Scalable Vector Graphics (SVG) 1.0 Specification

W3C Working Draft *03 December 1999*

This version: http://www.w3.org/TR/1999/12/WD-SVG-19991203/
Latest version: http://www.w3.org/TR/SVG
Previous version: http://www.w3.org/1999/08/WD-SVG-19990812/

Editor: Jon Ferraiolo <jferraio@adobe.com>

Authors: John Bowler, Microsoft Corporation <johnbo@microsoft.com>
Milt Capsimalis, Autodesk Inc. <milt@autodesk.com>
Richard Cohn, Adobe Systems Incorporated <cohn@adobe.com>
David Dodds, Open Text <ddodds@opentext.com>
Andrew Donoho, IBM <awd@us.ibm.com>
David Duce, RAL (CCLRC) <d.a.duce@rl.ac.uk>
Jerry Evans, Sun Microsystems <jerry.evans@Eng.sun.com>
Jon Ferraiolo, Adobe Systems Incorporated <jferraio@adobe.com>
Scott Furman, Netscape Communications Corporation <fur@netscape.com>
Peter Graffagnino, Apple <pgraff@apple.com>
Rick Graham, BitFlash Graphics Inc. <rick@bitflash.com>
Lofton Henderson, OASIS, <lofton@qwestinternet.net>
Alan Hester, Xerox Corporation <Alan.Hester@usa.xerox.com>
Bob Hopgood, RAL (CCLRC) <frah@inf.rl.ac.uk>
Christophe Jolif, ILOG <jolif@ilog.fr>
Kelvin Lawrence, IBM <klawrenc@us.ibm.com>
Chris Lilley, W3C <chris@w3.org>
Philip Mansfield, Inso Corporation <philipm@schemasoft.com>
Kevin McCluskey, Netscape Communications Corporation <kmcclusk@netscape.com>
Tuan Nguyen, Microsoft Corporation <tuann@microsoft.com>
Troy Sandal, Visio Corporation <TroyS@visio.com>
Peter Santangeli, Macromedia <psantangeli@macromedia.com>
Haroon Sheikh, Corel Corporation <haroons@corel.ca>
Gavriel State, Corel Corporation <gavriels@COREL.CA>
Robert Stevahn, Hewlett-Packard Company <rstevahn@boi.hp.com>
Shenxue Zhou, Quark <szhou@quark.com>

Abstract

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

Status of this document

This document is a public review draft version of the SVG specification. This working draft attempts to address Last Call review comments from the previous public working draft (the "Last Call" draft of 12 August 1999) plus modifications resulting from continuing collaboration with other working groups and continuing work within the SVG working group.

An accumulative list of changes to the specification since the first public working draft of SVG (05 February 1999) is supplied in Appendix I: Change History.

One area of current activity where some changes are expected is the detailed definition of some of SVG's DOM interfaces.

This is a draft document and might be updated, replaced or obsoleted by other documents at any time. While we do not anticipate substantial changes, we still caution that further changes are possible. It is inappropriate to use this document as reference material or to cite it as other than "work in progress".

We explicitly invite comments on this specification. Please send them to svg-comments@w3.org.

The SVG working group has been using a staged approach. Initially, the working group developed a detailed set of SVG Requirements, which are listed in SVG Requirements. These requirements were posted for public review initially in October 1998. For the most part, the specification has been developed to provide the feature set listed in the requirements document. At some point, an updated version of SVG Requirements might be posted which contains detailed editorial comments about which requirements have been addressed in this draft (along with hyperlinks to the relevant sections of the specification) and notes about which requirements have not been addressed yet and why.

Public discussion of SVG features takes place on www-svg@w3.org, which is an automatically archived email list. Information on how to subscribe to public W3C email lists can be found at http://www.w3.org/Mail/Request.

The home page for the W3C graphics activity is http://www.w3.org/Graphics/Activity.

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

Available formats

The SVG specification is available in the following formats. (In future versions, the specification's vector drawings will be available in both SVG and raster image formats. For now, only raster image formats are available.)

HTML 4.0:

http://www.w3.org/TR/1999/12/WD-SVG-19991203/index.html

a zip file:

http://www.w3.org/TR/1999/12/WD-SVG-19991203.zip

and a PDF file:

http://www.w3.org/TR/1999/12/WD-SVG-19991203.pdf.

In case of a discrepancy between the various forms of the specification, the HTML is considered the definitive version.

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.

Quick Table of Contents

The following sections have not been written yet, but are expected to be be present in later versions of this specification:

- Appendix J: Element, attribute and property index
- Appendix K: Index

Full Table of Contents

The following sections have not been written yet, but are expected to be be present in later versions of this specification:

- Appendix J: Element, attribute and property index
- Appendix K: Index

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. 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 supplemental scripting language with access to SVG's 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 that higher-level information be made available somehow, either by making the original source document available, or making an alternative version available in an alternative format which conveys the higher-level information, or by using SVG's facilities to include the higher-level information within the SVG content. For suggested techniques in achieving greater accessibility, see Accessibility.

1.2 SVG MIME Type

The MIME type for SVG will be "`image/svg`". The W3C will register this MIME type around the time when SVG is approved as a W3C Recommendation.

1.3 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 is tracking and will conform with "XML Linking Language (XLink)" [XLINK].
- SVG's syntax for referencing element IDs is compatible with the ID referencing syntax in "XML Pointer Language (XPointer)" [XPTR].
- SVG content can be styled by either CSS (see "Cascading Style Sheets (CSS) level 2" specification [CSS2]) or XSL (see "XSL Transformations (XSLT) Version 1.0" [XSLT1]).
- SVG supports relevant properties and approaches common to CSS and XSL, plus selected semantics and features of CSS (see SVG's Use of Cascading Style Sheets).
- SVG can be used with "XSL Transformations (XSLT) Version 1.0" [XSLT1]. In particular, XSLT can style XML documents, with SVG output being a possible result of XSLT transformations.
- External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0" [ESS].
- SVG includes a complete Document Object Model (DOM) and conforms to the "Document Object Model (DOM) level 1" Recommendation [DOM1]. The SVG DOM has a high level of compatibility and consistency with the HTML DOM that is defined in the DOM level 1 specification. Additionally, the SVG DOM supports and incorporates many of the facilities described in "Document Object Model (DOM) level 2" [DOM2], including support for the CSS object model and event handling.
- SVG incorporates some features and approaches that are part of the "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification" [SMIL1], including the 'switch' element, the system-required attribute and the system-language attribute.
- SVG's animation features (see Animation) were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [SMIL1]. SVG's animation features incorporate and extend the general-purpose XML animation capabilities described in the "SMIL Animation" specification [SMILAnim].
- SVG has been designed to allow future versions of SMIL to use animated or static SVG content

as media components.

- SVG attempts to achieve maximum compatibility with both the "HTML 4.0 Specification" [HTML40] and the most recent working drafts of "XHTML(tm) 1.0: The Extensible HyperText Markup Language" [XHTML10]. Many of SVG's facilities are modeled directly after HTML, including its use of CSS [CSS2], its approach to event handling, its approach to its Document Object Model [DOM1].
- SVG is compatibility with W3C work on internationalization. References (W3C and otherwise) include: [UNICODE], [UNICODE21] and [CHARMOD]. Also, see Internationalization Support.
- SVG is compatible with W3C work on Web Accessibility [WAI]. Also, see Accessibility Support.

In environments which support [DOM2] for other XML grammars (e.g., XHTML [XHTML10]) 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.

1.4 Terminology

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

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

1.5 Definitions

basic shape

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

canvas

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

clipping path

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

container element

An element which can have graphics elements and other container elements as child elements.

Specifically: 'svg', 'g', 'defs' 'symbol', 'clipPath', 'mask', 'pattern', 'marker', 'a' and 'switch'.

current SVG document fragment

The XML document sub-tree which starts with the most immediate ancestor 'svg' element of a given SVG element

current transformation matrix (CTM)

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

fill

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

font

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

glyph

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

graphics element

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

graphics referencing element

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

local URI reference

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

mask

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

non-local URI reference

A Uniform Resource Identifier [URI] that includes an <absoluteURI> or <relativeURI> and thus (usually) represents a reference to a different document or an element/fragment within a different

document. See References and the 'defs' element.

paint

A paint represents a way of putting color values onto the canvas. A paint might consists 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.

shape

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

stroke

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

SVG canvas

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

SVG document fragment

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

SVG viewport

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

transformation

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

transformation matrix

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

URI Reference

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

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.

1.6 Error processing

There are various scenarios where an SVG document fragment is technically in error:

- 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. (Exception: 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. See 'path'

element implementation notes.) This approach will provide a visual clue to the user/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 perceptive 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 developer/user 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().

---

2 SVG Concepts

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 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 they can be referenced or included inside other SVG graphics, 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, maximise the advantages of HTTP cacheing and avoid the need for a centralised 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 rasterised; client side in the case of vector graphics, as opposed to already rasterised on the server. SVG gives control over the rasterisation process, for example to allow anti-aliased artwork without the ugly aliasing typical of low quality vector implementations. SVG also provided 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 standalone, or as an XML namespace with other grammars.

XML

XML, a W3C Recommendation for structured information exchange, has become extremely popular and is both widely and reliably implemented. By being written in XML, SVG builds on this strong foundation and gains many advantages such as a sound basis for internationalisation, 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 standalone, 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 [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.

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 geometric objects 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, 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 centralised 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 rasterisation 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 seachability 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.

---

3 Basic Data Types and Interfaces

Contents

3.1 Basic data types

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

- <angle>: An angle value is a <number> optionally followed immediately with an angle unit identifier. Angle unit identifiers are:
  - deg: degrees
  - grad: grads
  - rad: radians

  For properties defined in [CSS2], an angle unit identifier must be provided. For SVG-specific attributes and properties, the angle unit identifier is optional. If not provided, the angle value is assumed to be in degrees.
  The corresponding SVG DOM interface definition for <angle> is SVGAngle.
- <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' and 'stop-color', which also offer optional ICC-based color specifications.
  A <color> is either a keyword or a numerical RGB specification. The list of keyword color names is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. These 16 colors are defined in HTML 4.0 ([HTML40]). The format of an RGB value in hexadecimal notation is a '#' immediately followed by either three or six hexadecimal characters. The three-digit RGB notation (#rgb) is converted into six-digit form (#rrggbb) by replicating digits, not by adding zeros. For example, #fb0 expands to #ffbb00. This ensures that white (#ffffff) can be specified

with the short notation (#fff) and removes any dependencies on the color depth of the display. The format of an RGB value in the functional notation is 'rgb(' followed by a comma-separated list of three numerical values (either three integer values or three percentage values) followed by ')'. The integer value 255 corresponds to 100%, and to F or FF in the hexadecimal notation: rgb(255,255,255) = rgb(100%,100%,100%) = #FFF. Whitespace characters are allowed around the numerical values. All RGB colors are specified in the sRGB color space (see [SRGB]). Using sRGB provides an unambiguous and objectively measurable definition of of the color, which can be related to international standards (see [COLORIMETRY]).
The corresponding SVG DOM interface definitions for <color> are defined in [DOM2-CSS]; in particular, see the [DOM2-CSS-RGBCOLOR]. SVG's extension to color, including the ability to specify ICC-based colors, are represented in DOM interface SVGColor.

- <coordinate>: The format of a <coordinate> is a <number> optionally followed immediately by a CSS unit identifier.
  If the <coordinate> is expressed as a simple number without a CSS unit identifiers (e.g., 48), then the value represents a coordinate value in the current user coordinate system.
  If one of the CSS unit identifier is provided (e.g., 12mm), the <coordinate> represents the X-coordinate in the user coordinate system that is the given distance (measured in the viewport coordinate system) from the origin of the user coordinate system. (See Processing rules for CSS units and percentages.)
  If a percentage is provided (e.g., 10%), the <coordinate> represents the X-coordinate in the user coordinate system that is the given distance (measured as a percentage of the width of the viewport coordinate system) from the origin of the user coordinate system. (See Processing rules for CSS units and percentages.)
  Within the SVG DOM, a <coordinate> is represented as an SVGLength since both values have the same syntax (although the semantics are not identical).
- <frequency>: Frequency values are used with aural cascading style sheets (see [CSS2]). A frequency value is a <number> immediately followed by a frequency unit identifier. Frequency unit identifiers are:
  - Hz: Hertz
  - kHz: kilo Hertz

  Frequency values may not be negative.
  The corresponding SVG DOM interface definitions for <frequency> are defined in [DOM2-CSS].
- <integer>: An <integer> is specified as an optional sign character ('+' or '-', with '+' being the default) followed by one or more digits "0" to "9".
  Unless stated otherwise for a particular attribute or property, the range for a <integer> encompasses (at a minimum) -2147483648 to -2147483647.
  Within the SVG DOM, an <integer> is represented as an SVGInteger.
- <length>: A length is a distance measurement. The format of a <length> is a <number> optionally followed immediately by a CSS unit identifier. (Note that a <number> has different formulations depending on whether it is applied to a CSS property or an XML attribute.)
  If the <length> is expressed as a value without a CSS unit identifiers (e.g., 48), then the <length> represents a distance in the current user coordinate system.
  If one of the CSS unit identifier is provided (e.g., 12mm), then the <length> represents a width value in the viewport coordinate system. (See Processing rules for CSS units and percentages.)
  If a percentage is provided, (e.g., 10%), then the given percentage represents a percentage of the width of the viewport. (See Processing rules for CSS units and percentages.)
  Within the SVG DOM, a <length> is represented as an SVGLength.
- <list of xxx> (where xxx represents a value of some type): A list consists of a separated sequence of

values. The specification of lists is different for CSS property values than for XML attribute values.

- Lists in CSS property values are comma-separated, with optional white space before or after the comma.
- Lists within SVG's XML attributes are either comma-separated, with optional white space before or after the comma, or white space-separated.

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

- <number> (real number value): The specification of real number values is different for CSS property values than for XML attribute values.
  - CSS2 [CSS2] states that a property value which is a <number> is specified in decimal notation (i.e., a <decimal-number>), which consists of either an <integer>, or an optional sign character followed by zero or more digits followed by a dot (.) followed by zero or more digits with at least one digit required either before or after the dot. Thus, for conformance with CSS2, any property in SVG which accepts <number> values is specified in decimal notation only.
  - For SVG's XML attributes, to provide as much scalability in numeric values as possible, real number values can be provided either in decimal notation or in scientific notation (i.e., a <scientific-number>), which consists of a <decimal-number> immediately followed by the letter "e" or "E" immediately followed by an <integer>.

  Unless stated otherwise for a particular attribute or property, a <number> has the capacity for at least a single-precision floating point number (see [ICC32]) and has a range (at a minimum) of -3.4e+38F to +3.4e+38F.
  It is recommended that higher precision floating point storage and computation be performed on operations such as coordinate system transformations to provide the best possible precision and to prevent round-off errors.
  Conforming High-Quality SVG Viewers are required to use at least double-precision floating point (see [ICC32]) intermediate calculations on certain numerical operations.
  Within the SVG DOM, a <number> is represented as an SVGNumber.
- <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> is described in Specifying paint.
  Within the SVG DOM, <paint> is represented as an SVGPaint.
- <percentage>: The format of a percentage value is a <number> immediately followed by a '%'.
  Percentage values are always relative to another value, for example a length. Each attribute or property that allows percentages also defines the reference distance measurement to which the percentage refers.
  Within the SVG DOM, a <percentage> is represented as an SVGLength.
- <time>: A time value is a <number> immediately followed by a time unit identifier. Time unit identifiers are:
  - ms: milliseconds
  - s: seconds

  Time values are used in CSS properties and may not be negative.
  The corresponding SVG DOM interface definitions for <time> are defined in [DOM2-CSS].
- <transform-list> : The detailed description of the possible values for a <transform-list> are detailed in Modifying the User Coordinate System: the transform attribute.

Within the SVG DOM, <transform-list> is represented as an SVGTransformList.

- <uri> (Uniform Resource Identifiers [URI] references): A URI is the address of a resource on the Web. For the specification of URI references in SVG, see URI references.
  Within the SVG DOM, <uri> is represented as a DOMString.

3.2 Basic DOM interfaces

3.2.1 Overview

The section describes the basic DOM interfaces for SVG's Document Object Model that are common to multiple parts of the SVG DOM. Many of these interfaces correspond directly with SVG's basic data types.

3.2.2 Interface SVGAngle

This interface corresponds to the <angle> basic data type.

```
interface SVGAngle  {
  // Unit Types
  const unsigned short kSVG_ANGLETYPE_UNKNOWN     = 0;  // invalid, must be retrieved as a string
  const unsigned short kSVG_ANGLETYPE_UNSPECIFIED = 1 ; // no value provided, default kicks in
  const unsigned short kSVG_ANGLETYPE_DEG         = 2; // Degrees or unitless (means degrees)
  const unsigned short kSVG_ANGLETYPE_RAD         = 3;
  const unsigned short kSVG_ANGLETYPE_GRAD        = 4;
  readonly attribute  unsigned short unittype;

  // Setting any of these causes the other values to
  // be updated automatically.
  // If kSVG_ANGLETYPE_UNSPECIFIED, these reflect the default value.
  attribute float angle;   // in degrees
  attribute float angleInSpecifiedUnits;
  attribute DOMString angleAsString;

  // Utility methods
  void newAngleSpecifiedUnits(in unsigned short unittype, in float angleInSpecifiedUnits);
  void convertToSpecifiedUnits(in unsigned short unittype);

  // If this attribute or style property currently is being animated,
  // animatedValue reflects the current animated value of the
  // attribute or style property.
  // Otherwise, when not animated, it will equal 'value'
  readonly attribute float animatedValue;  // in user units
};
```

3.2.3 Interface SVGInteger

This interface corresponds to the <integer> basic data type.

```
interface SVGInteger  {
  attribute long value;

  // If this attribute or style property currently is being animated,
  // animatedValue reflects the current animated value of the
  // attribute or style property.
  // Otherwise, when not animated, it will equal 'value'
  readonly attribute float animatedValue;
};
```

3.2.4 Interface SVGLength

This interface corresponds to the <length> basic data type.

```
interface SVGLength  {
  // Unit Types
  const unsigned short kSVG_LENGTHTYPE_UNKNOWN     = 0; // invalid, must be retrieved as a string
  const unsigned short kSVG_LENGTHTYPE_UNSPECIFIED = 1; // no value provided, defaults kick in
  const unsigned short kSVG_LENGTHTYPE_NUMBER      = 2; // Unitless, meaning user units
  const unsigned short kSVG_LENGTHTYPE_PERCENTAGE  = 3;
  const unsigned short kSVG_LENGTHTYPE_EMS         = 4;
  const unsigned short kSVG_LENGTHTYPE_EXS         = 5;
  const unsigned short kSVG_LENGTHTYPE_PX          = 6;
  const unsigned short kSVG_LENGTHTYPE_CM          = 7;
  const unsigned short kSVG_LENGTHTYPE_MM          = 8;
  const unsigned short kSVG_LENGTHTYPE_IN          = 9;
  const unsigned short kSVG_LENGTHTYPE_PT          = 10;
  const unsigned short kSVG_LENGTHTYPE_PC          = 11;
  readonly attribute  unsigned short unittype;

  // Setting any of these causes the other values to
  // be updated automatically.
  // If kSVG_LENGTHTYPE_UNSPECIFIED, these reflect the default value.
  attribute float value;  // in user units
  attribute float valueInSpecifiedUnits;
  attribute DOMString valueAsString;

  // Utility methods
  void newValueSpecifiedUnits(in unsigned short unittype, in float valueInSpecifiedUnits);
  void convertToSpecifiedUnits(in unsigned short unittype);

  // If this attribute or style property currently is being animated,
  // animatedValue reflects the current animated value of the
  // attribute or style property.
  // Otherwise, when not animated, it will equal 'value'
  readonly attribute float animatedValue;  // in user units
};
```

3.2.5 Interface SVGLengthList

This interface corresponds to values which represent a list of <length> values.

```
interface SVGLengthList  {
  SVGLength     createSVGLength();     // Returns unattached length of 0 user units

  readonly attribute  unsigned long number_of_lengths;
  SVGLength      getSVGLength(in unsigned long index);

  // Replace all existing entries with a single entry.
  void           initialize(in SVGLength newSVGLength)
                                       raises(DOMException);
  void           clear(); // Clear all entries, giving an empty list
  SVGLength      insertBefore(in SVGLength newSVGLength,
                                       in unsigned long index)
                                       raises(DOMException);
  SVGLength      replace(in SVGLength newSVGLength,
                                  in unsigned long index)
                                  raises(DOMException);
  SVGLength      remove(in unsigned long index)
                                 raises(DOMException);
  SVGLength      append(in SVGLength newSVGLength)
                                 raises(DOMException);
};
```

Used to values that can be expressed as an array of SVGLengths.

