'xml:base'	'a', 'altGlyph', 'altGlyphDef', 'altGlyphItem', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'color-profile', 'cursor', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feDistantLight', 'feFlood', 'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', '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'	
'xml:lang'	'a', 'altGlyph', 'altGlyphDef', 'altGlyphItem', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'color-profile', 'cursor', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feDistantLight', 'feFlood', 'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', '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'	
'xml:space'	'a', 'altGlyph', 'altGlyphDef', 'altGlyphItem', 'animate', 'animateColor', 'animateMotion', 'animateTransform', 'circle', 'clipPath', 'color-profile', 'cursor', 'defs', 'desc', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feDistantLight', 'feFlood', 'feFuncA', 'feFuncB', 'feFuncG', 'feFuncR', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', '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'	
'y'	'altGlyph'	√
'y'	'cursor'	√
'y'	'fePointLight'	√
'y'	'feSpotLight'	√
'y'	'filter'	√
'y'	'foreignObject'	√
'y'	'glyphRef'	
'y'	'image'	√
'y'	'pattern'	√
'y'	'rect'	√
'y'	'svg'	√
'y'	'text'	√
'y'	'use'	√

'y'	'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence'	√
'y'	'mask'	√
'y'	'tref', 'tspan'	√
'y1'	'line'	√
'y1'	'linearGradient'	√
'y2'	'line'	√
'y2'	'linearGradient'	√
'yChannelSelector'	'feDisplacementMap'	√
'z'	'fePointLight'	√
'z'	'feSpotLight'	√
'zoomAndPan'	'svg', 'view'	

M.2 Presentation attributes

As described in the Styling chapter, for each property there exists a corresponding presentation attribute. The table below lists the presentation attributes and the elements on which they may be specified.

Presentation attributes	Elements on which the attributes may be specified
'alignment-baseline', 'baseline-shift', 'clip-path', 'clip-rule', 'clip', 'color-interpolation-filters', 'color-interpolation', 'color-profile', 'color-rendering', 'color', 'cursor', 'direction', 'display', 'dominant-baseline', 'enable-background', 'fill-opacity', 'fill-rule', 'fill', 'filter', 'flood-color', 'flood-opacity', 'font-family', 'font-size-adjust', 'font-size', 'font-stretch', 'font-style', 'font-variant', 'font-weight', 'glyph-orientation-horizontal', 'glyph-orientation-vertical', 'image-rendering', 'kerning', 'letter-spacing', 'lighting-color', 'marker-end', 'marker-mid', 'marker-start', 'mask', 'opacity', 'overflow', 'pointer-events', 'shape-rendering', 'stop-color', 'stop-opacity', 'stroke-dasharray', 'stroke-dashoffset', 'stroke-linecap', 'stroke-linejoin', 'stroke-miterlimit', 'stroke-opacity', 'stroke-width', 'stroke', 'text-anchor', 'text-decoration', 'text-rendering', 'unicode-bidi', 'visibility', 'word-spacing' and 'writing-mode'	'a', 'altGlyph', 'animate', 'animateColor', 'circle', 'clipPath', 'defs', 'ellipse', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feTurbulence', 'filter', 'font', 'foreignObject', 'g', 'glyph', 'glyphRef', 'image', 'line', 'linearGradient', 'marker', 'mask', 'missing-glyph', 'path', 'pattern', 'polygon', 'polyline', 'radialGradient', 'rect', 'stop', 'svg', 'switch', 'symbol', 'text', 'textPath', 'tref', 'tspan' and 'use'

Appendix N: Property Index

This appendix is informative, not normative.

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
'alignment-baseline'	auto | baseline | before-edge | text-before-edge | middle | central | after-edge | text-after-edge | ideographic | alphabetic | hanging | mathematical | inherit	see property description	**'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** elements	no	N/A	visual	yes
'baseline-shift'	baseline | sub | super | <percentage> | <length> | inherit	baseline	**'tspan'**, **'tref'**, **'altGlyph'**, **'textPath'** elements	no	refer 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
'clip'	<shape> | auto | inherit	auto	elements which establish a new viewport, **'pattern'** elements and **'marker'** elements	no	N/A	visual	yes
'clip-path'	<funciri> | none | inherit	none	container elements and graphics elements	no	N/A	visual	yes
'clip-rule'	nonzero | evenodd | inherit	nonzero	graphics elements within a **'clipPath'** element	yes	N/A	visual	yes
'color'	<color> | inherit	depends on user agent	elements to which properties **'fill'**, **'stroke'**, **'stop-color'**,	yes	N/A	visual	yes

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
			'flood-color', **'lighting-color'** apply				
'color-interpolation'	auto | sRGB | linearRGB | inherit	sRGB	container elements, graphics elements and **'animateColor'**	yes	N/A	visual	yes
'color-interpolation-filters'	auto | sRGB | linearRGB | inherit	linearRGB	filter primitives	yes	N/A	visual	yes
'color-profile'	auto | sRGB | <name> | <funciri> | inherit	auto	**'image'** elements that refer to raster images	yes	N/A	visual	yes
'color-rendering'	auto | optimizeSpeed | optimizeQuality | inherit	auto	container elements, graphics elements and **'animateColor'**	yes	N/A	visual	yes
'cursor'	[[<funciri> ,]* [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	N/A	visual, interactive	yes
'direction'	ltr | rtl | inherit	ltr	text content elements	yes	N/A	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	N/A	all	yes

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
'dominant-baseline'	auto | use-script | no-change | reset-size | ideographic | alphabetic | hanging | mathematical | central | middle | text-after-edge | text-before-edge | inherit	auto	text content elements	no	N/A	visual	yes
'enable-background'	accumulate | new [<x> <y> <width> <height>] | inherit	accumulate	container elements	no	N/A	visual	no
'fill'	<paint> (See Specifying paint)	black	shapes and text content elements	yes	N/A	visual	yes
'fill-opacity'	<opacity-value> | inherit	1	shapes and text content elements	yes	N/A	visual	yes
'fill-rule'	nonzero | evenodd | inherit	nonzero	shapes and text content elements	yes	N/A	visual	yes
'filter'	<funciri> | none | inherit	none	container elements and graphics elements	no	N/A	visual	yes
'flood-color'	currentColor | <color> [<icccolor>] | inherit	black	**'feFlood'** elements	no	N/A	visual	yes
'flood-opacity'	<opacity-value> | inherit	1	**'feFlood'** elements	no	N/A	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	see individual properties	visual	yes [1]
'font-family'	[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>] | inherit	depends on user agent	text content elements	yes	N/A	visual	yes

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
'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	N/A	visual	yes [1]
'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	N/A	visual	yes
'font-style'	normal | italic | oblique | inherit	normal	text content elements	yes	N/A	visual	yes
'font-variant'	normal | small-caps | inherit	normal	text content elements	yes	N/A	visual	yes
'font-weight'	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	normal	text content elements	yes	N/A	visual	yes
'glyph-orientation-horizontal'	<angle> | inherit	0deg	text content elements	yes	N/A	visual	no
'glyph-orientation-vertical'	auto | <angle> | inherit	auto	text content elements	yes	N/A	visual	no
'image-rendering'	auto | optimizeSpeed | optimizeQuality | inherit	auto	images	yes	N/A	visual	yes
'kerning'	auto | <length> | inherit	auto	text content elements	yes	N/A	visual	yes
'letter-spacing'	normal | <length> | inherit	normal	text content elements	yes	N/A	visual	yes
'lighting-color'	currentColor |	white	**'feDiffuseLighting'** and	no	N/A	visual	yes

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
	<color> [<icccolor>] | inherit		'feSpecularLighting' elements				
'marker'	see individual properties	see individual properties	'path', 'line', 'polyline' and 'polygon' elements	yes	N/A	visual	yes
'marker-end' 'marker-mid' 'marker-start'	none | inherit | <funciri>	none	'path', 'line', 'polyline' and 'polygon' elements	yes	N/A	visual	yes
'mask'	<funciri> | none | inherit	none	container elements and graphics elements	no	N/A	visual	yes
'opacity'	<opacity-value> | inherit	1	container elements and graphics elements	no	N/A	visual	yes
'overflow'	visible | hidden | scroll | auto | inherit	see prose	elements which establish a new viewport, 'pattern' elements and 'marker' elements	no	N/A	visual	yes
'pointer-events'	visiblePainted | visibleFill | visibleStroke | visible | painted | fill | stroke | all | none | inherit	visiblePainted	graphics elements	yes	N/A	visual	yes
'shape-rendering'	auto | optimizeSpeed | crispEdges | geometricPrecision | inherit	auto	shapes	yes	N/A	visual	yes
'stop-color'	currentColor | <color> [<icccolor>] | inherit	black	'stop' elements	no	N/A	visual	yes
'stop-opacity'	<opacity-value> | inherit	1	'stop' elements	no	N/A	visual	yes
'stroke'	<paint> (See Specifying paint)	none	shapes and text content elements	yes	N/A	visual	yes
'stroke-dasharray'	none | <dasharray> | inherit	none	shapes and text content elements	yes	N/A	visual	yes [1]