3.2.6 Interface SVGNumber

This interface corresponds to the <number> basic data type.

```
interface SVGNumber  {
  attribute float value;  // in user units

  // If this attribute or style property currently is being animated,
  // animatedValue reflects the current animated value of the
  // attribute or style property.
  // Otherwise, when not animated, it will equal 'value'
  readonly attribute float animatedValue;  // in user units
};
```

3.2.7 Interface SVGRect

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

```
interface SVGRect  {
  attribute SVGNumber x;
  attribute SVGNumber y;
  attribute SVGNumber width;
  attribute SVGNumber height;

};
```

---

4 SVG Rendering Model

Contents

4.1 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.)

4.2 The painters model

SVG uses a "painters model" of rendering. Paint is applied in successive operations to the output device such that each operation paints over some area of the output device. When the area overlaps a previously painted area the new paint partially or completely obscures the old. When the paint is not completely

opaque the result on the output device is defined by the (mathematical) rules for compositing described under Simple Alpha Blending.

4.3 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.

4.4 Grouping

Grouping elements such as the 'g' have the effect of producing a temporary separate canvas onto which child elements are painted. Upon the completion of the group, the effect is as if the group's canvas is painted onto the ancestors canvas using the standard rendering rules for individual graphic objects.

4.5 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.)

4.5.1 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 above the shape object which is using the given marker symbol. The graphical contents of a marker symbol are rendered using the same methods are 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

4.5.2 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.

4.6 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, of a size determined by the rules given in Filter Effects then filtered by the processes defined in Filter Effects.

4.7 Clipping, masking and object opacity

SVG allows any painting operation to be limited to a sub-region 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. "Within" is defined by the same rules used to determine the interior of a path for painting.

Masking uses the alpha channel or color information in a referenced SVG element to restrict the painting operation. In this case the opacity information within the alpha channel is used to define the region to which paint can be applied - any region of the output device that, after resampling the alpha channel appropriately, has a zero opacity must not be affected by the paint operation. All other regions composite the paint from the paint operation onto the the output device using the algorithms described in Clipping, Masking and Compositing.

A supplemental masking operation may also be specified by applying a "global" opacity to a set of rendering operations. In this case the mask defines an infinite alpha channel with a single opacity. (See 'opacity' property.)

In all cases the SVG implementation must behave as though all painting and filtering performed within the clip or masks is done first to an intermediate (imaginary) canvas then filtered through the clip area or masks. 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.

4.8 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 Styling

Contents

5.1 Styling SVG content

SVG content can be styled by either CSS (see "Cascading Style Sheets (CSS) level 2" specification [CSS2]) or XSL (see "XSL Transformations (XSLT) Version 1.0" [XSLT1]).

SVG content using CSS or XSL for styling can reference external style sheets (see Referencing external style sheets) or embed style sheets within an SVG 'style' element or both.

CSS style declarations can also be specified within style attributes on particular elements. For many applications, element-specific styling is convenient and advantageous, but in situations where multiple elements have common styling, it is usually better to express styling through through the 'style' element or, even better, through external style sheets which may be shared by several related SVG graphics.

Styling the same document using both CSS and XSL style sheets is not recommended at this time as the processing model for this is not well-defined.

5.2 Referencing external style sheets

External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0" [ESS].

5.3 SVG's use of Cascading Style Sheets

SVG supports various relevant properties and approaches common to CSS and XSL, plus selected semantics and features of CSS (see the "Cascading Style Sheets (CSS) level 2" Recommendation [CSS2].

SVG uses styling properties to describe many of its document parameters. In particular, SVG uses styling properties for the following:

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

The following properties from CSS2 [CSS2] are used by SVG:

- Font properties:
  - 'font-family'
  - 'font-style'
  - 'font-variant'
  - 'font-weight'
  - 'font-stretch'
  - 'font-size'
  - 'font-size-adjust'
  - 'font'
- Text properties:
  - 'text-decoration'
  - 'letter-spacing'
  - 'word-spacing'
  - 'direction'
  - 'unicode-bidi'
- Other properties for visual media:
  - 'visibility'

- 'display'
- 'overflow' (Only applicable to elements which establish a new viewport)
- 'clip' (Only applicable to outermost 'svg')
- 'color' is used to provide a potential indirect value (currentColor) for the 'fill' and 'stroke' properties. (The SVG properties which support color allow a color specification which is extended from CSS2 to accommodate color definitions in arbitrary color spaces defined by the 'color-space' property.)
- 'cursor'

The following facilities from CSS2 are supported in SVG:

- CSS2 syntax rules, including allowable data types
- Style sheet declarations, including selectors.
- SVG supports both external CSS style sheets [ESS] and the inclusion of style rules within SVG content using 'style' elements and style attributes attached to specific SVG elements.
- CSS2 rules for assigning property values, cascading and inheritance
- @font-face, @media, @import and @charset rules within style sheets
- CSS2's dynamic pseudo-classes (i.e., :hover, :active and :focus) [CSS2-DYNPSEUDO]. (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 [CSS2-AURAL] can be applied as defined in [CSS2]. (See Aural style sheets.)

One variation SVG offers to CSS is that unit-less lengths and sizes are allowed in SVG properties. (Refer to the discussion of Units.)

Additionally, SVG defines a new @color-profile at-rule [CSS2-ATRULES] for defining color profiles to use within SVG content.

5.4 The 'style' element

```
<!ELEMENT style (#PCDATA)* >
<!ATTLIST style type CDATA "text/css" >
```

Attribute definitions:

type = *content-type*

This attribute specifies the style sheet language of the element's contents and overrides the default style sheet language. The style sheet language is specified as a content type (e.g., "text/css"). Authors must supply a value for this attribute; there is no default value for this attribute.
Animatable: no.

The 'style' element allows authors to put style sheet rules embedded within SVG content. 'style' elements are only allowed as children of 'defs' elements.

The syntax of style data depends on the style sheet language.

Some style sheet implementations might allow a wider variety of rules in the 'style' element than in the style attribute that is available to container elements and graphics elements. For example, with CSS [CSS2], rules can be declared within a 'style' element that cannot be declared within a style attribute.

The following is an example of defining and using a text style using a CSS internal style sheet:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs>
    <style><![CDATA[
      .TitleText { font-size: 16; font-family: Helvetica } ]]>
    </style>
  </defs>
  <text class="TitleText">Here is my title</text>
</svg>
```

Download this example

Note how the CSS style sheet is placed within a `CDATA` construct (i.e., `<![CDATA[ ... ]]>`), which is necessary since CSS style sheets are not expressed in XML.

An XSL style sheet ([XSLT1]) can also be embedded within a 'style' element, in which case it is not necessary to enclose the style sheet within a `CDATA` construct, since XSL style sheets are expressed in XML.

5.5 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 attributes 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>
```

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 }
```

5.6 The style attribute

Attribute definitions:

style = *style*

> This attribute specifies style information for the current element. The style attribute specifies style information for a single element. The style sheet language of inline style rules is given by the default style sheet language. The syntax of style data depends on the style sheet language. Animatable: yes.

The default style sheet language for the style attribute is "text/css" unless any HTTP headers specify the "Content-Style-Type", in which case the last one in the character stream determines the default style sheet language.

This example sets fill and font size information for the text in a specific 'text' element:

```
<text style="font-size: 12pt; fill: fuchsia">Isn't styling wonderful?</text>
```

In CSS, property declarations have the form "name : value" and are separated by a semi-colon.

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.

5.7 Cascading and inheritance of properties

SVG conforms fully to the cascading style rules of CSS (i.e., the rules by which the SVG user agent decides which property setting applies to a given element). See the CSS2 specification for a discussion of these rules.

The definition of each CSS property indicates whether the property can inherit the value of its parent.

5.8 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 stylesheet PI) 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 HTML/XHTML document, and one for the SVG document. Style sheets defined anywhere within the HTML/XHTML document (in style elements or style attributes, or in external style sheets linked with the stylesheet PI) apply across the entire HTML/XHTML document. Since inheritance is down a parse tree, these styles do not affect the SVG document. Style sheets defined anywhere within the SVG document (in style elements or style attributes, or in external style sheets linked with the stylesheet PI) apply across the entire SVG document. These styles do not affect the containing HTML/XHTML document. To get the same styling across both HTML/XHTML document and SVG document, link them both to the same stylesheet.

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 stylesheet PI) 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.

5.9 The 'display' property

'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:	all elements
Inherited:	no
Percentages:	N/A
Media:	all
Animatable:	yes

A value other than display: none indicates that the given element shall be rendered by the SVG user agent.

5.10 Default style sheet for SVG

The user agent's default style sheet for elements in the SVG namespace for visual media [CSS2-VISUAL] must include the following entries:

```
svg, symbol, marker, pattern, view { overflow: hidden }
```

Refer the description of SVG's use of the 'overflow' property for more information.

Also, refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2].

5.11 DOM interfaces

5.11.1 Interface SVGStyleElement

The SVGStyleElement interface corresponds to the 'style' element.

```
interface SVGStyleElement : SVGElement {
  attribute DOMString type;
};
```

---

6 SVG Document Structure

Contents

6.1 Defining an SVG document fragment: the 'svg' element

6.1.1 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 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://someplace.org"
       xmlns:svg="http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
   <!-- parent stuff here -->
   <svg:svg width="5cm" height="8cm">
      <svg:ellipse rx="200" ry="130" />
   </svg:svg>
   <!-- ... -->
</parent>
```

Download this example

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Four separate rectangles
  </desc>
    <rect width="20" height="60"/>
    <rect width="30" height="70"/>
    <rect width="40" height="80"/>
    <rect width="50" height="90"/>
</svg>
```

Download this example

'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 and alter the meaning of CSS unit specifiers. See Establishing a new viewport and Redefining the meaning of CSS unit specifiers.

6.1.2 The 'svg' element

```
<!ENTITY % svgExt "" >
<!ELEMENT svg (%descTitleDefs;,metadata?,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%svgExt;)*) >
<!ATTLIST svg
  xmlns CDATA #FIXED 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  %documentEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  refX CDATA #IMPLIED
```

```
refY CDATA #IMPLIED
viewBox CDATA #IMPLIED
preserveAspectRatio CDATA 'xMidYMid meet'
enableZoomAndPanControls (true | false) "true"
contentScriptType CDATA #IMPLIED >
```

Attribute definitions:

xmlns [:*prefix*] = "resource-name"

Standard XML attribute for identifying an XML namespace. Refer to the "Namespaces in XML" Recommendation [XML-NS].
Animatable: no.

id = "name"

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

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.

x = "<coordinate>"

(Has no meaning or effect on outermost 'svg' elements.) The *x-coordinate* of one corner of the rectangular region into which an embedded 'svg' element is placed. The default x-coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

y = "<coordinate>"

(Has no meaning or effect on outermost 'svg' elements.) The *y-coordinate* of one corner of the rectangular region into which an embedded 'svg' element is placed. The default y-coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

width = "<length>"

For outermost 'svg' elements, the intrinsic width of the SVG document fragment, with *length* being any valid expression for a length in SVG. For embedded 'svg' elements, the width of the rectangular region into which the 'svg' element is placed.
Animatable: yes.

height = "<length>"

For outermost 'svg' elements, the intrinsic height of the SVG document fragment, with *length* being any valid expression for a length in SVG. For embedded 'svg' elements, the height of the rectangular region into which the 'svg' element is placed.
Animatable: yes.

refX = "<coordinate>"

When referenced in the context that requires a reference point (e.g., a motion path animation), the x-coordinate of the reference point.
Animatable: yes.

refY = "<coordinate>"

When referenced in the context that requires a reference point (e.g., a motion path animation), the y-coordinate of the reference point.
Animatable: yes.

Attributes defined elsewhere:

class, style, %graphicsElementEvents;, %documentEvents;, system-required, system-language, viewBox, preserveAspectRatio, enableZoomAndPanControls, contentScriptType.

6.2 Grouping and Naming Collections of Drawing Elements: the 'g' element

6.2.1 Overview

The 'g' element is the element for grouping and naming collections of drawing elements. If several drawing elements share similar attributes, they can be collected together using a 'g' element. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Two groups, each of two rectangles
  </desc>
  <g style="fill:red">
    <rect x="100" y="100" width="100" height="100" />
    <rect x="300" y="100" width="100" height="100" />
  </g>
  <g style="fill:blue">
    <rect x="100" y="300" width="100" height="100" />
    <rect x="300" y="300" width="100" height="100" />
  </g>
</svg>
```

Download this example

A group of drawing elements, as well as individual objects, can be given a name. Named groups are needed for several purposes such as animation and re-usable objects. The following example organizes the drawing elements into two groups and assigns a name to each group:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Two named groups
  </desc>
  <g id="OBJECT1">
    <rect x="100" y="100" width="100" height="100" />
  </g>
  <g id="OBJECT2">
    <circle cx="150" cy="300" r="25" />
  </g>
</svg>
```

Download this example

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

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Groups can nest
  </desc>
  <g>
     <g>
       <g>
       </g>
     </g>
   </g>
</svg>
```

Download this example

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

6.2.2 The 'g' element

```
<!ENTITY % gExt "" >
<!ELEMENT g (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|
                   animate|set|animateMotion|animateColor|animateTransform
                   %ceExt;%gExt;)*) >
<!ATTLIST g
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED >
```

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

6.3 References and the 'defs' element

6.3.1 Overview

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

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

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

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

In SVG, the following facilities allow URI references:

- the 'a' element
- the 'altGlyph' element
- the 'animate' element
- the 'animateColor' element
- the 'animateMotion' element
- the 'animateTransform' element
- the 'clip-path' property
- the 'cursor' element and 'cursor' property
- the 'feImage' element
- the 'fill' property
- the 'filter' element and 'filter' property
- the 'image' element
- the 'linearGradient' element
- the 'marker','marker-start','marker-mid' and 'marker-end properties
- the 'mask' property
- the 'pattern' element
- the 'radialGradient' element
- the 'script' element
- the 'src' descriptor on an @color-profile definition
- the 'stroke' property
- the 'textpath' element
- the 'tref' element

- the 'set' element
- the 'use' element

URI references are defined in either of the following forms:

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

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

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

SVG supports two types of URI references:

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

The following rules apply to the processing of URI references:

- All URI references to SVG elements (local or non-local) must be to elements which are immediate children of a 'defs' element. References to elements which are not immediate children of a 'defs' element shall be treated as invalid references. (This requirement allows SVG user agents to potentially perform optimizations because only those elements defined in a 'defs' element need to be retained as the remainder of the document is processed.)
- All local URI references must be to elements defined earlier in the document. Local URI references to elements later in the document (i.e., forward references) shall treated as invalid references.
- URI references to elements that don't exist shall be treated as invalid references.
- URI references to elements which are inappropriate targets for the given reference shall be treated as invalid references. For example, the 'clip-path' property can only refer to <clipPath> elements. The property setting clip-path:url(#MyElement) is an invalid reference if the referenced element is not a <clipPath>.

Unless a given attribute or property has defined fallback behavior in case a reference cannot be resolved, invalid references are treated as errors (see Error Processing). For example, if there is no element with ID "BogusReference" in the current document, then **fill="url(#BogusReference)"** would represent an invalid reference and would be an error.

6.3.2 URI reference attributes

```
<!ENTITY % xlinkRefAttrs
  xmlns:xlink CDATA #FIXED "http://www.w3.org/XML/XLink/0.9"
  xlink:type (simple|extended|locator|arc) #FIXED "simple"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|embed|replace) #FIXED 'embed'
  xlink:actuate (user|auto) #FIXED 'auto'
>

  xlink:href CDATA #REQUIRED
```

xmlns [:*prefix*] = "resource-name"

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

xlink:type = 'simple'

Identifies the type of XLink being used. For hyperlinks in SVG, only simple links are available. Refer to the "XML Linking Language (XLink)" [XLink].
Animatable: no.

xlink:role = '<string>'

A generic string used to describe the function of the link's content. Refer to the "XML Linking Language (XLink)" [XLink].
Animatable: no.

xlink:title = '<string>'

Human-readable text describing the link. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:show = 'embed'

Indicates that upon activation of the link the contents of the referenced object are incorporated appropriately into the current SVG document fragment. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:actuate = 'auto'

Indicates that the contents of the referenced object are incorporated into the current document automatically (i.e., without user action). Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:href = "<uri>"

The location of the referenced object, expressed as a URI reference. Each element in SVG which has an xlink:href attribute will describe the particular usage rules relevant to that element. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: yes.

6.3.3 The 'defs' element

The 'defs' element is used to identify those objects which will be referenced by other objects later in the document. It is a requirement that all referenced objects be defined within a 'defs' element. (See References and the 'defs' element.)

The child elements within a 'defs' element are not drawn.

```
<!ENTITY % defsExt "" >
<!ELEMENT defs (script|style|symbol|marker|clipPath|mask|
                linearGradient|radialGradient|pattern|filter|cursor|font|
                animate|set|animateMotion|animateColor|animateTransform|
                path|text|rect|circle|ellipse|line|polyline|polygon|
                use|image|svg|g|view|switch|altGlyphDef
                %ceExt;%defsExt;)* >
<!ATTLIST defs
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED >
```

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style.

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

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN" "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Local URI references within ancestor's 'defs' element.</desc>
  <defs>
    <linearGradient id="Gradient01">
      <stop offset="30%" style="color:#39F"/>
    </linearGradient>
  </defs>
  <g>
    <g>
      <rect x="0%" y="0%" width="100%" height="100%"
            style="fill:url(#Gradient01)" />
    </g>
  </g>
</svg>
```

Download this example

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.

6.4 The 'desc' and 'title' elements

Each container element or graphics element in an SVG drawing can supply a 'desc' and/or a 'title' description string where the description is text-only. For visual presentation, the SVG default stylesheet has 'display:none' for 'desc' and 'title' elements. Thus, they are not drawn as text in the graphic. User agents may, 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 readilly achieved by using a different (perhaps user) stylesheet. 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.

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/SVG-19991203.dtd">
<svg 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>
```

Download this example

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

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns="http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
   <desc xmlns:mydoc="http://foo.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>
```

Download this example

6.5 The 'symbol' element

The 'symbol' element is used to define graphical objects which are meant for any of the following uses:

- A template object which will be used (i.e., instantiated) multiple times within a given document
- A member of a standard drawing symbol library that is referenced by a variety of different SVG content
- The definition of a graphic to use as a custom glyph within a 'text' element (e.g., generalize "text-on-a-path" to "SVG-on-a-path")
- Definition of a sprite for an animation

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

```
<!ENTITY % symbolExt "" >
<!ELEMENT symbol (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%symbolExt;)*) >
<!ATTLIST symbol
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet' >
```

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, refX, refY, viewBox, preserveAspectRatio.

6.6 The 'use' element

Any 'svg', 'symbol', 'g', or graphics element that is a child of a 'defs' element and has been assigned an ID is potentially a template object that can be re-used (i.e., "instanced") anywhere 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.

The 'use' element can reference either:

- an element within the same SVG document whose immediate ancestor is a 'defs' element
- an element within a different SVG document whose immediate ancestor is a 'defs' element

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

In the example below, the first 'g' element has inline content. After this comes a 'use' element whose href value indicates which graphics element is included/rendered at that point in the document. Finally, the second 'g' element has both inline and referenced content. In this case, the referenced content will draw first, followed by the inline content.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs>
    <symbol id="TemplateObject01">
      <!-- symbol definition here -->
    </symbol>
  </defs>

  <desc>Examples of inline and referenced content
  </desc>

  <!-- <g> with inline content -->
  <g>
    <!-- Inline content goes here -->
  </g>

  <!-- referenced content -->
  <use xlink:href="#TemplateObject01" />

  <!-- <g> with both referenced and inline content -->
  <g>
    <use xlink:href="#TemplateObject01" />
    <!-- Inline content goes here -->
  </g>
</svg>
```

Download this example

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.

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 its original parents.