Name	Values	Initial value	Applies to	Inh.	Percentages	Media	Anim.
'stroke-dashoffset'	<percentage> | <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	N/A	visual	yes
'stroke-linejoin'	miter | round | bevel | inherit	miter	shapes and text content elements	yes	N/A	visual	yes
'stroke-miterlimit'	<miterlimit> | inherit	4	shapes and text content elements	yes	N/A	visual	yes
'stroke-opacity'	<opacity-value> | inherit	1	shapes and text content elements	yes	N/A	visual	yes
'stroke-width'	<percentage> | <length> | inherit	1	shapes and text content elements	yes	N/A	visual	yes
'text-anchor'	start | middle | end | inherit	start	text content elements	yes	N/A	visual	yes
'text-decoration'	none | [underline || overline || line-through || blink] | inherit	none	text content elements	no (see prose)	N/A	visual	yes
'text-rendering'	auto | optimizeSpeed | optimizeLegibility | geometricPrecision | inherit	auto	**'text'** elements	yes	N/A	visual	yes
'unicode-bidi'	normal | embed | bidi-override | inherit	normal	text content elements	no	N/A	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	N/A	visual	yes
'word-spacing'	normal | <length> | inherit	normal	text content elements	yes	N/A	visual	yes
'writing-mode'	lr-tb | rl-tb | tb-rl | lr | rl | tb | inherit	lr-tb	**'text'** elements	yes	N/A	visual	no

[1] The '**font**', '**font-size-adjust**' and '**stroke-dasharray**' properties are animatable but do not support additive animation.

Appendix O: Feature Strings

Contents

This appendix is normative.

O.1 Introduction

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 DOM Level 2 Core [DOM2] (see Feature strings for the **hasFeature** method call). In some cases the feature strings map directly to a set of attributes, properties or elements, in others they represent some functionality of the user agent (that it is a dynamic viewer for example). Note that the format and naming for feature strings changed from SVG 1.0 [SVG10] to SVG 1.1. The SVG 1.0 feature strings are listed below after the SVG 1.1 feature strings and User Agents should support all listed feature strings for compatibility reasons. However, the SVG 1.0 feature strings can be considered deprecated.

O.2 SVG 1.1 feature strings

Feature String:
http://www.w3.org/TR/SVG11/feature#SVG
User Agent Supports:
At least one of the following (all of which are described subsequently): "http://www.w3.org/TR/SVG11/feature#SVG-static", "http://www.w3.org/TR/SVG11/feature#SVG-animation", "http://www.w3.org/TR/SVG11/feature#SVG-dynamic" or "http://www.w3.org/TR/SVG11/feature#SVGDOM". (Because the feature string "http://www.w3.org/TR/SVG11/feature#SVG" can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)

Feature String:
http://www.w3.org/TR/SVG11/feature#SVGDOM
User Agent Supports:
At least one of the following (all of which are described subsequently): "http://www.w3.org/TR/SVG11/feature#SVGDOM-static", "http://www.w3.org/TR/SVG11/feature#SVGDOM-animation" or "http://www.w3.org/TR/SVG11/feature#SVGDOM-dynamic". (Because the feature string "http://www.w3.org/TR/SVG11/feature#SVGDOM" can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)

Feature String:
http://www.w3.org/TR/SVG11/feature#SVG-static
User Agent Supports:
The following features (described below)

- http://www.w3.org/TR/SVG11/feature#CoreAttribute
- http://www.w3.org/TR/SVG11/feature#Structure
- http://www.w3.org/TR/SVG11/feature#ContainerAttribute
- http://www.w3.org/TR/SVG11/feature#ConditionalProcessing
- http://www.w3.org/TR/SVG11/feature#Image
- http://www.w3.org/TR/SVG11/feature#Style
- http://www.w3.org/TR/SVG11/feature#ViewportAttribute
- http://www.w3.org/TR/SVG11/feature#Shape
- http://www.w3.org/TR/SVG11/feature#Text
- http://www.w3.org/TR/SVG11/feature#PaintAttribute
- http://www.w3.org/TR/SVG11/feature#OpacityAttribute
- http://www.w3.org/TR/SVG11/feature#GraphicsAttribute
- http://www.w3.org/TR/SVG11/feature#Marker
- http://www.w3.org/TR/SVG11/feature#ColorProfile
- http://www.w3.org/TR/SVG11/feature#Gradient
- http://www.w3.org/TR/SVG11/feature#Pattern
- http://www.w3.org/TR/SVG11/feature#Clip
- http://www.w3.org/TR/SVG11/feature#Mask
- http://www.w3.org/TR/SVG11/feature#Filter
- http://www.w3.org/TR/SVG11/feature#XlinkAttribute
- http://www.w3.org/TR/SVG11/feature#Font
- http://www.w3.org/TR/SVG11/feature#Extensibility

For SVG viewers, "http://www.w3.org/TR/SVG11/feature#SVG-static" indicates that the viewer can process and render successfully all of the language features corresponding to the feature strings listed above.

Feature String:
http://www.w3.org/TR/SVG11/feature#SVGDOM-static
User Agent Supports:
All of the DOM interfaces and methods that correspond to the language features for "http://www.w3.org/TR/SVG11/feature#SVG-static".

Feature String:
http://www.w3.org/TR/SVG11/feature#SVG-animation
User Agent Supports:
All of the language features from "http://www.w3.org/TR/SVG11/feature#SVG-static" plus the feature "http://www.w3.org/TR/SVG11/feature#Animation". For SVG viewers running on media capable of

rendering time-based material, such as displays, "http://www.w3.org/TR/SVG11/feature#SVG-animation" indicates that the viewer can process and render successfully all of the corresponding language features.

Feature String:
http://www.w3.org/TR/SVG11/feature#SVGDOM-animation
User Agent Supports:
All of the DOM interfaces and methods that correspond to the language features for "http://www.w3.org/TR/SVG11/feature#SVG-animation".

Feature String:
http://www.w3.org/TR/SVG11/feature#SVG-dynamic
User Agent Supports:
All of the language features from "http://www.w3.org/TR/SVG11/feature#SVG-animation" plus the following features:

- http://www.w3.org/TR/SVG11/feature#Hyperlinking
- http://www.w3.org/TR/SVG11/feature#Scripting
- http://www.w3.org/TR/SVG11/feature#View
- http://www.w3.org/TR/SVG11/feature#Cursor
- http://www.w3.org/TR/SVG11/feature#GraphicalEventsAttribute
- http://www.w3.org/TR/SVG11/feature#DocumentEventsAttribute
- http://www.w3.org/TR/SVG11/feature#AnimationEventsAttribute

For SVG viewers running on media capable of rendering time-based material, such as displays, "http://www.w3.org/TR/SVG11/feature#SVG-dynamic" indicates that the viewer can process and render successfully all of the corresponding language features.

Feature String:
http://www.w3.org/TR/SVG11/feature#SVGDOM-dynamic
User Agent Supports:
All of the DOM interfaces and methods that correspond to the language features for "http://www.w3.org/TR/SVG11/feature#SVG-dynamic".