The following example illustrates property inheritance with the 'use' element:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs style="stroke:green">
    <!-- Note that parent's stroke:green will have no effect below -->
    <circle id="TemplateObject02" cx="50" cy="50" r="30" style="fill:red" />
  </defs>

  <desc>Examples of <use> property inheritance
  </desc>

  <g style="fill:yellow;stroke:blue" >
    <!-- Draws a circle with fill:red and stroke:blue. -->
    <!-- Note that the referenced element specifies fill:red,
         which takes precedence over the inherited fill:yellow. -->
    <use xlink:href="#TemplateObject02" />
  </g>
</svg>
```

Download this example

In the example above, property inheritance for 'use' element shown above is as if the 'use' element were replaced by a container object whose contents are the referenced element:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs style="stroke:green">
    <!-- Note that parent's stroke:green will have no effect below -->
    <circle id="TemplateObject02" cx="50" cy="50" r="30" style="fill:red" />
  </defs>

  <desc>Examples of <use> property inheritance
  </desc>

  <g style="fill:yellow;stroke:blue" >
    <!-- Draws a circle with fill:red and stroke:blue. -->
    <!-- Note that the referenced element specifies fill:red,
         which takes precedence over the inherited fill:yellow. -->
    <g>
      <circle cx="50" cy="50" r="30" style="fill:red" />
    </g>
  </g>
</svg>
```

Download this example

```
<!ENTITY % useExt "" >
<!ELEMENT use (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%useExt;)*) >
<!ATTLIST use
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

x = "<coordinate>"

The x coordinate of one corner of the rectangular region into which the referenced element is placed. The default x coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

y = "<coordinate>"

The y coordinate of one corner of the rectangular region into which the referenced element is placed. The default y coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

width = "<length>"

The width of the rectangular region into which the referenced element is placed.
Animatable: yes.

height = "<length>"

The height of the rectangular region into which the referenced element is placed.
Animatable: yes.

xlink:href = "<uri>"

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

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language, %xlinkAttrs;.

6.7 The 'image' element

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

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

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

```
<!ENTITY % imageExt "" >
<!ELEMENT image (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                  %geExt;%imageExt;)*) >
<!ATTLIST image
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

x = "<coordinate>"

The x coordinate of one corner of the rectangular region into which the referenced document is placed. The default x coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

y = "<coordinate>"

The y coordinate of one corner of the rectangular region into which the referenced document is placed. The default y coordinate is zero. See Coordinate Systems, Transformations and Units.
Animatable: yes.

width = "<length>"

The width of the rectangular region into which the referenced document is placed.
Animatable: yes.

height = "<length>"

The height of the rectangular region into which the referenced document is placed.
Animatable: yes.

xlink:href = "<uri>"

A URI reference.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language, %xlinkAttrs;.

A valid example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <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>
```

Download this example

A well-formed example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns='http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <desc>This links to an external image
  </desc>
  <image x="200" y="200" width"100px" height="100px"
         xlink:type="simple" xlink:show="embed" xlink:actuate="auto"
         xlink:href="myimage.png">
    <title>My image</title>
  </image>
</svg>
```

Download this example

6.8 Conditional processing

6.8.1 Conditional processing overview

SVG contains a 'switch' element along with attributes system-required and system-language to provide an ability to specify alternate viewing depending on the capabilities of a given user agent or the user's language. These features operate with the same semantics as the corresponding features within the SMIL 1.0 Recommendation [SMIL1].

Attributes system-required and system-language act as tests and return either true or false results. The 'switch' renders the first of its children for which both attributes test true.

6.8.2 The 'switch' element

The 'switch' element evaluates the system-required and system-language attributes on its direct child elements in order, and then processes and renders the first child for which these two 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.

```
<!ENTITY % switchExt "" >
<!ELEMENT switch (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|foreignObject|
                   animate|set|animateMotion|animateColor|animateTransform
                   %ceExt;%switchExt;)*) >
<!ATTLIST switch
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED >
```

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

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

6.8.3 The system-required attribute

Definition of system-required:

system-required = *list-of-features*

The value is a comma-separated list of feature strings. Determines whether all of the named *features* are supported by the user agent. If one of the given features is not supported, then the current element and its children are processed; otherwise, the current element and its children are skipped and thus will not be rendered and cannot be referenced by another element.
Animatable: no.

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

- The feature string "SVG" indicates that the user agent supports at least one of the following (all of which are described subsequently): "SVGLang", "SVGDynamic", "SVGStatic" or "SVGDOM". (Because this feature string can be ambiguous in some circumstances, it is recommended that more specific feature strings be used.)
- The feature string "SVGLang" indicates that the user agent can parse and process all of the language features defined in this specification. This value indicates that there is no language feature defined in this specification which will cause the user agent to fail in its processing.
- The feature string "SVGStatic" indicates the availability of all of the language capabililities defined in:
  - Basic Data Types and Interfaces
  - SVG 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 system-required attribute
    - The system-language attribute

  For SVG viewers, "SVGStatic" indicates that the viewer can process and render successfully all of the language features listed above.
- The feature string "SVGDOMStatic" indicates the availability of all of the DOM interfaces and methods that correspond to the language features for "SVGStatic".
- The feature string "SVGAnimation" includes all of the language capabilities defined for "SVGStatic" plus the availability of all of the language capabililities and DOM interfaces defined in Animation. For SVG viewers running on media capable of rendering time-based material, such as displays, "SVGAnimation" indicates that the viewer can process and render successfully all of the corresponding language features.
- The feature string "SVGDOMAnimation" corresponds to the availability of DOM interfaces and methods that correspond to the language features for "SVGAnimation".
- The feature string "SVGDynamic" includes all of the language capabilities defined for "SVGAnimation" plus the availability of all of the language capabililities and DOM interfaces 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, "SVGDynamic" indicates that the viewer can process and render successfully all of the corresponding language features.
- The feature string "SVGDOMDynamic" corresponds to the availability of DOM interfaces and methods that correspond to the language features for "SVGDynamic".
- The feature string "SVGAll" corresponds to the availability of all of the language capabilities defined in this specification.
- The feature string "SVGDOMAll" corresponds to the availability of all of the DOM interfaces defined in this specification.

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

system-required is often used in conjunction with the 'switch' element. If the system-required is used in other situations, then it represented a simple switch on the given element whether to render the element or not.

6.8.4 The system-language attribute

The attribute value is a comma-separated list of language names as defined in [RFC1766].

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.

Further description of the system-language attribute can be found at [SMIL10-SYSLANG].

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 system-required, the attribute returns "false".

6.9 DOM interfaces

6.9.1 Overview

This section describes the SVG-specific DOM interfaces that correspond to the topics described in this chapter.

6.9.2 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 [XHTML10], 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 standalone SVG file (i.e., a file with MIME type "image/svg"). In this case, the SVGDocument object will be the 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" 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 Document Object Model (HTML) Level 1 chapter of the [DOM1] specification.

IDL Definition

```
interface SVGDocument : Document {
         attribute  DOMString            title;
readonly attribute  DOMString            referrer;
readonly attribute  DOMString            domain;
readonly attribute  DOMString            URL;

       attribute SVGSVGElement         rootElement;
Element                  getElementById(in DOMString elementId);

};
```

Attributes

title

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

referrer

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

domain

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

URL

The complete URI of the document.

Methods

getElementById

Returns the Element whose id is given by elementId. If no such element exists, returns null. Behavior is not defined if more than one element has this id.

Parameters

elementId The unique id value for an element.

Return Value

The matching element.

This method raises no exceptions.

6.9.3 The getSVGDocument method

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

Methods

`getSVGDocument`

Returns the SVGDocument object for the referenced SVG document.

No Parameters

Return Value

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

Exceptions

`DOMException` NO_SVG_DOCUMENT: No SVGDocument object is available.

6.9.4 Interface SVGElement

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

IDL Definition

```
interface SVGElement : Element {
  attribute DOMString id;
  attribute DOMString lang;
  attribute DOMString space;
  readonly attribute  SVGSVGElement ownerSVGElement;
  readonly attribute  SVGElement viewportElement;
};
```

Attributes

id

The value of the id attribute on the given element.

lang

The value of the xml:lang attribute on the given element.

lang

The value of the xml:space attribute on the given element.

ownerSVGElement

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

viewportElement

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

6.9.5 Interface SVGStyledElement

Interface SVGStyledElement is the base class for elements which can be styled but not transformed (i.e., elements which have a style attribute but not a transform attribute).

IDL Definition

```
interface SVGStyledElement : SVGElement {
  readonly attribute CSSStyleDeclaration  style;
           attribute DOMString            className;
};
```

Attributes

style

The value of the style attribute on the given element.

className

The value of the class attribute on the given element.

6.9.6 Interface SVGTransformedElement

Interface SVGTransformedElement is the base class for elements which can be transformed but not styled (i.e., elements which have a transform attribute but not a style attribute).

IDL Definition

```
interface SVGTransformedElement : SVGElement {
  readonly attribute  SVGElement       nearestViewportElement;
  readonly attribute  SVGElement       farthestViewportElement;
           attribute  SVGTransformList transform;

  SVGRect      getBBox();
  SVGMatrix    getCTM(); // returns CTM (userspace to [nearest 'svg'] viewport transform matrix)
  SVGMatrix    getTransformToElement(in SVGTransformedElement element)
                    raises(SVGException);
                    // returns transform matrix which maps coordinates from
                    // current user space to the user space of "element"
  SVGMatrix    getScreenCTM(); // returns CTM (userspace to screen units transform matrix)
};
```

Attributes

nearestViewportElement

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

farthestViewportElement

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

transform

The value of the transform attribute on the given element.

Methods

getBBox

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

No Parameters

Return Value

The bounding box.

Exceptions

`SVGException`	SVG_NO_GRAPHICS_ELEMENTS: There are no graphics elements on which to perform the given action.

getCTM

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

No Parameters

Return Value

The transformation matrix.

No Exceptions

getTransformToElement

Returns the transformation matrix from the user coordinate system on the current element (after application of the transform attribute) to the user coordinate system on **element** (after application of its transform attribute)

Parameters

element

The target element.

Return Value

The transformation matrix.

No Exceptions

getScreenCTM

Returns the transformation matrix from current user units (i.e., after application of the transform attribute) 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 know, then an algorithm similar to the CSS2 definition of a "pixel" can be

used instead.

No Parameters

Return Value

The transformation matrix.

No Exceptions

6.9.7 Interface SVGStyledAndTransformedElement

Interface SVGStyledAndTransformedElement is the base class for elements which can be both styled and transformed (i.e., elements which have both style and transform attributes).

IDL Definition

```
interface SVGStyledAndTransformedElement : SVGElement {
  readonly attribute CSSStyleDeclaration  style;
           attribute DOMString            className;

  readonly attribute  SVGElement       viewportElement; // element that established current viewport
           attribute  SVGTransformList transform;

  SVGRect      getBBox(); // tight bounding box on geometry of all contained
                          // graphics elements, in userspace.
                          // Doesn't take into account stroke-width or filter effects, for example

  SVGMatrix    getNearestCTM(); // returns CTM (userspace to [nearest 'svg'] viewport transform matrix)
  SVGMatrix    getNearestCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)
  SVGMatrix    getFurthestCTM(); // returns CTM (userspace to [outermost 'svg'] viewport transform matrix)
  SVGMatrix    getFurthestCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)
  SVGMatrix    getScreenCTM(); // returns CTM (userspace to screen units transform matrix)
  SVGMatrix    getScreenCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)
};
```

6.9.8 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.

IDL Definition

```
interface SVGSVGElement : SVGStyledElement {

  // Viewport definition in coordinate system of object
  // containing this 'svg' element.
  // (See coords.html for discussion of containing parent.
  // Values are unitless values in parent's coordinate system.
  // If parent uses CSS layout, then values represent CSS pixels.)
  readonly attribute SVGRect viewport;

  // Size of a CSS "pixel", using the CSS2 definition of a pixel,
  // which represents a unit somewhere in the range of 70dpi to 120dpi,
  // and, on systems that support this, might actually match the
  // characteristics of the target medium. On systems where it is
  // impossible to know the size of a pixel, a suitable default
  // pixel size is provided.
  readonly attribute float CSSPixelToMillimeterX;
  readonly attribute float CSSPixelToMillimeterY;

  // Size of a screen "pixel", which is the unit of size
  // returned to UI event handlers by the level 2 DOM.
  // (Again, various caveats about reliability of these values...)
  readonly attribute float ScreenPixelToMillimeterX;
  readonly attribute float ScreenPixelToMillimeterY;

  // If most recent user action was hyperlink into document
  // using SVGViewSpec, then this is true and the
  // "currentView" object overrides the standard viewing
  // attributes on 'svg' element.
  // Otherwise, if the user more recently did a "Normal size", then the
  // boolean becomes false, and the attributes on the 'svg' element
```

```
  // are active, not currentView.
  attribute boolean useCurrentView;
  readonly attribute SVGViewSpec currentView;

  // Corresponds to the various viewing attributes on the 'svg' element.
  // Active only if 'useCurrentView' is false.
  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;
  attribute boolean enableZoomAndPanControls;

  // Information about the current zoom and pan factors
  // relative to the base view (i.e., either currentView
  // or viewBox).
  // These attributes are equivalent to the 2x3 matrix
  // [a b c d e f] = [currentScale 0 0 currentScale currentTranslate.x currentTranslate.y]
  attribute float currentScale;
  attribute SVGPoint currentTranslate;


  // XML attributes on the 'svg' element.
  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
  attribute SVGLength refX;
  attribute SVGLength refY;

  // Methods to eliminate flicker in scripted animations.
  unsigned long suspendRedraw(in unsigned long max_wait_milliseconds);
  void          unsuspendRedraw(in unsigned long suspend_handle_id)
                                raises(DOMException);
  void          unsuspendRedrawAll();
  void          forceRedraw();


  // Methods to manage running animations.
  void          pauseAnimations()
                                raises(DOMException);
  void          unpauseAnimations()
                                raises(DOMException);
  boolean       animationsPaused();
  float         getDocumentBeginTime()
                                raises(DOMException);
  float         getCurrentTime()
                                raises(DOMException);
  void          setCurrentTime(in float seconds)
                                raises(DOMException);


  void          deSelectAll(); // Unselects any text strings that are currently selected.

  // Methods to create unattached standard object types.
  // Upon creation, these object types are unattached to
  // the document, but they can be assigned to document
  // attributes of the same type. For example:
  //    // Create a default SVGLength (zero user units).
  //    SVGLength myLength = createSVGLength();
  //    // Set the length to 2.3cm
  //    myLength.newValueSpecifiedUnits(myLength.kSVG_LENGTHTYPE_CM, 2.3);
  //    // Change the width of a rectangle to 2.3cm.
  //    myRect.width = myLength;
  SVGNumber              createSVGNumber();     // Returns unattached number 0
  SVGLength              createSVGLength();     // Returns unattached length of 0 user units
  SVGLengthList          createSVGLengthList(); // Returns unattached empty list
  SVGAngle               createSVGAngle();      // Returns unattached angle of 0 degrees
  SVGPoint               createSVGPoint();      // Returns unattached point (0,0) in user units
  SVGPointList           createSVGPointList();  // Returns unattached empty list of points
  SVGMatrix              createSVGMatrix();     // Returns unattached identity matrix. (e,f)=(0,0) in user units
  SVGPreserveAspectRatio createSVGPreserveAspectRatio();  // Returns unattached object with values 'none' and 'meet'
  SVGRect                createSVGRect();       // Returns unattached rect x=y=width=height=0 in user units.
  SVGTransformList       createSVGTransformList(); // Returns unattached SVGTransform spec with identity matrix
  SVGTransformList       createSVGTransformListFromMatrix(in SVGMatrix matrix); // Returns unattached SVGTransform spec
  SVGTransform           createSVGTransform();     // Returns unattached identity matrix
  SVGTransform           createSVGTransformFromMatrix(in SVGMatrix matrix); // Returns unattached transform
  SVGLengthList          createSVGTransformList(); // Returns unattached empty list
  SVGICCColor            createSVGICCColor();   // Returns empty unattached list
  SVGColor               createSVGColor();      // Returns RGBColor=black, no ICC color
  SVGPaint               createSVGPaint();      // Returns paint of 'none'

  // Generic event creation ability.
  Event                  createEvent(in DOMString type)
                                     raises(DOMException);

  Element                getElementById(in DOMString elementID);
};
```

Methods

suspendRedraw

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

Parameters

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

Return Value

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.

This method raises no exceptions.

unsuspendRedraw

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

Parameters

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

Return Value

None.

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

unsuspendRedrawAll

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

Parameters

None.

Return Value

None.

This method raises no exceptions.

6.9.9 Interface SVGGElement

The SVGGElement interface corresponds to the 'g' element.

IDL Definition

```
interface SVGGElement : SVGStyledAndTransformedElement {
};
```

6.9.10 Interface SVGDefsElement

The SVGDefsElement interface corresponds to the 'defs' element.

IDL Definition

```
interface SVGDefsElement : SVGStyledElement {
};
```

6.9.11 Interface SVGDescElement

The SVGDescElement interface corresponds to the 'desc' element.

IDL Definition

```
interface SVGDescElement : SVGStyledElement {
};
```

6.9.12 Interface SVGTitleElement

The SVGTitleElement interface corresponds to the 'title' element.

IDL Definition

```
interface SVGTitleElement : SVGStyledElement {
};
```

6.9.13 Interface SVGUseElement

The SVGUseElement interface corresponds to the 'use' element.

IDL Definition

```
interface SVGUseElement : SVGStyledAndTransformedElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;

  // XML attributes on the 'use' element.
  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
};
```

6.9.14 Interface SVGImageElement

The SVGImageElement interface corresponds to the 'image' element.

IDL Definition

```
interface SVGImageElement : SVGStyledAndTransformedElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;

  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;

  // XML attributes on the 'use' element.
  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
};
```

6.9.15 Interface SVGSymbolElement

The SVGSymbolElement interface corresponds to the 'symbol' element.

IDL Definition

```
interface SVGSymbolElement : SVGStyledElement {
  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;

  // XML attributes on the 'svg' element.
  attribute SVGLength refX;
  attribute SVGLength refY;
};
```

---

7 Coordinate Systems, Transformations and Units

Contents

7.1 Introduction

For all media, the SVG canvas describes "the space where the SVG content is rendered." The canvas is infinite for each dimension of the space, but rendering occurs relative to a finite rectangular region of the canvas. This finite rectangular region is called the SVG viewport. For visual media [CSS2-VISUAL], the SVG viewport is the viewing area where the user sees the SVG content.

The size of the SVG viewport (i.e., its width and height) is determined by a negotiation process (see Establishing the size of the initial viewport) between the SVG document fragment and its parent (real or implicit). Once that negotiation process is completed, the SVG user agent is provided the following information:

- an integer value that represents the width in "pixels" of the viewport

- an integer value that represents the height in "pixels" of the viewport
- (highly desirable but not required) a real number value that indicates how many millimeters a "pixel" represents

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, 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 and the user coordinate system is also called user space.

Lengths in SVG can be specified as:

- (if no unit designator is provided) values in user space -- for example, "15"
- (if a CSS unit specifier is provided) a length in CSS units -- for example, "15mm"

The supported CSS length unit specifiers 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 some of the various CSS unit specifiers (px, pt, pc, cm, mm, in, and percentages) 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 using any CSS positioning parameters on the outermost 'svg' element and the width= and height= XML attributes that are required on the 'svg' element to determine the viewport into which the SVG user agent can render the document. In the negotiation process, if the parent document uses CSS positioning and the outermost 'svg' element contains CSS positioning properties [CSS2-POSN] which are sufficient to establish the width of the viewport, then the CSS positioning properties establish the viewport's width; otherwise, the width= attribute on the outermost 'svg' element establishes the viewport's width. Similarly, if the parent document uses CSS positioning and the outermost 'svg' element contains CSS positioning properties [CSS2-POSN] which are sufficient to establish the height of the viewport, then the CSS positioning properties establish the viewport's height; otherwise, the height= attribute on the outermost 'svg' element establishes the viewport's height.

In the following example, an SVG graphic is embedded within a parent XML document which is formatted using CSS layout rules. The width="100px" and height="200px" attributes are used to determine the initial viewport:

```
<?xml version="1.0" standalone="yes"?>
<parent xmlns="http://some.url">

   <!-- SVG graphic -->
   <svg xmlns='http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'
      width="100px" height="200px">
      <path d="M100,100 Q200,400,300,100"/>
      <!-- rest of SVG graphic would go here -->
   </svg>

</parent>
```

Download this example

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" in the viewport. In most cases, such as stand-alone SVG documents or SVG document fragments embedded 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.