Feature String:
http://www.w3.org/TR/SVG11/feature#CoreAttribute
User Agent Supports:
the 'id', 'xml:base', 'xml:lang' and 'xml:space' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#Structure
User Agent Supports:
'svg', 'g', 'defs', 'desc', 'title', 'metadata', 'symbol' and 'use' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicStructure
User Agent Supports:
'svg', 'g', 'defs', 'desc', 'title', 'metadata' and 'use' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#ContainerAttribute
User Agent Supports:
the '**enable-background**' property

Feature String:
http://www.w3.org/TR/SVG11/feature#ConditionalProcessing
User Agent Supports:
the 'switch' element, and the 'requiredFeatures', 'requiredExtensions' and 'systemLanguage' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#Image
User Agent Supports:
the 'image' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Style
User Agent Supports:
the 'style' element

Feature String:
http://www.w3.org/TR/SVG11/feature#ViewportAttribute
User Agent Supports:
the '**clip**' and '**overflow**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#Shape
User Agent Supports:
the 'rect', 'circle', 'line', 'polyline', 'polygon', 'ellipse' and 'path' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#Text
User Agent Supports:
the 'text', 'tspan', 'tref', 'textPath', 'altGlyph', 'altGlyphDef', 'altGlyphItem' and 'glyphRef' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicText

User Agent Supports:
the 'text' element

Feature String:
http://www.w3.org/TR/SVG11/feature#PaintAttribute
User Agent Supports:
the '**color**', '**fill**', '**fill-rule**', '**stroke**', '**stroke-dasharray**', '**stroke-dashoffset**', '**stroke-linecap**', '**stroke-linejoin**', '**stroke-miterlimit**', '**stroke-width**', '**color-interpolation**' and '**color-rendering**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicPaintAttribute
User Agent Supports:
the '**color**', '**fill**', '**fill-rule**', '**stroke**', '**stroke-dasharray**', '**stroke-dashoffset**', '**stroke-linecap**', '**stroke-linejoin**', '**stroke-miterlimit**', '**stroke-width**' and '**color-rendering**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#OpacityAttribute
User Agent Supports:
the '**opacity**', '**stroke-opacity**' and '**fill-opacity**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#GraphicsAttribute
User Agent Supports:
the '**display**', '**image-rendering**', '**pointer-events**', '**shape-rendering**', '**text-rendering**' and '**visibility**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicGraphicsAttribute
User Agent Supports:
the '**display**' and '**visibility**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#Marker
User Agent Supports:
the 'marker' element

Feature String:
http://www.w3.org/TR/SVG11/feature#ColorProfile
User Agent Supports:
the 'color-profile' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Gradient

User Agent Supports:
the 'linearGradient', 'radialGradient' and 'stop' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#Pattern
User Agent Supports:
the 'pattern' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Clip
User Agent Supports:
the 'clipPath' element and the '**clip-path**' and '**clip-rule**' properties

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicClip
User Agent Supports:
the 'clipPath' element and the '**clip-path**' property

Feature String:
http://www.w3.org/TR/SVG11/feature#Mask
User Agent Supports:
the 'mask' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Filter
User Agent Supports:
the 'filter', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feConvolveMatrix', 'feDiffuseLighting', 'feDisplacementMap', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', 'feMorphology', 'feOffset', 'feSpecularLighting', 'feTile', 'feDistantLight', 'fePointLight', 'feSpotLight', 'feFuncR', 'feFuncG', 'feFuncB' and 'feFuncA' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicFilter
User Agent Supports:
the 'filter', 'feBlend', 'feColorMatrix', 'feComponentTransfer', 'feComposite', 'feFlood', 'feGaussianBlur', 'feImage', 'feMerge', 'feMergeNode', 'feOffset', 'feTile', 'feFuncR', 'feFuncG', 'feFuncB' and 'feFuncA' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#DocumentEventsAttribute
User Agent Supports:
the 'onunload', 'onabort', 'onerror', 'onresize', 'onscroll' and 'onzoom' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#GraphicalEventsAttribute
User Agent Supports:
the 'onfocusin', 'onfocusout', 'onactivate', 'onclick', 'onmousedown', 'onmouseup', 'onmouseover', 'onmousemove', 'onmouseout' and 'onload' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#AnimationEventsAttribute
User Agent Supports:
the 'onbegin', 'onend', 'onrepeat' and 'onload' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#Cursor
User Agent Supports:
the 'cursor' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Hyperlinking
User Agent Supports:
the 'a' element

Feature String:
http://www.w3.org/TR/SVG11/feature#XlinkAttribute
User Agent Supports:
the 'xlink:type', 'xlink:href', 'xlink:role', 'xlink:arcrole', 'xlink:title', 'xlink:show' and 'xlink:actuate' attributes

Feature String:
http://www.w3.org/TR/SVG11/feature#ExternalResourcesRequired
User Agent Supports:
the 'externalResourcesRequired' attribute

Feature String:
http://www.w3.org/TR/SVG11/feature#View
User Agent Supports:
the 'view' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Script
User Agent Supports:
the 'script' element

Feature String:
http://www.w3.org/TR/SVG11/feature#Animation
User Agent Supports:
the 'animate', 'set', 'animateMotion', 'animateTransform', 'animateColor' and 'mpath' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#Font
User Agent Supports:
the 'font', 'font-face', 'glyph', 'missing-glyph', 'hkern', 'vkern', 'font-face-src', 'font-face-uri', 'font-face-format' and 'font-face-name' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#BasicFont
User Agent Supports:
the 'font', 'font-face', 'glyph', 'missing-glyph', 'hkern', 'font-face-src' and 'font-face-name' elements

Feature String:
http://www.w3.org/TR/SVG11/feature#Extensibility
User Agent Supports:
the 'foreignObject' element

O.3 SVG 1.0 feature strings

All SVG 1.0 [SVG10] feature strings referring to language capabilities begin with **"org.w3c.svg"**. All SVG 1.0 feature strings referring to SVG DOM capabilities begin with **"org.w3c.dom.svg"**.

- 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.

Appendix P: Media Type Registration for image/svg+xml

Contents

This appendix is normative.

P.1 Introduction

This appendix registers a new MIME media type, "image/svg+xml" in conformance with BCP 13 and W3CRegMedia.

P.2 Registration of media type image/svg+xml

Type name:
image
Subtype name:
svg+xml
Required parameters:
None.
Optional parameters:
charset
Same as application/xml media type, as specified in [RFC3023] or its successors.
Encoding considerations:
Same as for application/xml. See [RFC3023], section 3.2 or its successors.
Security considerations:
As with other XML types and as noted in [RFC3023] section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

Several SVG elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, SVG documents may reference external media such as images, audio, video, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply.

In addition, because of the extensibility features for SVG and of XML in general, it is possible that "image/svg+xml" may describe content that has security implications beyond those described here. However, if

the processor follows only the normative semantics of the published specification, this content will be outside the SVG namespace and shall be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

The published specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the SVG namespace and in other namespaces.

Because SVG is extensible, conformant "image/svg+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

SVG has a published Test Suite and associated implementation report showing which implementations passed which tests at the time of the report. This information is periodically updated as new tests are added or as implementations improve.

Published specification:

This media type registration is extracted from Appendix P of the SVG 1.1 specification.

Applications that use this media type:

SVG is used by Web browsers, often in conjunction with HTML; by mobile phones and digital cameras, as a format for interchange of graphical assets in desk top publishing, for industrial process visualization, display signage, and many other applications which require scalable static or interactive graphical capability.

Additional information:

Magic number(s):

File extension(s):

svg

Note that the extension 'svgz' is used as an alias for 'svg.gz' *[RFC1952]*, *i.e.* octet streams of type image/svg+xml, subsequently compressed with gzip.

Macintosh file type code(s):

"svg " (all lowercase, with a space character as the fourth letter).

Note that the Macintosh file type code 'svgz' (all lowercase) is used as an alias for GZIP *[RFC1952]* compressed "svg ", *i.e.* octet streams of type image/svg+xml, subsequently compressed with gzip.

Macintosh Universal Type Identifier code:

`org.w3c.svg` conforms to `public.image` and to `public.xml`

Windows Clipboard Name:

"SVG Image"

Fragment Identifiers

For documents labeled as application/svg+xml, the fragment identifier notation is either Shorthand Pointers (formerly called barenames) or the SVG-specific SVG Views syntax; both described in the fragment identifiers section of the SVG specification.

Person & email address to contact for further information:
Chris Lilley, Doug Schepers (member-svg-media-type@w3.org).

Intended usage:
COMMON

Restrictions on usage:
None

Author:
The SVG specification is a work product of the World Wide Web Consortium's SVG Working Group.

Change controller:
The W3C has change control over this specification.

Appendix Q: Changes

This appendix is informative, not normative.

No substantial changes have been made since the previous publication of SVG 1.1 Second Edition.