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="300px" height="100px">
  <desc>Example InitialCoords - SVG's initial coordinate system</desc>

  <g style="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 style="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 style="font-size:14 font-family:Verdana">
    <text x="10" y="20">(0,0)</text>
    <text x="240" y="20">(300,0)</text>
    <text x="10" y="90">(0,100)</text>
  </g>
</svg>
```

Example InitialCoords

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

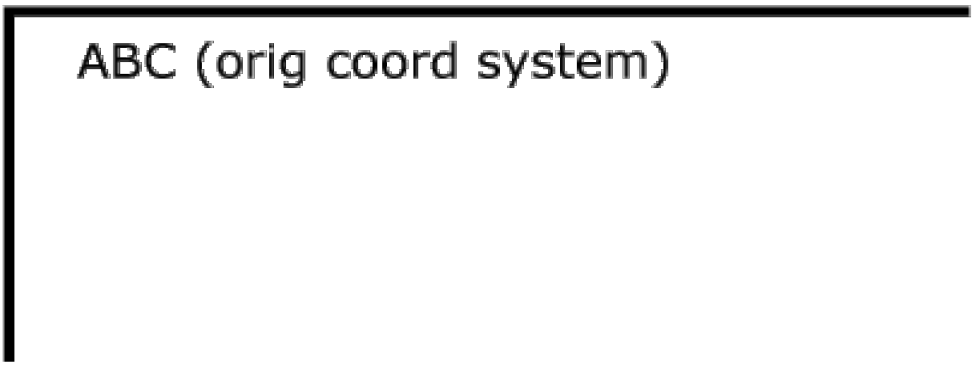
7.4 Coordinate system transformations

A new user space (i.e., a new current coordinate system) can be established by specifying transformations in the form of a transform attribute on a container element or graphics element. The transform attribute transforms all user space coordinates and lengths on the given element and all of its ancestors. Transformations can be nested, in which case the effect of the transformations are cumulative.

The following demonstrates simple transformations:

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="150px">
  <desc>Example OrigCoordSys - Simple transformations: original picture</desc>
  <g style="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" style="font-size:20 font-family:Verdana">
      ABC (orig coord system)
    </text>
  </g>
</svg>
```



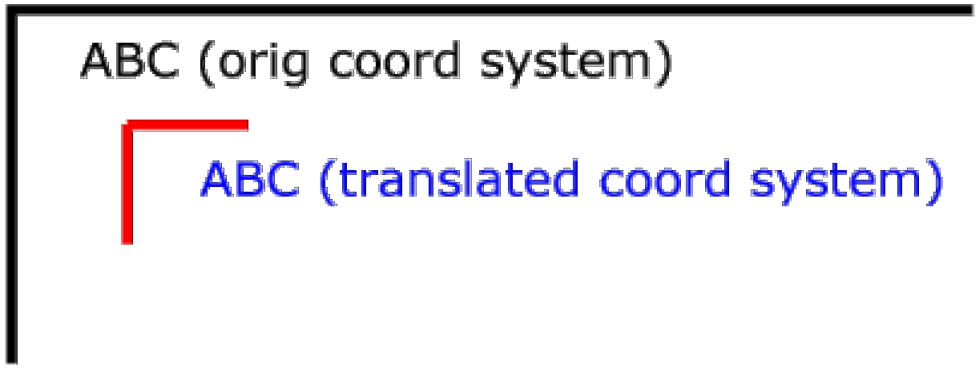
Example OrigCoordSys

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

Example NewCoordSys establishes a new user coordinate system by specifying transform="translate(50,50)" on the third 'g' element below. The new user coordinate system has its origin at location (50,50) in the original coordinate system. The result of this transformation is that the coordinate (30,30) in the new user coordinate system gets mapped to coordinate (80,80) in the original coordinate system (i.e., the coordinates have been translated by 50 units in X and 50 units in Y).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="150px">
  <desc>Example NewCoordSys - New user coordinate system</desc>
  <g style="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" style="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 style="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" style="stroke:red"/>
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="30" y="30" style="font-size:20 font-family:Verdana">
      ABC (translated coord system)
    </text>
  </g>
</svg>
```



Example NewCoordSys

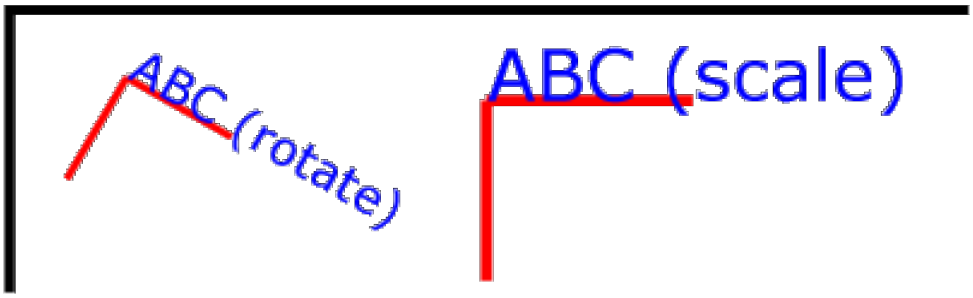
View this example as SVG (SVG-enabled browsers only)

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

- one which is the result of a translation by 50 units in X and 30 units in Y, followed by a rotation of 30 degrees
- another which is the result of a translation by 200 units in X and 40 units in Y, followed by a scale transformation of 1.5.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="120px">
  <desc>Example RotateScale - Rotate and scale transforms</desc>
  <g style="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 style="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" style="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 style="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" style="font-size:20; font-family:Verdana; fill:blue">
        ABC (scale)
      </text>
    </g>
  </g>
</svg>
```

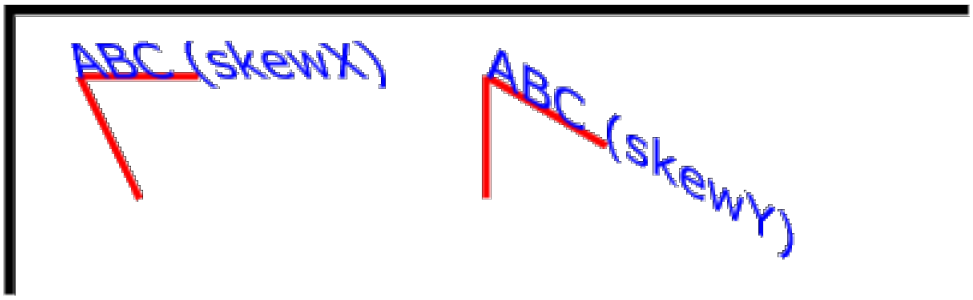


Example RotateScale

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

Example Skew defines two coordinate systems which are **skewed** relative to the origin coordinate system.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="120px">
  <desc>Example Skew - Show effects of skewX and skewY</desc>
  <g style="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 style="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" style="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 style="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" style="font-size:20; font-family:Verdana; fill:blue">
        ABC (skewY)
      </text>
    </g>
  </g>
</svg>
```



Example Skew

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

Mathematically, all transformations can be represented as 3x3 transformation matrices of the following form:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Since only six values are used in the above 3x3 matrix, a transformation matrix is also expressed as a vector: **[a b c d e f]**.

Transformations map coordinates and lengths from a new coordinate system into a previous coordinate system:

$$\begin{bmatrix} x_{prevCoordSys} \\ y_{prevCoordSys} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{newCoordSys} \\ y_{newCoordSys} \\ 1 \end{bmatrix}$$

Simple transformations are represented in matrix form as follows:

- Translation is equivalent to the matrix

  $$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[1 0 0 1 tx ty]**, where *tx* and *ty* are the distances to translate coordinates in *X* and *Y*, respectively.

- Scaling is equivalent to the matrix

  $$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  or **[sx 0 0 sy 0 0]**. One unit in the *X* and *Y* directions in the new coordinate system equals *sx* and *sy* units in the previous coordinate system, respectively.

- Rotation 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:

$$\begin{bmatrix} x_{prev} \\ y_{prev} \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{curr} \\ y_{curr} \\ 1 \end{bmatrix}$$

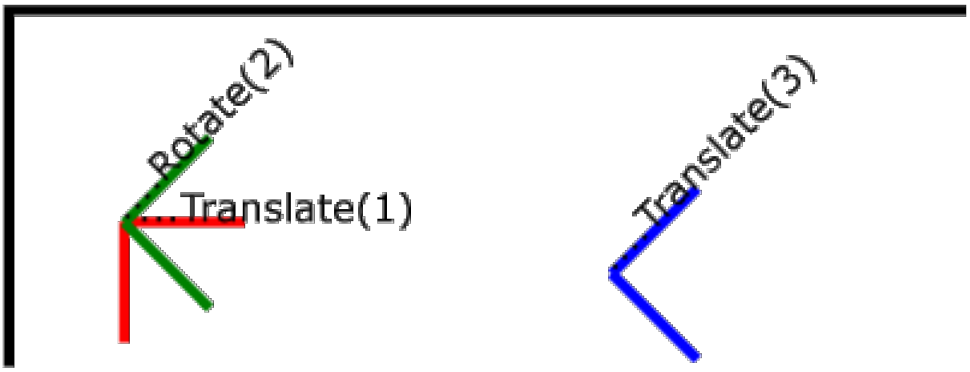
For each given element, the accumulation of all transformations that have been defined on the given element and all of its ancestors up to and including the element which established the current viewport (usually, the 'svg' element which is the most immediate ancestor to the given element) is called 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="400px" height="150px">
  <desc>Example Nested - Nested transformations</desc>
  <g style="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 style="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" style="font-size:16; font-family:Verdana">
      ....Translate(1)
    </text>
        <!-- Second, a rotate -->
    <g transform="rotate(-45)">
      <g style="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" style="font-size:16; font-family:Verdana">
        ....Rotate(2)
      </text>
          <!-- Third, another translate -->
      <g transform="translate(130,160)">
        <g style="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" style="font-size:16; font-family:Verdana">
          ....Translate(3)
        </text>
      </g>
    </g>
  </g>
</svg>
```



Example Nested

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

In the example above, the CTM within the 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 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]**. The *e* and *f* values can be specified with CSS unit specifiers.

- translate(<tx> [<ty>]), which specifies a translation by *tx* and *ty*. *tx* and *ty* values can be specified with CSS unit specifiers.

- 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>), which specifies a rotation by <rotate-angle> about the origin of the current user coordinate system.
- skewX(<skew-angle>), which specifies a skew transformation along the X axis.
- skewY(<skew-angle>), which specifies a skew transformation along the Y axis.

All numeric values are real numbers. All angle values are expressed according to the rules for basic data type <angle>.

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 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
       length comma-wsp
       length wsp* ")"

translate:
    "translate" wsp* "(" wsp* length ( comma-wsp length )? wsp* ")"

scale:
    "scale" wsp* "(" wsp* length ( comma-wsp number )? wsp* ")"

rotate:
    "rotate" wsp* "(" wsp* number wsp* ")"

skewX:
    "skewX" wsp* "(" wsp* number wsp* ")"

skewY:
    "skewY" wsp* "(" wsp* number wsp* ")"

length:
    number unit-specifier?

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"

unit-specifier:
    "em"| "ex" | "px" | "pt" | "pc" | "cm" | "mm" | "in" | "%"

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

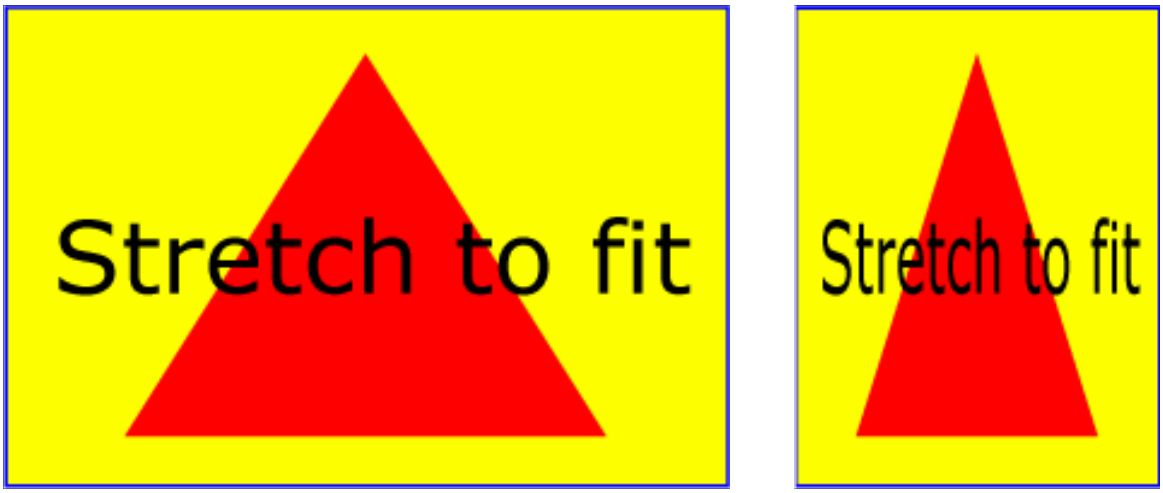
It is often desirable to specify that a given set of graphics stretch to fit a particular container element. The viewBox attribute provides this capability.

All elements that establish a new viewport (see elements that establish viewports) have attribute viewBox. The value of the viewBox attribute is a list of four numbers <min-x>, <min-y>, <width> and <height> 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.

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="300px" height="200px" viewBox="0 0 1500 1000">
  <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" style="fill:yellow" />

  <!-- A large, red triangle -->
  <path style="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" style="font-size:150; font-family:Verdana">
    Stretch to fit
  </text>
</svg>
```

Rendered into viewport with width=300px, height=200px	Rendered into viewport with width=150px, height=200px



Example ViewBox

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

The effect of the viewBox attribute is that the user agent automatically supplies the appropriate transformation matrix to map the specified rectangle in user space to the bounds of 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="300px" height="200px">

  <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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="150px" height="200px">

  <g transform="scale(0.1 0.2)">

    <!-- Rest of document goes here -->

  </g>
</svg>
```

(Note: in some cases the user agent will need to supply a **translate** transformation in addition to a **scale** transformation. For example, on an outermost 'svg', a **translate** transformation will be needed if the viewBox attributes specifies values other than zero for <min-x> or <min-y>.)

For the viewBox attribute:

Animatable: yes.

7.8 The preserveAspectRatio attribute

In some cases, it is necessary that force a uniform scaling transform to be used when utilizing viewBox, usually for the purposes of preserving the aspect ratio of the graphics being rendered in the viewport. A supplemental attribute preserveAspectRatio="<align> [<meetOrSlice>]", which is available for all elements that establish a new viewport (see elements that establish viewports), indicates whether or not to force uniform scaling. 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 (the default) - 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.
- 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 - 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 must be one of the following strings:

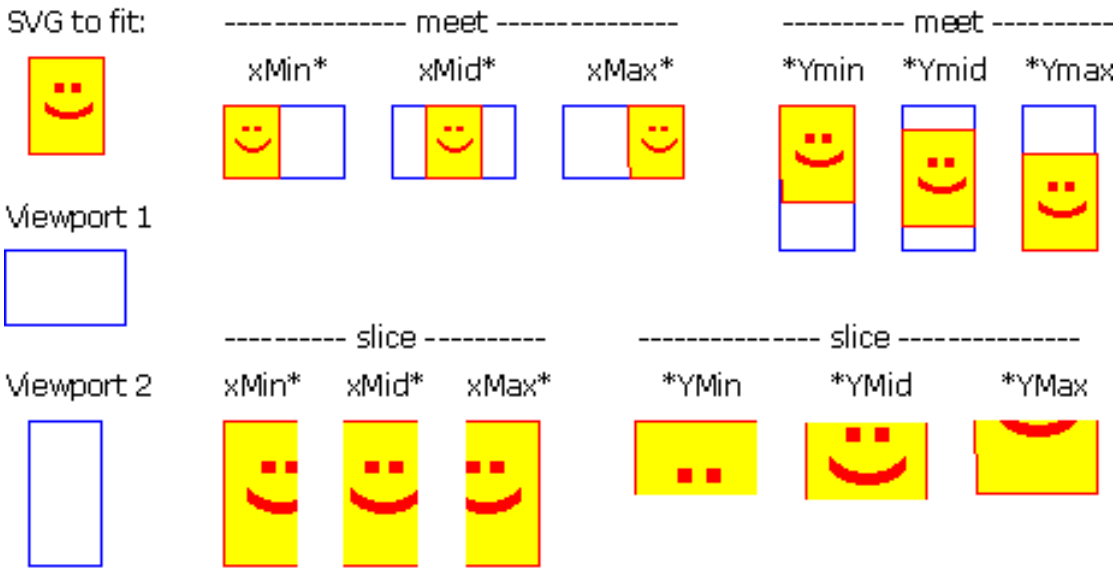
- meet (the default) - Scale the graphic such that:
  - aspect ratio is preserved
  - the entire viewBox is visible within the viewport
  - the viewBox is scaled up as much as possible, while still meeting the other criteria

  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). The smile is drawing the text string ":)" rotated 90 degrees.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd" [
<!ENTITY Smile "
  <rect x='.5' y='.5' width='29' height='39' style='fill:yellow; stroke:red'/>
  <g transform='rotate(90)'>
    <text x='10' y='10' style='font-family:Verdana;
          font-weight:bold; font-size:14'>:)</text>
  </g>">
<!ENTITY Viewport1 "<rect x='.5' y='.5' width='49' height='29'
                          style='fill:none; stroke:blue'/>">
<!ENTITY Viewport2 "<rect x='.5' y='.5' width='29' height='59'
                          style='fill:none; stroke:blue'/>">
]>
<svg width="480px" height="270px" style="font-family:Verdana; font-size:8">
  <desc>Example PreserveAspectRatio - demonstrate available options</desc>
  <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>
  <text x="100" y="30">--------------- meet ---------------</text>
  <g transform="translate(100,60)"><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(170,60)"><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(240,60)"><text y="-10">xMax*</text>&Viewport1;
    <svg preserveAspectRatio="xMaxYMax meet" viewBox="0 0 30 40"
         width="50" height="30">&Smile;</svg></g>
  <text x="330" y="30">---------- meet ----------</text>
  <g transform="translate(330,60)"><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(380,60)"><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(430,60)"><text y="-10">*YMax</text>&Viewport2;
    <svg preserveAspectRatio="xMaxYMax meet" viewBox="0 0 30 40"
         width="30" height="60">&Smile;</svg></g>
  <text x="100" y="160">---------- slice ----------</text>
  <g transform="translate(100,190)"><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(150,190)"><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(200,190)"><text y="-10">xMax*</text>&Viewport2;
    <svg preserveAspectRatio="xMaxYMax slice" viewBox="0 0 30 40"
         width="30" height="60">&Smile;</svg></g>
  <text x="270" y="160">--------------- slice ---------------</text>
  <g transform="translate(270,190)"><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(340,190)"><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(410,190)"><text y="-10">*YMax</text>&Viewport1;
    <svg preserveAspectRatio="xMaxYMax slice" viewBox="0 0 30 40"
         width="50" height="30">&Smile;</svg></g>
</svg>
```



Example PreserveAspectRatio

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

For the preserveAspectRatio attribute:

Animatable: yes.

7.9 Establishing a new viewport

At any point in an SVG drawing, you can establish a new viewport into which all contained graphics is drawn by including an 'svg' element inside SVG content. By establishing a new viewport, you also implicitly establish a new initial user space, new meanings for many of the CSS unit specifiers and, potentially, a new clipping path. The bounds of the new viewport are defined by the x, y, width and height attributes on the 'svg' element. Here is an example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <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.

In addition to the 'svg' element, the following other elements also establish a new viewport:

- A 'use' or 'image'element establishes a temporary new viewport for drawing instances of referenced elements or files
- A 'marker' element establishes a temporary new viewport for drawing arrowheads and polymarkers
- When the text on a path facility tries to draw a referenced 'symbol' or 'svg' element, it establishes a new temporary new viewport for the referenced graphic.
- When a pattern is used to fill or stroke an object by reference to a 'pattern' element, a temporary new viewport

is established for each drawn instance of the pattern.

- When a 'mask' element is used to establish a mask for an object and maskUnits="objectBoundingBox", a temporary new viewport is established to draw the elements within the 'mask' element.

Whether a new viewport also establishes a new additional clipping path is determined by the value of the 'overflow' property on the element which 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 in one of the following ways:

- User units. If no unit specifier is provided, a given coordinate or length is assumed to be in user units (i.e., a value in user space). For example:

```
<text style="font-size: 50">Text size is 50 user units</text>
```

- CSS units. If a CSS unit specifier is provided on a coordinate or length value, then the given value is assumed to be in CSS units. Available CSS unit specifiers are the absolute and relative unit specifiers from CSS (em, ex, px, pt, pc, cm, mm, in and percentages). As in CSS, the *em* and *ex* unit specifiers are relative to the current font's *font-size* and *x-height*, respectively. Initially, the various absolute unit specifiers from CSS (i.e., px, pt, pc, cm, mm, in) represent lengths within the initial user coordinate system and do not change their meaning as transformations alter the current coordinate system. Thus, "12pt" can be made to represent exactly 12 points on the actual visual medium even if the user coordinate system has been scaled. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Demonstration of coordinate transforms
  </desc>
  <!-- The following two text elements will both draw with a
        font height of 12 pixels -->
   <text style="font-size: 12">This prints 12 pixels high.</text>
   <text style="font-size: 12px">This prints 12 pixels high.</text>

   <!-- Now scale the coordinate system by 2. -->
   <g transform="scale(2)">

      <!-- The following text will actually draw 24 pixels high
           because each unit in the new coordinate system equals
           2 units in the previous coordinate system. -->
      <text style="font-size: 12">This prints 24 pixels high.</text>

      <!-- The following text will actually still draw 12 pixels high
           because the CSS unit specifier has been provided. -->
      <text style="font-size: 12px">This prints 12 pixels high.</text>

   </g>
</svg>
```

Download this example

If possible, the SVG user agent must be passed the actual size of a *px* unit in inches or millimeters by its parent user agent. (See Conformance Requirements and Recommendations.) If such information is not available from the parent user agent, then the SVG user agent shall assume a *px* is defined to be exactly .28mm.

7.11 Redefining the meaning of CSS unit specifiers

The process of establishing a new viewport, such as when there is 'svg' element inside of another SVG 'svg', changes the meaning of the following CSS unit specifiers: px, pt, pc, cm, mm, in, and % (percentages). A "pixel" (the px unit) becomes equivalent to a single unit in the user coordinate system for the given 'svg' element. The meaning of the other absolute unit specifiers (pt, pc, cm, mm, in) are determined as an appropriate multiple of a *px* unit using the actual size of *px* unit (as passed from the parent user agent to the SVG user agent). Any percentage values that are relative to the current viewport will also represent new values.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="300px" height="300px">
  <desc>Transformation with establishment of a new viewport
  </desc>
  <!-- The following two text elements will both draw with a
        font height of 12 pixels -->
   <text style="font-size: 12">This prints 12 pixels high.</text>
   <text style="font-size: 12px">This prints 12 pixels high.</text>

   <!-- Now scale the coordinate system by 2. -->
   <g transform="scale(2)">

      <!-- The following text will actually draw 24 pixels high
           because each unit in the new coordinate system equals
           2 units in the previous coordinate system. -->
      <text style="font-size: 12">This prints 24 pixels high.</text>

      <!-- The following text will actually still draw 12 pixels high
           because the CSS unit specifier has been provided. -->
      <text style="font-size: 12px">This prints 12 pixels high.</text>
   </g>

   <!-- This time, scale the coordinate system by 3. -->
   <g transform="scale(3)">

      <!-- Establish a new viewport and thus change the meaning of
           some CSS unit specifiers. -->
      <svg style="left:0; top:0; right:100; bottom:100"
           width="100%" height="100%">

         <!-- The following two text elements will both draw with a
              font height of 36 screen pixels. The first text element
              defines its height in user coordinates, which have been
              scaled by 3. The second text element defines its height
              in CSS px units, which have been redefined to be three times
              as big as screen pixels due the <svg> element establishing
              a new viewport. -->
         <text style="font-size: 12">This prints 36 pixels high.</text>
         <text style="font-size: 12px">This prints 36 pixels high.</text>

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

Download this example

7.12 Processing rules for CSS units and percentages

Any values expressed in CSS units or percentages of the current viewport shall be implemented such that these values map to corresponding values in user space as follows:

- For any x-coordinate value or width value (xValueInVPSpace) expressed using CSS units (other than percentages), first convert xValueInVPSpace into viewport pixel units using the SVG user agent's standard conversion factor from pixels to real world units (e.g., millimeters) to yield xValueInVPPixels. Then transform the points (0,0) and (xValueInVPPixels,0), from viewport space to current user space using the inverse of the current transformation matrix, yielding two points in userspace Q1 and Q2. Do a distance calculation between

Q1 and Q2 (sqrt((Q2x-Q1x)**2 + (Q2y-Q1y)**2)) and use that as the value for the given operation.

- For any y-coordinate value or height value (yValueInVPSpace) expressed using CSS units (other than percentages), do the same thing as above, but use points (0,0) and (0,yValueInVPPixels) instead.
- For any x-coordinate value or width value (xValueInVPSpace) expressed as a percentage of the viewport, transform the points (0,0) and (percentageValue*vpWidthInPixels,0), from viewport space to current user space using the inverse of the current transformation matrix, yielding two points in userspace Q1 and Q2. Do a distance calculation between Q1 and Q2 (sqrt((Q2x-Q1x)**2 + (Q2y-Q1y)**2)) and use that as the value for the given operation.
- For any y-coordinate value or height value (yValueInVPSpace) expressed as a percentage of the viewport, do the same thing as above, but use points (0,0) and (0,percentageValue*vpHeightInPixels) instead.
- For any other length value in viewport space (lengthVPSpace), the following approach is used to give appropriate weighting to the contribution of the two dimensions of the viewport. First, convert lengthVPSpace into viewport pixel units using the SVG user agent's standard conversion factor from pixels to to real world units (e.g., millimeters) to yield lengthVPPixels. Calculate the distance from (0,0) and (vpWidthInPixels,vpHeightInPixels) in viewport space using the formula: vpDiagLengthVPPixels=sqrt(vpWidthInPixels**2 + vpHeightInPixels**2). Using the inverse of the current transformation matrix, determine the points in user space (P1x,P1y) and (P2x,P2y) which correspond to the points (0,0) and (vpWidthInPixels,vpHeightInPixels) in viewport space. Calculate the distance from (P1x,P1y) and (P2x,P2y) in user space using the formula: vpDiagLengthUserSpace=sqrt((P2x-P1x)**2 + (P2y-P1y)**2)). Then, convert the original viewport-relative length into a length in user space using the formula: lengthUserSpace = lengthVPPixels * (vpDiagLengthUserSpace/vpDiagLengthVPPixels).
- If a viewport-relative percentage value is given, then calculate lengthVPPixels as lengthVPPixels=percentageValue*sqrt(vpWidthPixels**2 + vpHeightPixels**2)/sqrt(2).

7.13 DOM interfaces

7.13.1 Overview

This section describes the SVG-specific DOM interfaces that correspond to the topics described in this chapter.

7.13.2 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]
```

```
interface SVGPoint  {
  attribute SVGLength x;
  attribute SVGLength y;

  SVGPoint    matrixTransform(in SVGMatrix matrix); // vector' = matrix * vector
};
```

7.13.3 Interface SVGMatrix

```
interface SVGMatrix  {
  attribute float a;
  attribute float b;
  attribute float c;
  attribute float d;
  attribute SVGLength e;
  attribute SVGLength f;

  // Matrix utility functions.
  SVGMatrix    multiply(in SVGMatrix secondMatrix); // post-multiply secondMatrix
  SVGMatrix    inverse()
                        raises(SVGException);  // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE).
  SVGMatrix    translate(in SVGLength x, in SVGLength y); // same as translate transform.
  SVGMatrix    scale(in float scaleFactor); // same as scale transform.
  SVGMatrix    scaleNonOrthogonal(in float scaleFactorX, in float scaleFactorY);
  SVGMatrix    rotate(in SVGAngle angle); // same as rotate transform.
  SVGMatrix    rotateFromVector(in SVGLength x, in SVGLength y); // atan(y/x)
  SVGMatrix    flipX();  // +x <-> -x
  SVGMatrix    flipY();  // +y <-> -y
  SVGMatrix    skewX(in SVGAngle angle);  // same as skewX transform.
  SVGMatrix    skewY(in SVGAngle angle);  // same as skewY transform.
};
```

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]
```

7.13.4 Interfaces SVGTransformList and SVGTransform

The SVGTransformList and SVGTransforminterfaces correspond to the various attributes which specify a transformations, such as the transform attribute, which is available for many of SVG's elements.

```
interface SVGTransformList {
  SVGTransform      createSVGTransform(); // Returns unattached identify matrix
  SVGTransform      createSVGTransformFromMatrix(in SVGMatrix matrix); // Returns unattached SVGTransform

  readonly attribute unsigned long  number_of_transforms;
  SVGTransform      getTransform(in unsigned long index);

  // Replace all existing entries with a single entry.
  void              initialize(in SVGTransform newSVGTransform)
                                       raises(DOMException);
  void              clear(); // Clear all entries, giving an empty list
  SVGTransform      insertBefore(in SVGTransform newSVGTransform,
                                       in unsigned long index)
                                       raises(DOMException);
  SVGTransform      replace(in SVGTransform newSVGTransform,
                                in unsigned long index)
                                raises(DOMException);
  SVGTransform      remove(in unsigned long index)
                               raises(DOMException);
  SVGTransform      append(in SVGTransform newSVGTransform)
                                raises(DOMException);

  // Consolidate all existing transforms into a single matrix transform
  SVGTransform      consolidate();
};
```

```
interface SVGTransform {
  // Transform Types
  const unsigned short kSVG_TRANSFORM_UNKNOWN   = 0;
  const unsigned short kSVG_TRANSFORM_MATRIX    = 1;
  const unsigned short kSVG_TRANSFORM_TRANSLATE = 2;
  const unsigned short kSVG_TRANSFORM_SCALE     = 3;
  const unsigned short kSVG_TRANSFORM_ROTATE    = 4;
  const unsigned short kSVG_TRANSFORM_SKEWX     = 5;
  const unsigned short kSVG_TRANSFORM_SKEWY     = 6;
  readonly attribute  unsigned short type;

  // a,b,c,d,e,f represent a matrix that represents the given transformation.
  // angle is a convenience value for kSVG_TRANSFORM_ROTATE,
  // kSVG_TRANSFORM_SKEWX and kSVG_TRANSFORM_SKEWY.
  //
  // For kSVG_TRANSFORM_MATRIX, the matrix contains the a, b, c, d, e, f values supplied by the user.
  //   (angle=0).
  // For kSVG_TRANSFORM_TRANSLATE, e and f represents the translation amounts.
  //   (a=1,b=0,c=0,d=1,angle=0).
  // For kSVG_TRANSFORM_SCALE, a and d represents the scale amounts.
  //   (b=0,c=0,e=0,f=0,angle=0).
  // For kSVG_TRANSFORM_ROTATE, kSVG_TRANSFORM_SKEWX and kSVG_TRANSFORM_SKEWY,
  // a, b, c and d represent the matrix which will result in the given transformation.
  // angle contains the angle specified for the operation.
  //   (e=0,f=0).
  attribute  SVGMatrix    matrix;
  attribute  SVGAngle     angle;
};
```

7.13.5 Interface SVGPreserveAspectRatio

The SVGPreserveAspectRatio interface corresponds to the preserveAspectRatio attribute, which is available for some of SVG's elements.

```
interface SVGPreserveAspectRatio {
  // Alignment Types
  const unsigned short kSVG_PRESERVEASPECTRATIO_NONE     = 0;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMINYMIN = 1;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMIDYMIN = 2;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMAXYMIN = 3;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMINYMID = 4;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMIDYMID = 5;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMAXYMID = 6;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMINYMAX = 7;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMIDYMAX = 8;
  const unsigned short kSVG_PRESERVEASPECTRATIO_XMAXYMAX = 9;
  readonly attribute  unsigned short align;

  // Meet-or-slice Types
  const unsigned short kSVG_MEETORSLICE_MEET  = 1;
  const unsigned short kSVG_MEETORSLICE_SLICE = 2;
  readonly attribute  unsigned short meetOrSlice;
};
```

---

8 Paths

Contents

8.1 Introduction

Paths represent the outline of a shape which can be filled, stroked, (see Filling, Stroking and Paint Servers) used as a clipping path (see Clipping, Masking and Compositing), or for any combination of the three.

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 an outline of an object which is defined in terms of *moveto* (set a new current point), *lineto* (draw a straight line), *curveto* (draw a curve using a cubic bezier), *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 subpaths, each consisting of a single *moveto* followed by one or more line or curve operations) are possible to allow effects such as "donut holes" in objects.

A path is defined in SVG using the 'path' element.

8.2 The 'path' element

```
<!ENTITY % pathExt "" >
<!ELEMENT path (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%pathExt;)*) >
<!ATTLIST path
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  d CDATA #REQUIRED
  nominalLength CDATA #IMPLIED >
```

Attribute definitions:

d = "*path data*"

The definition of the outline of a shape. See Path data.
Animatable: yes.

nominalLength = "<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 nomimalLength to the user agent's own computed value for total path length. nomimalLength potentially affects calculations for text on a path, motion animation and various stroke operations.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

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 beziers), *arc* and *closepath* instructions. The following example 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="yes"?>
<svg width="4in" height="3in"
     xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <path d="M 100 100 L 140 100 L 120 140 z"/>
</svg>
```

Download this example

Path data values 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 a path data value.

The syntax of path data is very abbreviated 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 (upper case means absolute coordinates, lower case 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 period (".") and no other delimiter characters are allowed. (For example, the following is an invalid numeric value in a path data stream: "13,000.56". Instead, say: "13000.56".)

In the tables below, the following notation is used:

- (): grouping of parameters
- +: 1 or more of the given parameter(s) is required

The following sections list the commands.

8.3.2 The "moveto" commands

The "moveto" commands (**M** or **m**) establish a new current point. The effect is as if the "pen" were lifted and moved to a new location. A path data segment must begin with either one of the "moveto" commands or one of the "arc" commands. 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 relative moveto (**m**) appears as the first element of the path, then it is treated as a pair of absolute coordinates. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.

8.3.3 The "closepath" command

The "closepath" (**Z** or **z**) causes an automatic straight line to be drawn from the current point to the initial point of the current subpath. "Closepath" 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':

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 most recent starting point (usually, the most recent moveto point).

8.3.4 The "lineto" commands

The various "lineto" commands draw straight lines from the current point to a new point:

Command	Name	Parameters	Description

L (absolute) **l** (relative)	lineto	(x y)+	Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. **L** (uppercase) indicates that absolute coordinates will follow; **l** (lowercase) indicates that relative coordinates will follow. A number of coordinates pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final set of coordinates provided.
H (absolute) **h** (relative)	horizontal lineto	x+	Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). **H** (uppercase) indicates that absolute coordinates will follow; **h** (lowercase) indicates that relative coordinates will follow. Multiple x values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (x, cpy) for the final value of x.
V (absolute) **v** (relative)	vertical lineto	y+	Draws a vertical line from the current point (cpx, cpy) to (cpx, y). **V** (uppercase) indicates that absolute coordinates will follow; **v** (lowercase) indicates that relative coordinates will follow. Multiple y values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (cpx, y) for the final value of y.

8.3.5 The curve commands

These three groups of commands that draw curves:

- Cubic bezier commands (**C**, **c**, **S** and **s**). A cubic bezier segment is defined by a start point, an end point, and two control points.
- Quadratic bezier commands (**Q**, **q**, **T** and **T**). A quadratic bezier 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.

The cubic bezier commands are as follows:

Command	Name	Parameters	Description

C (absolute) **c** (relative)	curveto	(x1 y1 x2 y2 x y)+	Draws a cubic bezier 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 polybezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybezier.
S (absolute) **s** (relative)	shorthand/smooth curveto	(x2 y2 x y)+	Draws a cubic bezier 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 polybezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybezier.

The quadratic bezier commands are as follows:

Command	Name	Parameters	Description

Q (absolute) **q** (relative)	quadratic bezier curveto	(x1 y1 x y)+	Draws a quadratic bezier 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 polybezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybezier.
T (absolute) **t** (relative)	Shorthand/smooth quadratic bezier curveto	(x y)+	Draws a quadratic bezier 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 an 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 polybezier.

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 is defined 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.

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: (Pictures will be forthcoming in a future version of the spec) **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)).

(We need examples to illustrate all of this! Here is one for the moment. Suppose you have a circle with center (5,5) and radius 2 and you wish to draw an arc from 0 degrees to 90 degrees. Then one way to achieve this would be `M 7,5 A 2,2 0 0 1 5,7`. In this example, you move to the "0 degree" location on the circle, which is (7,5), since the center is at (5,5) and the circle has radius 2. Since we have circle, the two radii are the same, and in this example both are equal to 2. Since our sweep is 90 degrees, which is less than 180, we set large-arc-flag to 0. We want to draw the sweep in a positive angle direction, so we set sweep-flag to 1. Since we want to draw the arc to the point which is at the 90 degree location of the circle, we set (x,y) to (5,7).)

8.3.6 The grammar for path data

The following notation is used in the 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* subpaths? wsp*

subpaths:
    subpath
    | subpath subpaths

subpath:
    moveto subpath-elements?

subpath-elements:
    subpath-element
    | subpath-element wsp* subpath-elements

subpath-element:
    closepath
    | lineto
    | horizontal-lineto
    | vertical-lineto
    | curveto
    | smooth-curveto
    | quadratic-bezier-curveto
    | smooth-quadratic-bezier-curveto
    | elliptical-arc

moveto:
    ( "M" | "m" ) wsp* moveto-argument-sequence

moveto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? lineto-argument-sequence

closepath:
    ("Z" | "z")

lineto:
    ( "L" | "l" ) wsp* lineto-argument-sequence

lineto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? lineto-argument-sequence

horizontal-lineto:
    ( "H" | "h" ) wsp* horizontal-lineto-argument-sequence

horizontal-lineto-argument-sequence:
    coordinate
    | coordinate comma-wsp? horizontal-lineto-argument-sequence
```

```
vertical-lineto:
    ( "V" | "v" ) wsp* vertical-lineto-argument-sequence

vertical-lineto-argument-sequence:
    coordinate
    | coordinate comma-wsp? vertical-lineto-argument-sequence

curveto:
    ( "C" | "c" ) wsp* curveto-argument-sequence

curveto-argument-sequence:
    curveto-argument
    | curveto-argument comma-wsp? curveto-argument-sequence

curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair comma-wsp? coordinate-pair

smooth-curveto:
    ( "S" | "s" ) wsp* smooth-curveto-argument-sequence

smooth-curveto-argument-sequence:
    smooth-curveto-argument
    | smooth-curveto-argument comma-wsp? smooth-curveto-argument-sequence

smooth-curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair

quadratic-bezier-curveto:
    ( "Q" | "q" ) wsp* quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument-sequence:
    quadratic-bezier-curveto-argument
    | quadratic-bezier-curveto-argument comma-wsp?
        quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument:
    coordinate-pair comma-wsp? coordinate-pair

smooth-quadratic-bezier-curveto:
    ( "T" | "t" ) wsp* smooth-quadratic-bezier-curveto-argument-sequence

smooth-quadratic-bezier-curveto-argument-sequence:
    coordinate-pair
    | coordinate-pair comma-wsp? smooth-quadratic-bezier-curveto-argument-sequence

elliptical-arc:
    ( "A" | "a" ) wsp* elliptical-arc-argument-sequence

elliptical-arc-argument-sequence:
    elliptical-arc-argument
    | elliptical-arc-argument comma-wsp? elliptical-arc-argument-sequence

elliptical-arc-argument:
    nonnegative-number comma-wsp? nonnegative-number comma-wsp?
        number comma-wsp? flag comma-wsp? flag comma-wsp? coordinate-pair

coordinate-pair:
    coordinate comma-wsp? coordinate

coordinate:
    number

nonnegative-number:
    integer-constant
    | floating-point-constant

number:
    sign? integer-constant
```

```
    | sign? floating-point-constant

flag:
    "0" | "1"

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

comma:
    ","

integer-constant:
    digit-sequence

floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent

fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."

exponent:
    ( "e" | "E" ) sign? digit-sequence

sign:
    "+" | "-"

digit-sequence:
    digit
    | digit digit-sequence

digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

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

The processing of the BNF must consume as much of a given BNF production as possible, stopping at the point when a character is encountered which no longer satisfies the production. Thus, in the string "M 100-200", the first coordinate for the "moveto" consumes the characters "100" and stops upon encountering the minus sign because the minus sign cannot follow a digit in the production of a "coordinate". The result is that the first coordinate will be "100" and the second coordinate will be "-200".

Similarly, for the string "M 0.6.5", the first coordinate of the "moveto" consumes the characters "0.6" and stops upon encountering the second decimal point because the production of a "coordinate" only allows one decimal point. The result is that the first coordinate will be "0.6" and the second coordinate will be ".5".

8.4 Distance along a path

Various operations, including text on a path and motion animation and various stroke operations, require that the user agent compute the distance along the geometry of a graphics element, such as a 'path'.

Exact mathematics exist for computing distance along a path, but the formulas are highly complex and require substantial computation. It is recommended that authoring products and user agents employ algorithms that produce as precise results as possible; however, to accommodate implementation differences and to help distance calculations produce results that approximate author intent, the nomimalLength 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 nomimalLength 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 SVGPathElement

The SVGPathElement interface corresponds to the 'path' element.

```
interface SVGPathElement : SVGStyledAndTransformedElement {
  attribute SVGLength nominalLength;

  // Create an empty SVGPathSeg, specifying the type via a number.
  // All values initialized to zero.
  SVGPathSeg  createSVGPathSeg(in unsigned short pathsegType)
                                    raises(DOMException);

  // Create an empty SVGPathSeg, specifying the type via a single character.
  // All values initialized to zero.
  SVGPathSeg  createSVGPathSegFromLetter(in DOMString pathsegTypeAsLetter)
                                          raises(DOMException);

  // Create an SVGPathSeg, specifying the path segment as a string.
  // For example, "M 100 200". All irrelevant values are set to zero.
  SVGPathSeg  createSVGPathSegFromString(in DOMString pathsegString)
                                          raises(DOMException);

  // This set of methods allows retrieval and modification
  // to the path segments attached to this path object.
  // All 20 defined types of path segments are available
  // through these attributes and methods.

  readonly attribute unsigned long  number_of_pathsegs;
  SVGPathSeg      getSVGPathSeg(in unsigned long index);
  DOMString       getSVGPathSegAsString(in unsigned long index);

  // Replace all existing entries with a single entry.
  void            initialize(in SVGPathSeg newSVGPathSeg)
                                          raises(DOMException);
  void            clear(); // Clear all entries, giving an empty list
  SVGPathSeg      insertBefore(in SVGPathSeg newSVGPathSeg,
                                          in unsigned long index)
                                          raises(DOMException);
  SVGPathSeg      replace(in SVGPathSeg newSVGPathSeg,
                                           in unsigned long index)
                                           raises(DOMException);
  SVGPathSeg      remove(in unsigned long index)
                                    raises(DOMException);
  SVGPathSeg      append(in SVGPathSeg newSVGPathSeg)
                                    raises(DOMException);

  // This alternate set of methods also allows retrieval and modification
  // to the path segments attached to this path object.
  // These attributes and methods provide a "normalized" view of
  // the path segments where the path is expressed in terms of
  // the following subset of SVGPathSeg types:
```

```
  // kSVG_PATHSEG_MOVETO_ABS (M), kSVG_PATHSEG_LINETO_ABS (L),
  // kSVG_PATHSEG_CURVETO_CUBIC_ABS (C) and kSVG_PATHSEG_CLOSEPATH (z).
  // Note that number_of_pathsegs and number_of_normalized_pathsegs
  // are not always the same. In particular, elements such as arcs may
  // be expanded into multiple kSVG_PATHSEG_CURVETO_CUBIC_ABS (C)
  // pieces when retrieved in the "normalized" view of the path object.

  readonly attribute unsigned long  number_of_normalized_pathsegs;
  SVGPathSeg      getNormalizedSVGPathSeg(in unsigned long index);
  DOMString       getNormalizedSVGPathSegAsString(in unsigned long index);
  SVGPathSeg      insertNormalizedBefore(in SVGPathSeg newSVGPathSeg,
                              in unsigned long index)
                              raises(DOMException);
  SVGPathSeg      replaceNormalized(in SVGPathSeg newSVGPathSeg,
                              in unsigned long index)
                              raises(DOMException);
  SVGPathSeg      removeNormalized(in unsigned long index)
                              raises(DOMException);
  SVGPathSeg      appendNormalized(in SVGPathSeg newSVGPathSeg)
                              raises(DOMException);

  // This set of methods performs various distance-along-a-path calculations.

  float          getTotalLength();
  SVGPoint        getPointAtLength(in float distance);
  SVGPathSeg      getPathSegAtLength(in float distance);
};
```

8.5.2 Interface SVGPathSeg

The SVGPathSeg interface corresponds to a single command within a path data specification.

```
interface SVGPathSeg {
  // Path Segment Types
  const unsigned short kSVG_PATHSEG_UNKNOWN                    = 0;  // ?
  const unsigned short kSVG_PATHSEG_CLOSEPATH                  = 1;  // z
  const unsigned short kSVG_PATHSEG_MOVETO_ABS                 = 2;  // M
  const unsigned short kSVG_PATHSEG_MOVETO_REL                 = 3;  // m
  const unsigned short kSVG_PATHSEG_LINETO_ABS                 = 4;  // L
  const unsigned short kSVG_PATHSEG_LINETO_REL                 = 5;  // l
  const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_ABS          = 6;  // C
  const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_REL          = 7;  // c
  const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_ABS      = 8;  // Q
  const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_REL      = 9;  // q
  const unsigned short kSVG_PATHSEG_ARC_ABS                    = 10; // A
  const unsigned short kSVG_PATHSEG_ARC_REL                    = 11; // a
  const unsigned short kSVG_PATHSEG_LINETO_HORIZONTAL_ABS      = 12; // H
  const unsigned short kSVG_PATHSEG_LINETO_HORIZONTAL_REL      = 13; // h
  const unsigned short kSVG_PATHSEG_LINETO_VERTICAL_ABS        = 14; // V
  const unsigned short kSVG_PATHSEG_LINETO_VERTICAL_REL        = 15; // v
  const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_SMOOTH_ABS   = 16; // S
  const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_SMOOTH_REL   = 17; // s
  const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS = 18; // T
  const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL = 19; // t

  readonly attribute  unsigned short pathsegType;
  readonly attribute  DOMString      pathsegTypeAsLetter;

  // Attribute values for a path segment.
  // Each pathseg has slots for any possible path seg type.
  attribute  float       x;   // end point for pathseg
  attribute  float       y;   // end point for pathseg
  attribute  float       x0,y0;         // for bezier control points
```

```
  attribute  float       x1,y1;         // for bezier control points
  attribute  float       r1,r2;         // radii for A/a
  attribute  float       angle;         // A/a
  attribute  boolean     largeArcFlag;  // for A/a
  attribute  boolean     sweepFlag;     // for A/a

  readonly attribute  Path parentPath;
  readonly attribute  SVGDocument ownerSVGDocument;
  readonly attribute  SVGPathSeg previousSibling;
  readonly attribute  SVGPathSeg nextSibling;
};
```

---

9 Basic Shapes

Contents

9.1 Introduction

SVG contains the following set of basic shape elements:

- rectangles (rectangle, including optional rounded corners)
- circles
- ellipses
- lines
- polylines

- polygons

Mathematically, these shape elements are equivalent to a 'path' element that would construct the same shape. The basic shapes may be stroked, filled and used as clip paths. All of the properties available for 'path' elements also apply to the basic shapes.

9.2 The 'rect' element

The 'rect' element defines a rectangle which is axis-aligned with the current user coordinate system. Rounded rectangles can be achieved by setting appropriate values for attributes rx and ry.

```
<!ENTITY % rectExt "" >
<!ELEMENT rect (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%rectExt;)*) >
<!ATTLIST rect
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  rx CDATA #IMPLIED
  ry CDATA #IMPLIED >
```

Attribute definitions:

x = "<coordinate>"

The X-axis coordinate of the side of the rectangle which has the smaller X-axis coordinate value in the current user coordinate system.
The default value is "0".
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.
The default value is "0".
Animatable: yes.

width = "<length>"

The width of the rectangle.
Animatable: yes.

height = "<length>"

The height of the rectangle.

Animatable: yes.

rx = "<length>"

For rounded rectangles, the x-axis radius of the ellipse used to round off the corners of the rectangle.
The default value is "0".
Animatable: yes.

ry = "<length>"

For rounded rectangles, the y-axis radius of the ellipse used to round off the corners of the rectangle.
The default value is "0".
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

If a properly specified value is provided for rx but not for ry, then the user agent processes the 'rect' element with the effective value for ry as equal to rx. If a properly specified value is provided for ry but not for rx, then the user agent processes the 'rect' element with the effective value for rx as equal to ry. If neither rx nor ry has a properly specified value, then the user agent processes the 'rect' element as if no rounding had been specified, resulting in square corners. If rx is greater than half of the width of the rectangle, then the user agent processes the 'rect' element with the effective value for rx as half of the width of the rectangle. If ry is greater than half of the height of the rectangle, then the user agent processes the 'rect' element with the effective value for ry as half of the height of the rectangle. If rx or ry is negative, then the user agent processes the 'rect' element as if the given attribute had a value of zero.

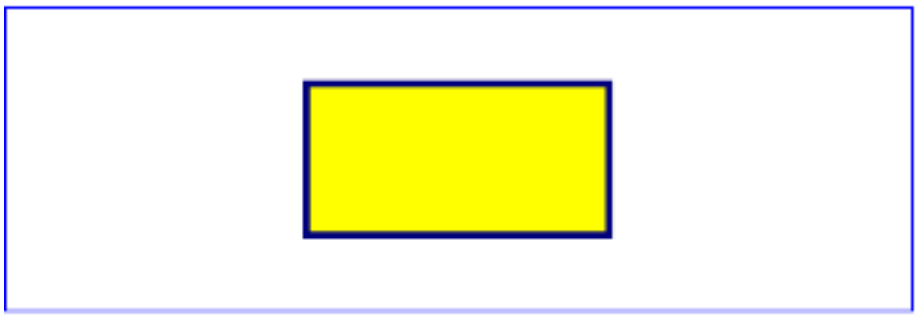
Mathematically, a 'rect' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto absolute 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 a horizontal lineto to location (*x*+*width*-*rx*,*y*), where *width* is the 'rect' element's width attribute converted to user space
- perform an arcto 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 relative arcto commands, 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 vertical *lineto* to location (*x*+*width*,*y*+*height*-*ry*), where *height* is the 'rect' element's height attribute converted to user space
- perform an arcto to coordinate (*x*+*width*-*rx*,*y*+*height*)
- perform a horizontal *lineto* to location (*x*+*rx*,*y*+*height*)
- perform an arcto to coordinate (*x*,*y*+*height*-*ry*)
- perform a vertical *lineto* to location (*x*,*y*+*ry*)

- perform an arcto to coordinate ($x+rx$,y)

Example rect01 below expresses all values in physical units (centimeters, in this case). The 'rect' element is filled with yellow and stroked with navy.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm">
  <desc>Example rect01 - rectangle expressed in physical units</desc>

  <rect x="4cm" y="1cm" width="4cm" height="2cm"
        style="fill:yellow; stroke:navy; stroke-width:0.1cm" />
</svg>
```



Example rect01

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

Example rect02 below specifies the coordinates of the two rounded rectangles in the user coordinate system established by the viewBox attribute on the 'svg' element and the transform attribute on the 'g' element. 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400">
  <desc>Example rect02 - rounded rectangles expressed in user coordinates</desc>

  <rect x="100" y="100" width="400" height="200" rx="50"
        style="fill:green;" />

  <g transform="translate(700 300); rotate(-30)">
    <rect x="0" y="0" width="400" height="200" rx="50"
          style="fill:none; stroke:purple; stroke-width:30" />
  </g>
</svg>
```

Example rect02

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

9.3 The 'circle' element

The 'circle' element defines a circle based on a center point and a radius.

```
<!ENTITY % circleExt "" >
<!ELEMENT circle (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%circleExt;)*) >
<!ATTLIST circle
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
  r CDATA #REQUIRED >
```

Attribute definitions:

cx = "<coordinate>"

The X-axis coordinate of the center of the circle.
The default value is "0".
Animatable: yes.

cy = "<coordinate>"

The Y-axis coordinate of the center of the circle.
The default value is "0".
Animatable: yes.

r = "<length>"

The radius of the circle.
Animatable: yes.

Attributes defined elsewhere:

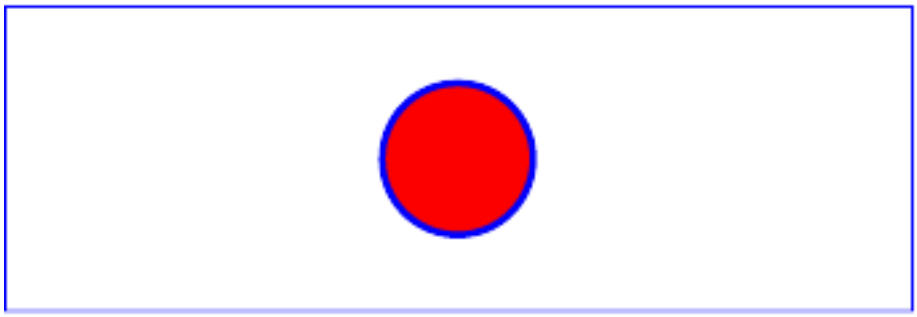
id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

Mathematically, a 'circle' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto absolute location (*cx*,*cy*+*r*), where *cx* is the value of the 'circle' element's cx attribute converted to user space, *r* is the effective value of the r attribute converted to user space and *cy* is the value of the cy attribute converted to user space
- perform an absolute arcto to location (*cx*,*cy*+*r*), where the the effective values for the r attribute on the 'circle' element converted to user space is used as the *rx* and *ry* attributes on the relative arcto commands, the *x-axis-rotation* is set to zero, the *large-arc-flag* is set to one, and the *sweep-flag* is set to one

Example circle01 below expresses all values in physical units (centimeters, in this case). The 'circle' element is filled with red and stroked with blue.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm">
  <desc>Example circle01 - circle expressed in physical units</desc>

  <circle cx="6cm" cy="2cm" r="1cm"
        style="fill:red; stroke:blue; stroke-width:0.1cm" />
</svg>
```



Example circle01

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

9.4 The 'ellipse' element

The 'ellipse' element defines an ellipse which is axis-aligned with the current user coordinate system based on a center point and two radii.

```
<!ENTITY % ellipseExt "" >
<!ELEMENT ellipse (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%ellipseExt;)*) >
<!ATTLIST ellipse
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
  rx CDATA #REQUIRED
  ry CDATA #REQUIRED >
```

Attribute definitions:

cx = "<coordinate>"

The X-axis coordinate of the center of the ellipse.
The default value is "0".
Animatable: yes.

cy = "<coordinate>"

The Y-axis coordinate of the center of the ellipse.
The default value is "0".
Animatable: yes.

rx = "<length>"

The X-axis radius of the ellipse.
Animatable: yes.

ry = "<length>"

The Y-axis radius of the ellipse.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

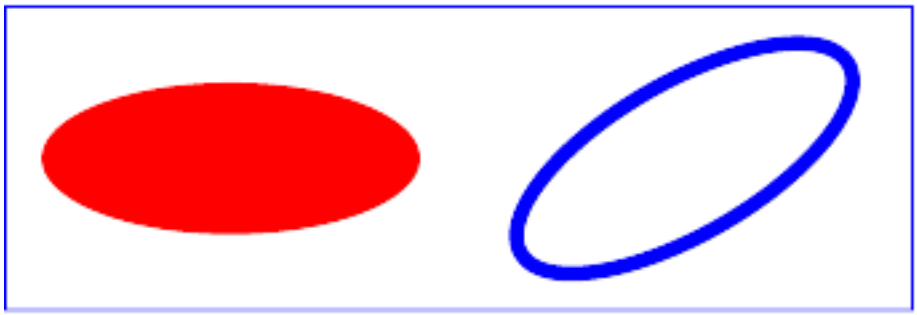
Mathematically, an 'ellipse' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto absolute location (*cx*,*cy*+*ry*), where *cx* is the value of the 'ellipse' element's cx attribute converted to user space, *ry* is the effective value of the ry attribute converted to user space and *cy* is the value of the cy attribute converted to user space
- perform an absolute arcto to location (*cx*,*cy*+*ry*), where the the effective values for the rx and ry attributes on the 'ellipse' element converted to user space are used as the *rx* and *ry* attributes on the relative arcto commands, respectively, the *x-axis-rotation* is set to zero, the *large-arc-flag* is

set to one, and the *sweep-flag* is set to one

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' 'ellipse' elements. Both ellipses uses 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400">
  <desc>Example ellipse01 - ellipses expressed in user coordinates</desc>

  <g transform="translate(300 200)">
    <ellipse rx="250" ry="100"
          style="fill:red" />
  </g>

  <ellipse transform="translate(900 200); rotate(30)"
        rx="250" ry="100"
        style="fill:none; stroke:blue; stroke-width: 20" />

</svg>
```



Example ellipse01

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

9.5 The 'line' element

The 'line' element defines a line segment that starts at one point and ends at another.

```
<!ENTITY % lineExt "" >
<!ELEMENT line (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%lineExt;)*) >
<!ATTLIST line
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x1 CDATA "0"
  y1 CDATA "0"
  x2 CDATA "0"
  y2 CDATA "0" >
```

Attribute definitions:

x1 = "<coordinate>"

The X-axis coordinate of the start of the line.
The default value is "0".
Animatable: yes.

y1 = "<coordinate>"

The Y-axis coordinate of the start of the line.
The default value is "0".
Animatable: yes.

x2 = "<coordinate>"

The X-axis coordinate of the end of the line.
The default value is "0".
Animatable: yes.

y2 = "<coordinate>"

The Y-axis coordinate of the end of the line.
The default value is "0".
Animatable: yes.

Attributes defined elsewhere:

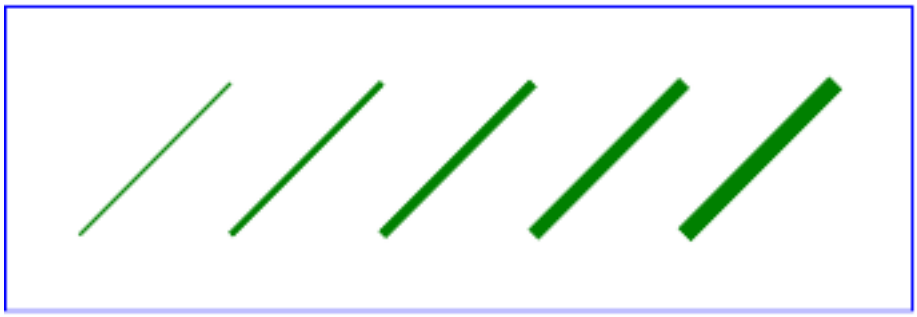
id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

Mathematically, an 'line' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto absolute location (*x1*,*y1*), where *x1* and *y1* are the values of the 'line' element's x1 and y1 attribute converted to user space, respectively
- perform an absolute lineto absolute location (*x2*,*y2*), where *x2* and *y2* are the values of the 'line' element's x2 and y2 attribute converted to user space, respectively

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400">
  <desc>Example line01 - lines expressed in user coordinates</desc>

  <g style="fill:none; stroke:green">
    <line x1="100" y1="300" x2="300" y2="100"
          style="stroke-width:5" />
    <line x1="300" y1="300" x2="500" y2="100"
          style="stroke-width:10" />
    <line x1="500" y1="300" x2="700" y2="100"
          style="stroke-width:15" />
    <line x1="700" y1="300" x2="900" y2="100"
          style="stroke-width:20" />
    <line x1="900" y1="300" x2="1100" y2="100"
          style="stroke-width:25" />
  </g>
</svg>
```



Example line01

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

9.6 The 'polyline' element

The 'polyline' element defines a set of connected straight line segments. Typically, 'polyline' elements define open shapes.

```
<!ENTITY % polylineExt "" >
<!ELEMENT polyline (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polylineExt;)*) >
<!ATTLIST polyline
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  points CDATA #REQUIRED >
```

Attribute definitions:

points = "<list-of-points>"

The points that make up the polyline. All coordinate values are in the user coordinate system. Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

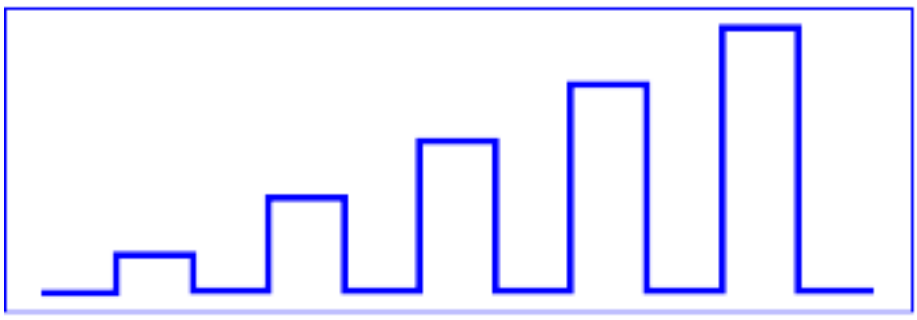
If an odd number of points is provided, then the element is in error, with the same user agent behavior as occurs with an incorrectly specified 'path' element.

Mathematically, a 'polyline' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute lineto 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400">
  <desc>Example polyline01 - increasingly larger bars</desc>

  <polyline style="fill:none; stroke:blue; stroke-width:10cm"
            points="50,375
                    150,375 150,325 250,325 250,375
                    350,375 350,250 450,250 450,375
                    550,375 550,175 650,175 650,375
                    750,375 750,100 850,100 850,375
                    950,375 950,25 1050,25 1050,375
                    1150,375" />
</svg>
```



Example polyline01

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

9.7 The 'polygon' element

The 'polygon' element defines a closed shape consisting of a set of connected straight line segments.

```
<!ENTITY % polygonExt "" >
<!ELEMENT polygon (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polygonExt;)*) >
<!ATTLIST polygon
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  points CDATA #REQUIRED >
```

Attribute definitions:

points = "<list-of-points>"

The points that make up the polygon. All coordinate values are in the user coordinate system. Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

If an odd number of points is provided, then the element is in error, with the same user agent behavior as occurs with an incorrectly specified 'path' element.

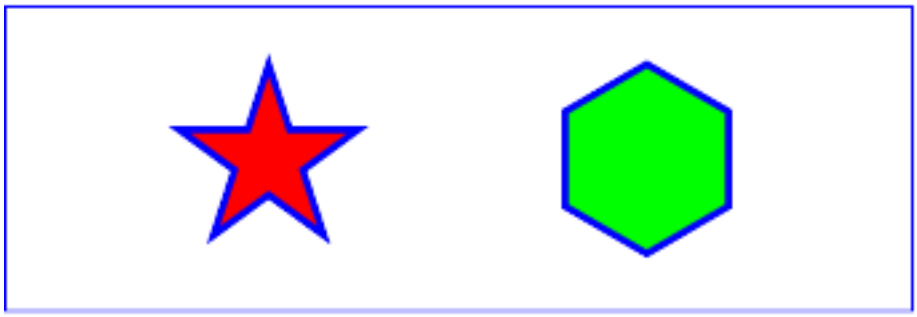
Mathematically, a 'polygon' element can be mapped to an equivalent 'path' element as follows.

- perform an absolute moveto to the first coordinate pair in the list of points
- for each subsequent coordinate pair, perform an absolute lineto 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400">
  <desc>Example polygon01 - star and hexagon</desc>

  <polygon style="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 style="fill:lime; stroke:blue; stroke-width:10"
           points="850,75  958,137.5 958,262.5
```

```
                    850,325 742,262.6 742,137.5" />
</svg>
```



Example polygon01

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

9.8 The grammar for points specifications in 'polyline' and 'polygon' elements

The following is the BNF for points specifications in 'polyline' and 'polygon' elements. The following notation is used:

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- (): grouping
- |: separates alternatives
- double quotes surround literals

```
list-of-points:
    wsp* coordinate-pairs?

coordinate-pairs:
    coordinate-pair
    | coordinate-pair comma-wsp coordinate-pairs

coordinate-pair:
    coordinate comma-wsp coordinate

coordinate:
    number-wsp

number-wsp:
    number wsp*

number:
    sign? integer-constant
    | sign? floating-point-constant

comma-wsp:
    comma? wsp*

comma:
    ","
```

```
integer-constant:
    digit-sequence

floating-point-constant:
    fractional-constant exponent?
    | digit-sequence exponent

fractional-constant:
    digit-sequence? "." digit-sequence
    | digit-sequence "."

exponent:
    ( "e" | "E" ) sign? digit-sequence

sign:
    "+" | "-"

digit-sequence:
    digit
    | digit digit-sequence

digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

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

9.9 DOM interfaces

9.9.1 Interface SVGRectElement

The SVGRectElement interface corresponds to the 'rect' element.

```
interface SVGRectElement : SVGStyledAndTransformedElement {
  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
  attribute SVGLength rx;
  attribute SVGLength ry;
};
```

9.9.2 Interface SVGCircleElement

The SVGCircleElement interface corresponds to the 'circle' element.

```
interface SVGCircleElement : SVGStyledAndTransformedElement {
  attribute SVGLength cx;
  attribute SVGLength cy;
  attribute SVGLength r;
};
```

9.9.3 Interface SVGEllipseElement

The SVGEllipseElement interface corresponds to the 'ellipse' element.

```
interface SVGEllipseElement : SVGStyledAndTransformedElement {
  attribute SVGLength cx;
  attribute SVGLength cy;
  attribute SVGLength rx;
  attribute SVGLength ry;
};
```

9.9.4 Interface SVGLineElement

The SVGLineElement interface corresponds to the 'line' element.

```
interface SVGLineElement : SVGStyledAndTransformedElement {
  attribute SVGLength x1;
  attribute SVGLength y1;
  attribute SVGLength x2;
  attribute SVGLength y2;
};
```

9.9.5 Interface SVGPointList

The SVGPointList interface is a base interface used by SVGPolylineElement and SVGPolygonElement.

```
interface SVGPointList  {
  SVGPoint     createSVGPoint();     // Returns unattached point (0,0)

  readonly attribute  unsigned long number_of_lengths;
  SVGPoint      getSVGPoint(in unsigned long index);

  // Replace all existing entries with a single entry.
  void           initialize(in SVGPoint newSVGPoint)
                                        raises(DOMException);
  void           clear(); // Clear all entries, giving an empty list
  SVGPoint      insertBefore(in SVGPoint newSVGPoint,
                                        in unsigned long index)
                                        raises(DOMException);
  SVGPoint      replace(in SVGPoint newSVGPoint,
                                    in unsigned long index)
                                    raises(DOMException);
  SVGPoint      remove(in unsigned long index)
                                   raises(DOMException);
  SVGPoint      append(in SVGPoint newSVGLength)
                                   raises(DOMException);
};
```

9.9.6 Interface SVGPolylineElement

The SVGPolylineElement interface corresponds to the 'polyline' element.

```
interface SVGPolylineElement : SVGStyledAndTransformedElement {
  attribute SVGPointList points;
};
```

9.9.7 Interface SVGPolygonElement

The SVGPolygonElement interface corresponds to the 'polygon' element.

```
interface SVGPolygonElement : SVGStyledAndTransformedElement {
  attribute SVGPointList points;
};
```

---

10 Text

Contents

10.1 Introduction

Text that is to be rendered as part of an SVG document fragment is specified using the 'text' element. The characters to be drawn are expressed as XML character data [XML10] inside the 'text' element.

SVG's 'text' elements are rendered like other graphics elements. Thus, coordinate system transformations, painting, clipping and masking features apply to 'text' elements in the same way as they apply to shapes such as paths and rectangles.

Each 'text' element causes a single string of text to be rendered. SVG performs no automatic line breaking or word wrapping. To achieve the effect of multiple lines of text:

- 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 [XHTML10] 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, right-to-left and bi-directional text (e.g., for mixing Roman scripts with Arabic or Hebrew scripts)
- when SVG fonts are used, automatic selection of the correct glyph corresponding to the current form for Arabic and Han text

(The layout rules for straight line text are described in Text layout. The layout rules for text on a path are described in Text on a path layout rules.)

Because SVG text is packaged as XML character data [XML10]:

- Text data in SVG content is readily accessible to the visually impaired (see Accessibility Support)
- In many viewing scenarios, the user will be able to search for and select text strings and copy selected text strings to the system clipboard (see Text selection)
- 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 XML characters, where each character is defined by a particular character (i.e., code point) in Unicode [UNICODE]. Fonts, on the other hand, consists of a collection of glyphs, where 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 Roman font 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.
- 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 map 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 text string. In these situations, a single Unicode character might map to one of several alternative glyphs.
- 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 only include only the glyphs needed for the given document) be available either as SVG fonts embedded within the SVG content or as web fonts posted at the same web location as the SVG content.

10.3 The 'text' element

The 'text' element defines a graphics element consisting of text. The XML [XML10] character data within the 'text' element, along with relevant attributes and properties and character-to-glyph mapping tables within the font itself, define the glyphs to be rendered. (See Characters and their corresponding glyphs.) The attributes and properties on the 'text' element indicate such things as the writing direction, font specification and painting attributes which describe how exactly to render the characters. Subsequent sections of this chapter describe the relevant text-specific attributes and properties.

Since 'text' elements are rendered using the same rendering methods as other graphics element, 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.

The 'text' renders its first character at the initial current text position, which is established by the x and y attributes. 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' text advance value (horizontal or vertical). See text layout for a description of glyph placement and glyph advance.

```
<!ENTITY % textExt "" >
<!ELEMENT text (#PCDATA|tspan|tref|textPath|altglyph|use|animate|set|animateMotion|animateColor|animateTransform
                %geExt;%textExt;)* >
<!ATTLIST text
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED >
```

Attribute definitions:

x = "<coordinate>"

The X-coordinate for the initial current text position for the text to be drawn. If the value is expressed as a simple <number> without a unit identifier (e.g., 48), then the value represents a coordinate in the current user coordinate system.
If one of the CSS unit identifiers is provided (e.g., 12pt or 10%), then the value represents a distance in viewport units relative to the origin of the user coordinate system. (See Processing rules for CSS units and percentages.) The default value is "0".
Animatable: yes.

y = "<coordinate>"

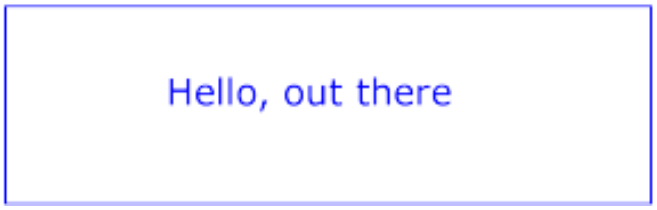
The corresponding Y-coordinate for the initial current text position. The default value is "0".
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

Example text01 below expresses all values in physical units such as centimeters and points. The 'text' element contains the text string "Hello, out there" which will be rendered onto the canvas using the Verdana font family with font size of 12 points with the glyphs filled with the color blue.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm">
  <desc>Example text01 - 'Hello, out there' in blue</desc>

  <text x="2.5cm" y="1.5cm"
        style="font-family:Verdana; font-size:16pt; fill:blue">
    Hello, out there
  </text>
</svg>
```



Example text01

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

Example text02 below expresses the x and y attributes and the 'font-size' property in the user coordinate system set up by the viewBox attribute on the 'svg' element. The 'text' element contains the text string "Text in user space."

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
  <desc>Example text02 - Text in user space</desc>

  <text x="250" y="150"
        style="font-family:Verdana; font-size:42.333; fill:blue">
    Text in user space
  </text>
</svg>
```

Example text02

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

10.4 The 'tspan' element

Within a 'text' element, text and font properties and the current text position can be adjusted with absolute or relative coordinate values by including a 'tspan' element.

```
<!ENTITY % tspanExt "" >
<!ELEMENT tspan (#PCDATA|tspan|tref|altglyph|animate|set|animateColor
                 %tspanExt;)* >
<!ATTLIST tspan
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED
  rotate CDATA #IMPLIED >
```

Attribute definitions:

x = "<coordinate>+"

If a single <coordinate> is provided, this value represents the new absolute X coordinate for the current text position for the first character within the 'tspan' element. 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 the first <n> characters within the 'tspan' element. 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 the starting X coordinate of each extra character is positioned at the X coordinate of the resulting current text position from rendering the previous character within the 'text' element.

CSS unit identifiers, such as cm, pt or %, can be provided for any <coordinate>. If a <coordinate> is provided without a unit identifier (e.g., 48), then the value represents a coordinate in the current user coordinate system. If a CSS unit identifier is provided (e.g., 12pt or 10%), then the value represents a distance in viewport units relative to the origin of the user coordinate system. (Processing rules for CSS units and percentages.) The default value is "0".

Animatable: yes.

y = "<coordinate>+"

The corresponding list of absolute Y coordinates for the characters within the 'tspan' element. The default value is "0".

Animatable: yes.

dx = "<coordinate>+"

If a single <coordinate> is provided, this value represents the new relative X coordinate for the current text position for the first character within the 'tspan' element. Thus, the current text position is shifted along the X axis of the current user coordinate system by <coordinate>. If a comma- or space-separated list of <n> <coordinate>s is provided, then the values represent new relative X coordinates for the current text position for the first <n> characters within the 'tspan' element. Thus, before each character is rendered, the current text position resulting from drawing the previous character (or, for the first character in a 'text' element, the initial current text position) is shifted along the X axis of the current user coordinate system by <coordinate>. If more <coordinate>s are provided than characters, then any extra <coordinate>s will have no effect on glyph positioning. If more characters exist than <coordinate>s, then the starting X coordinate of each extra character is positioned at the X coordinate of the resulting current text position from rendering the previous character within the 'text' element.

CSS unit identifiers, such as cm, pt or %, can be provided for any <coordinate>. If a <coordinate> is provided without a unit identifier (e.g., 48), then the value represents a length along the X axis in the current user coordinate system. If one of the CSS unit identifiers is provided (e.g., 12pt or 10%), then the value represents a distance in the viewport coordinate system. (Processing rules for CSS units and percentages.) The default value is "0".

Animatable: yes.

dy = "<coordinate>+"

The corresponding list of relative Y coordinates for the characters within the 'tspan' element. The default value is "0".
Animatable: yes.

rotate = "auto | <number>+"

A value of auto causes all characters to be oriented as specified by other text attributes without any supplemental rotation.
If a single <number> is provided, then this value represents a supplemental rotation about the current text position that will be applied to each glyph corresponding to each character within the 'tspan' element.
If a comma- or space-separated list of <number>s is provided, then the first <number> represents the supplemental rotation of the first character, the second <number> represents the supplemental rotation of 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 the extra characters will be rotated by the last <number> in the list. This supplemental rotation has no impact on the rules by which current text position is modified as glyphs get rendered.
The default value is "auto".
Animatable: yes (non-additive, 'set' and 'animate' elements only).

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, system-required, system-language.

The x, y, dx, dy and rotate on the 'tspan' element are useful in high-end typography scenarios where individual glyphs requires 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 advanced typographic control is required and micro-level positioning adjustment are necessary, 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 web font 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, rotate when they contain a list of numbers:

- Required behavior when multiple XML characters map to a single glyph (e.g., when a ligature is used) - Assume 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. For the (i+1)-th values, however, the x, y and rotate value are not applied (although the final rotate value 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.
- Relationship to right-to-left text and bi-directionality - Text is laid out in a two-step process, where any right-to-left and bi-directional 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">Roman and Arabic</span>
  ```

  and that the word "Arabic" 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 "Roman and cibarA" 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 characters will be positioned using standard left-to-right layout rules.
- Nested 'tspan' elements - The x, y, dx, dy and rotate attributes on a given 'tspan' element apply only to the character data that is directly within that 'tspan' element and do not apply to the character data within child (i.e., nested) 'tspan' elements. If the child/nested 'tspan' elements require positioning adjustments or rotation values, the child/nested 'tspan' elements need to specify x, y, dx, dy and rotate values for their own character data.

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.

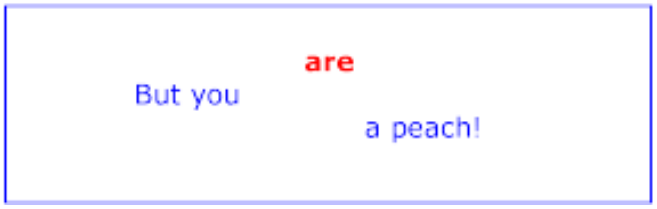
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm">
  <desc>Example tspan01 - using tspan to change visual attributes</desc>

  <g style="font-family:Verdana; font-size:12pt">
    <text x="2cm" y="1.5cm" style="fill:blue">
      You are
        <tspan style="font-weight:bold; fill:red">not</tspan>
      a banana.
    </text>
  </g>
</svg>
```

Example tspan01

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

Example tspan02 uses the dx and dy attributes on the 'tspan' 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm">
  <desc>Example tspan02 - using tspan's dx and dy attributes
        for incremental positioning adjustments</desc>

  <g style="font-family:Verdana; font-size:12pt">
    <text x="2cm" y="1.5cm" style="fill:blue">
      But you
        <tspan dx="2em" dy="-.5cm" style="font-weight:bold; fill:red">
          are
        </tspan>
        <tspan dy="1cm">
           a peach!
        </tspan>
    </text>
  </g>
</svg>
```



Example tspan02

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

Example tspan03 uses the x and y attributes on the 'tspan' 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm">
  <desc>Example tspan03 - using tspan's x and y attributes
        for multiline text and precise glyph positioning</desc>

  <g style="font-family:Verdana; font-size:12pt">
    <text style="fill:rgb(255,164,0)">
      <tspan x="3.0cm 3.5cm 4.0cm 4.5cm 5.5cm 6.0cm 6.5cm" y="1cm">
        Cute and
      </tspan>
      <tspan x="3.75cm 4.25cm 4.75cm 5.25cm 5.75cm" y="2cm">
         fuzzy
      </tspan>
    </text>
  </g>
</svg>
```

Example tspan03

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

10.5 The 'tref' element

The textual content for a 'text' can be either character data directly embedded within the 'text' element or the character data content of a referenced element, where the referencing is specified with a 'tref' element.

```
<!ENTITY % trefExt "" >
<!ELEMENT tref (animate|set|animateColor
                %trefExt;)* >
<!ATTLIST tref
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED
  rotate CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

xlink:href = "<uri>"

> A URI reference to an element/fragment within an SVG document fragment whose character data content shall be used as character data for this 'tref' element.
> Animatable: yes.

Attributes defined elsewhere:

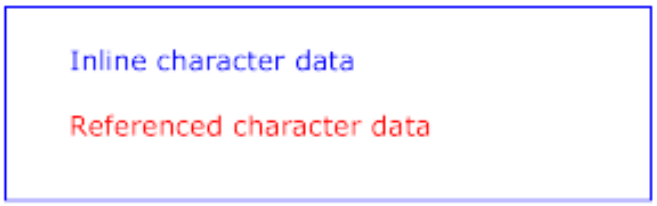
id, xml:lang, xml:space, class, style, transform, %graphicsElementEvents;, x, y, dx, dy, rotate, system-required, system-language, %xlinkAttrs;.

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 'tspan' 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm">
  <defs>
    <text id="ReferencedText">
      Referenced character data
    </text>
  </defs>
  <desc>Example tref01 - inline vs reference text content</desc>

  <text x="1cm" y="1cm" style="font-size:12pt; fill:blue">
    Inline character data
  </text>
  <text x="1cm" y="2cm" style="font-size:12pt; fill:red">
    <tref xlink:href="#ReferencedText"/>
  </text>
</svg>
```



Example tref01

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

10.6 Text layout

10.6.1 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., Roman scripts), right-to-left (e.g., Hebrew or Arabic), bi-directional (e.g., mixing Roman with 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.

Because SVG does not provide for automatic line breaks or word wrapping, internationalized text layout is simpler in SVG than in languages such as XHTML [XHTML10].

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 text advance direction, the glyph orientation relative to the text advance direction, the metrics 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 'tspan' elements.

For each glyph to be rendered, the SVG user agent determines an appropriate reference point on the glyph which will be placed exactly at the current text position. The reference point is determined based on character cell metrics in the glyph itself, the current text advance direction and the glyph orientation relative to the text advance direction For the most common uses of Roman text (i.e., 'writing-mode:lr', 'text-anchor:start', and 'glyph-anchor:baseline') the reference point in the glyph will be the intersection of left edge of the glyph character cell (or some other glyph-specific X axis coordinate indicating a left-side origin point) with the baseline of the glyph. For most cases with top-to-bottom vertical text layout, the reference point will be either a glyph-specific origin point for top-to-bottom vertical text or the intersection of the center of the glyph with its *top line* (see [CSS2] for a definition of *top line*).

The various text layout diagrams in this section use the following symbols:

- wide-cell glyph (e.g. Han) which is the *n*-th character in the text run

- narrow-cell glyph (e.g. Roman) which is the *n*-th glyph in the text run

- connected glyph (e.g. Hebrew or Arabic) 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 glyphs they represent are intended to assume when rendered in the user agent. Spacing between these characters in the diagrams is usually symbolic, unless intentionally changed to make a point.

10.6.2 Setting the primary text advance direction

The 'writing-mode' property specifies whether the primary text advance 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' and 'textPath' sub-elements. (Note that even when the primary text advance direction if left-to-right or right-to-left, some or all of the content within a given 'text' element might advance in the opposite direction because of the Unicode [UNICODE] bi-directional algorithm or because of explicit text advance overrides due to properties 'direction' and 'unicode-bidi'. For more on bi-directional text, see Relationship with bi-directionality.)

'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 primary text advance direction to left-to-right, as is common in most Roman-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 bi-directional algorithm, the text advance rules are more complex. See Relationship with bi-directionality).

rl-tb | rl

Sets the primary text advance direction to right-to-left, as is common in Arabic or Hebrew scripts.

tb-rl | tb

Sets the primary text advance direction to top-to-bottom, as is common in Asian scripts. 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.

10.6.3 Glyph orientation with a text run

In some cases, it is required to alter the orientation of a sequence of characters relative to the primary text advance direction. The requirement is particularly applicable to vertical layouts of East Asian documents, where sometimes half-width Roman text is to be displayed horizontally and other times vertically.

Two properties control the glyph orientation relative to the primary text advance direction. 'glyph-orientation-vertical' controls glyph orientation when the primary text advance direction is vertical. 'glyph-orientation-horizontal' controls glyph orientation when the primary text advance direction is horizontal.

'glyph-orientation-vertical'

Value:	<angle> | auto | inherit
Initial:	auto
Applies to:	'text', 'tspan', 'tref', 'textPath' elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

<angle>

The value of the angle is a <integer> restricted to the range of -360 to +360 in 90-degree increments.
A value of 0 indicates that all glyphs are oriented with the bottom of the glyphs toward the primary text advance direction, resulting in glyphs which are stacked vertically on top of each other. A value of 90 indicates a rotation of 90 degrees clockwise from the "0" orientation. Negative angle values are computed modulo 360; thus, a value of -90 is equivalent to a value of 270.

auto

The glyph orientation relative to the primary text advance direction is determined automatically based on the Unicode character number of the rendered glyph.

Full-width ideographic and full-width Roman glyphs (excluding ideographic punctuation) are oriented as if an <angle> of "0" had been specified (i.e., glyphs are oriented with the bottom of the glyphs toward the primary text advance direction, resulting in glyphs which are stacked vertically on top of each other).

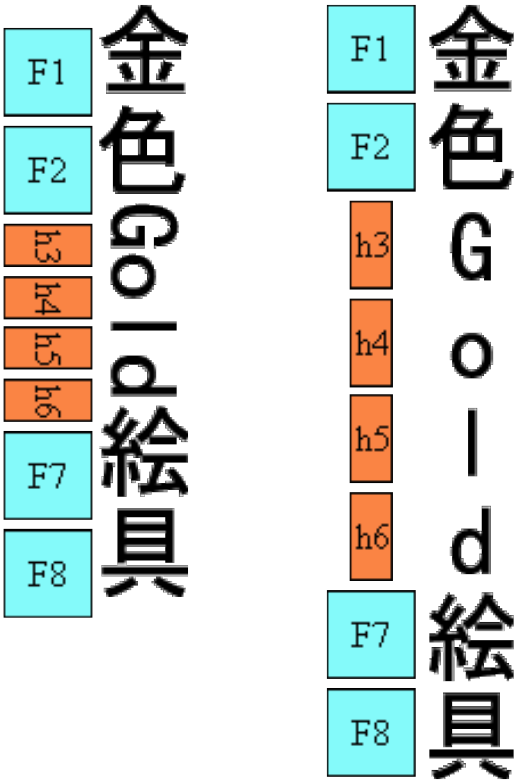
Ideographic punctuation and other ideographic characters having alternate horizontal and vertical forms shall use the vertical form of the glyph.

Text which is not full-width will be set as if an <angle> of "90" had been specified; thus, half-width Roman text will be rotated 90 degree clockwise versus full-width ideographic and full-width Roman text.

Note that a value of auto will generally produce the expected results in common uses of mixing Japanese with European characters; however, the exact algorithms are based on complex interactions between many factors, including font design, and thus different algorithms might be employed in different processing environments. For precise control, specify explicit <angle> values.

The glyph orientation affects the amount that the current text position advances as each glyph is rendered. When the primary text advance 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 diagrams below illustrate different uses of 'glyph-orientation-vertical'. The diagram on the left shows the result of the mixing of full-width ideographic characters with half-width Roman characters when 'glyph-orientation-vertical' for the Roman characters is either auto or 90. The diagram on the right show the result of mixing full-width ideographic characters with half-width Roman characters when Roman characters are specified to have a 'glyph-orientation-vertical' of 0.



'glyph-orientation-horizontal'

Value:	<angle> | inherit
Initial:	0
Applies to:	'text', 'tspan', 'tref', 'textPath' elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

<angle>

The value of the angle is a <integer> restricted to the range of -360 to +360 in 90-degree increments.
A value of 0 indicates that all glyphs are oriented with the right edge of the glyphs toward the primary text advance direction, resulting in glyphs which are positioned side by side. A value of 90 indicates an orientation of 90 degrees clockwise from the "0" orientation. Negative angle values are computed modulo 360; thus, a value of -90 is equivalent to a value of 270.

The glyph orientation affects the amount that the current text position advances as each glyph is rendered. When the primary text advance 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-vertical' 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.

10.6.4 Relationship with bi-directionality

The characters in certain scripts are written from right to left. In some documents, in particular those written with the Arabic or Hebrew script, and in some mixed-language contexts, text in a single line may appear with mixed directionality. This phenomenon is called bidirectionality, or "bidi" for short.

The Unicode standard ([UNICODE], section 3.11) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. The SVG user agent applies this bidirectional algorithm when determining the layout of characters within a 'text' element. The 'direction' and 'unicode-bidi' properties allow authors to override the inherent directionality of the content characters and thus explicitly control how the elements and attributes of a document language map to this algorithm. These two properties are only applicable when the primary text advance direction is horizontal.

Because the directionality of a text depends on the structure and semantics of the document language, in most cases these properties will be used only by designers of document type descriptions (DTDs) or authors of special documents.

A more complete discussion of bi-directionality can be found in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].

The processing model for right-to-left or bi-directional horizontal text is as follows. The user agent processes the characters which are provided in lexical order and re-orders the characters after processing the Unicode bi-directional algorithm and properties 'direction' and 'unicode-bidi', resulting in a potentially re-ordered list of characters which are now in left-to-right rendering order. Simultaneous with re-ordering of the characters, the x, y, 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. Similarly, text selection occurs on the re-ordered text (i.e., based on visual layout rather than lexical layout).

When included in a 'text' element whose primary text advance direction is vertical, Arabic text has a default orientation where the glyphs are rotated 90 degrees counter-clockwise from standard vertically-oriented glyphs, making the default orientation of the Arabic glyphs the same as for half-width Roman glyphs.

'direction'

Value:	ltr | rtl | inherit
Initial:	ltr
Applies to:	all elements, but see prose
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

This property specifies the base writing direction of text and the direction of embeddings and overrides (see 'unicode=bidi') for the Unicode bidirectional algorithm. For the 'direction' property to have any effect, the 'unicode=bidi' property's value must be 'embed' or 'override'. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for the specification for this property.

The 'direction' property applies only to text whose glyph orientation has the right edge of the glyphs oriented in the same direction as the primary text advance direction, which includes the usual case of horizontally-oriented Roman or Arabic text and the case of half-width Roman or Arabic characters rotated 90 degrees clockwise relative to a top-to-bottom primary text advance direction.

'unicode-bidi'

Value:	normal | embed | bidi-override | inherit
Initial:	normal
Applies to:	all elements, but see prose
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	no

Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for the specification for this property.

10.7 Text alignment properties

Each text element establishes an initial current text position. The following properties are used to align the contents of a 'text' element relative to the current text position.

'text-anchor'

Value:	start | middle | end | inherit
Initial:	start
Applies to:	'text' elements
Inherited:	yes
Percentages:	N/A

Media: visual
Animatable: yes

This property, which applies only to 'text' elements and is ignored for elements 'tspan', 'tref' and 'textPath', describes how the characters within a 'text' element are aligned relative to the initial current text position for the 'text' element. Values have the following meanings:

start

The rendered characters are aligned such that the start of the text string is at the initial current text position. For standard Roman text, this is comparable to left alignment. For Asian text with a vertical primary text direction, this is comparable to top alignment.

middle

The rendered characters are al current text position. For standard Roman text, this is comparable to center alignment.

end

The rendered characters are aligned such that the end of the text string is at the initial current text position. For standard Roman text, this is comparable to right alignment.

'glyph-anchor'

Value: text-top | topline | hanging | mathline | centerline | baseline | ideographic | text-bottom | inherit
Initial: baseline
Applies to: 'text', 'tspan', 'tref' and 'textPath' elements
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes

This property, which only applies to glyphs rendered horizontally (i.e., bottom of the glyph is parallel to the primary text advance direction), describes the vertical alignment of glyphs relative to the current text position. Values have the following meanings:

text-top

Align the glyph vertically relative such that the top of the glyph is aligned with the current text position. (Refer to the discussion of text-top under SVG fonts and the "text-top" value for the 'vertical-align' property in [CSS2].)

topline

Align the glyph vertically such that the "topline" of the glyph is aligned with the current text position. (Refer to the discussion of topline under SVG fonts and the "topline" font descriptor described in [CSS2].)

hanging

Align the glyph vertically such that the "hanging" position of the glyph is aligned with the current text position. (Refer to the discussion of hanging under SVG fonts.)

mathline

Align the glyph vertically such that the "mathline" of the glyph is aligned with the current text position. (Refer to the discussion of mathline under SVG fonts and the "mathline" font descriptor described in [CSS2].)

centerline

Align the glyph vertically such that the "centerline" of the glyph is aligned with the current text position. (Refer to the discussion of centerline under SVG fonts and the "centerline" font descriptor described in [CSS2].)

baseline

Align the glyph vertically such that the "baseline" of the glyph is aligned with the current text position. (Refer to the discussion of baseline under SVG fonts and the "baseline" font descriptor described in [CSS2].)

ideographic

Align the glyph vertically such that the "ideographic" position of the glyph is aligned with the current text position. (Refer to the discussion of ideographic under SVG fonts.)

text-bottom

Align the glyph vertically relative such that the bottom of the glyph is aligned with the current text position. (Refer to the discussion of text-bottom under SVG fonts and the "text-bottom" value for the 'vertical-align' property in [CSS2].)

'baseline-shift'

Value:	super | sub | <length> | inherit
Initial:	0
Applies to:	'text', 'tspan', 'tref' and 'textPath' elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes (non-additive, 'set' and 'animate' elements only)

This property, which only applies to glyphs rendered horizontally (i.e., bottom of the glyph is parallel to the primary text advance direction), provides for vertical adjustment of the current text position. Property values are cumulative; thus, the 'baseline-shift' value for the current element is added to all of the 'baseline-shift' values for its ancestors up to its parent 'text' element. Values have the following meanings:

super

Shift the text upward to the appropriate position for superscripts.

sub

Shift the text downward to the appropriate position for subscripts.

<length>

Shift the text vertically by the given distance.

10.8 Font selection properties

SVG uses the following font specification properties from CSS2. Any SVG-specific notes about these properties are contained in the descriptions below.

'font-family'

Value:	[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>] | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

This property 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'font-style'

Value:	normal | italic | oblique | inherit
Initial:	normal
Applies to:	all 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'font-variant'

Value:	normal | small-caps | inherit

Initial: normal
Applies to: all 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'font-weight'

Value: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
Initial: normal
Applies to: all 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'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: all 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'font-size'

Value: <absolute-size> | <relative-size> | <length> | <percentage> | inherit
Initial: medium
Applies to: all 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 CSS 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 processing rules for CSS units and percentages. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'font-size-adjust'

Value: <number> | none | inherit
Initial: none

Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes (non-additive, 'set' and 'animate' elements only)

This property allows authors to specify an aspect value for an element that will preserve the x-height of the first choice font in a substitute font. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'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: all elements
Inherited: yes
Percentages: allowed on 'font-size' and 'line-height'
Media: visual
Animatable: yes (non-additive, 'set' and 'animate' elements only)

Shorthand property for setting 'font-style', 'font-variant', 'font-weight', 'font-size', 'line-height' and 'font-family'. The 'line-height' property has no visual effect in SVG. 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.

Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

10.9 Spacing properties

'letter-spacing'

Value: normal | <length> | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes

This property specifies spacing behavior between text characters. 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 CSS 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 processing rules for CSS units and percentages. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

'word-spacing'

Value: normal | <length> | inherit
Initial: normal
Applies to: all 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 CSS 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 processing rules for CSS units and percentages. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

10.10 Text decoration

'text-decoration'

Value:	none | [underline || overline || line-through || blink] | inherit
Initial:	none
Applies to:	all 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. Refer to the "Cascading Style Sheets (CSS) level 2" specification [CSS2] for more information about this property.

10.11 Text on a path

10.11.1 Introduction to text on a path

In addition to text drawn in a straight line, SVG also includes the ability to place text along the shape of a 'path' element. To specify that a block of text is to be rendered along the shape of a 'path', include the given text within a 'textPath' element which includes an xlink:href attribute with a URI reference to a 'path' element.

10.11.2 The 'textPath' element

```
<!ENTITY % textPathExt "" >
<!ELEMENT textPath (#PCDATA|tspan|tref|altglyph|animate|set|animateColor
                %textPathExt;)* >
<!ATTLIST textPath
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  startOffset CDATA "0"
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

startOffset = "<length> | <percentage>"

> 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> without 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'.
> Animatable: yes.

xlink:href = "<uri>"

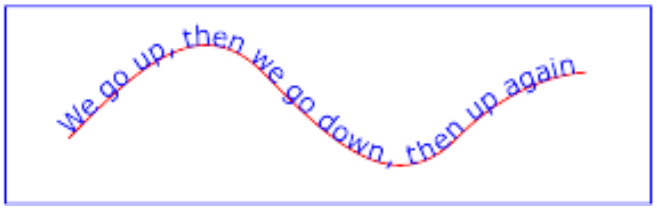
A URI reference to the 'path' element onto which the glyphs will be rendered. If <uri> is an invalid reference (e.g., no such element exists, or the referenced element is not a 'path'), then the 'textPath' element is in error and its entire contents shall not be rendered by the user agent.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, %graphicsElementEvents;, system-required, system-language, %xlinkAttrs;.

Example toap01 provides a simple example of text on a path:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
  <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" style="stroke:red" />
  <text style="font-family:Verdana; font-size:42.3333; fill:blue">
    <textPath xlink:href="#MyPath">
      We go up, then we go down, then up again
    </textPath>
  </text>
</svg>
```



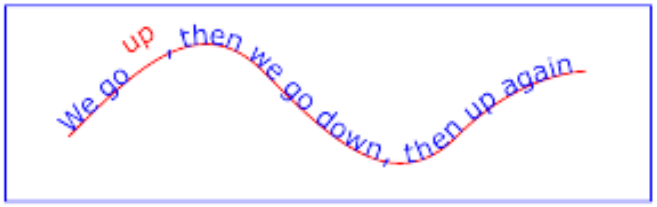
Example toap01

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

Example toap02 shows how 'tspan' elements can be included within 'textPath' elements to adjust styling attributes and adjust the current text position before rendering a particular glyph. The first occurrence of the word "up" is filled with the color red. Attribute dy is used to lift the word "up" from the baseline.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
  <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" style="fill:none; stroke:red" />
  <text style="font-family:Verdana; font-size:42.3333; fill:blue">
    <textPath xlink:href="#MyPath">
      We go
      <tspan dy="30" style="fill:red">
        up
      </tspan>
      <tspan dy="-30">
        ,
      </tspan>
```

```
      then we go down, then up again
    </textPath>
  </text>
</svg>
```

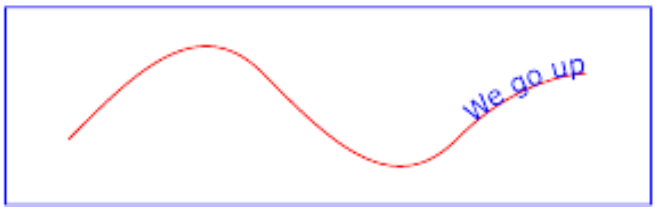


Example toap02

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

Example toap03 demonstrates the use of the startOffset attribute on the 'textPath' element to specify the start position of the text string as a particular position along the path. Notice that glyphs that fall off the end of the path are not rendered (see text on a path layout rules).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
  <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" style="fill:none; stroke:red" />
  <text style="font-family:Verdana; font-size:42.3333; fill:blue">
    <textPath xlink:href="#MyPath" startOffset="80%">
      We go up, then we go down, then up again
    </textPath>
  </text>
</svg>
```



Example toap03

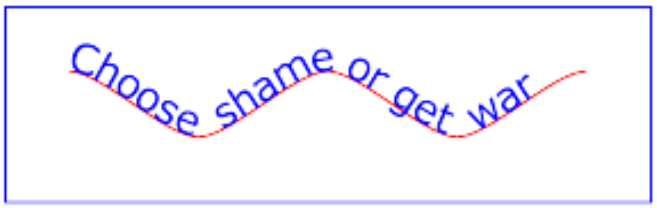
View this example as SVG (SVG-enabled browsers only)

10.11.3 Text on a path layout rules

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
  <defs>
    <path id="MyPath"
          d="M 100 100
             C 150 100 250 200 300 200
             C 350 200 450 100 500 100
             C 550 100 650 200 700 200
```

```
            C 750 200 850 100 900 100" />
  </defs>
  <desc>Example toap04 = text on a path layout rules</desc>

  <use xlink:href="#MyPath" style="fill:none; stroke:red" />
  <text style="font-family:Verdana; font-size:63.5; fill:blue">
    <textPath xlink:href="#MyPath">
      Choose shame or get war
    </textPath>
  </text>
</svg>
```



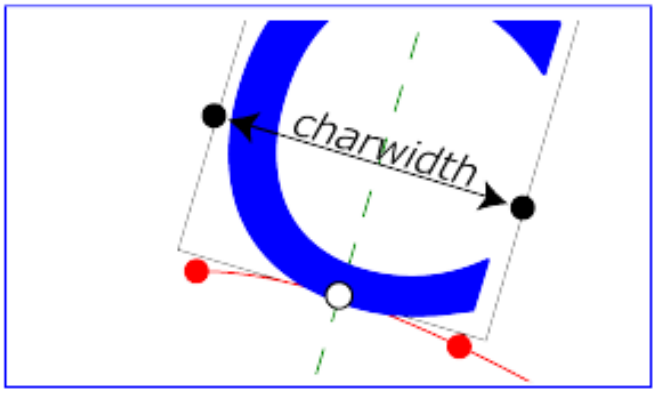
Example toap04

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

The following picture does an initial zoom in on the first glyph in the 'text' element.

The small dot above shows the point at which the glyph is attached to the path. The box around the glyph shows the glyph is rotated such that its horizontal axis is parallel to the tangent of the curve at the point at which the glyph is attached to the path. The box also shows the glyph's charwidth (i.e., the amount which the current text position advances horizontally when the glyph is drawn using horizontal text layout).

The next picture zooms in further to demonstrate the detailed layout rules.



For horizontal text layout along a path, the layout rules are as follows:

- Determine the startpoint-on-the-path for the first glyph using attribute startOffset and, if present, the dx attribute 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 'glyph-anchor' and any specified values for attribute dy on a 'tspan' element. In the example above, the 'glyph-anchor' property is unspecified, so the initial value of 'glyph-anchor: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 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 the 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.

For 'tspan' elements that are children of 'textPath' elements, x and y attributes on 'tspan' elements have no effect on text layout.

Vertical, right-to-left and bi-directional text layout rules also apply to text on a path. Conceptually, 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.

10.12 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 glyph is used than the glyph which normally corresponds to the given character data. Also, The W3C Character Model [CHARMOD] encourages creators of XML to normalize character data to facilitate meaningful exchange of character data and to promote correct comparisons between character strings. This normalization potentially loses some information about which specific glyph is required to achieve a particular visual result.

The 'altGlyph' element provides control over the glyphs used to render particular character data.

```
<!ENTITY % altGlyphExt "" >
<!ELEMENT altGlyph (#PCDATA %altGlyphExt;)* >
<!ATTLIST altGlyph
  id ID #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

xlink:href = "<uri>"

A URI reference either to a 'glyph' element in an SVG document fragment or to a 'altGlyphDef' element. If the reference is to a 'glyph' element, then that glyph is rendered instead of the character(s) that are inside of the 'altGlyph' element. If the reference is to a 'altGlyphDef' element, then if an appropriate alternate glyph is located from processing the 'altGlyphDef' element, then that alternate glyph is rendered the that glyph is rendered instead of the character(s) that are inside of the 'altGlyph' element. If the reference does not result in successful identification of an alternate glyph to use, then the character(s) that are inside of the 'altGlyph' element are rendered.
Animatable: no.

Attributes defined elsewhere:

id, %xlinkAttrs;.

The 'altGlyphDef' element, which can only appear as a child of a 'defs' element, defines a list of possible glyph substitutions which can be referenced from an 'altGlyph' element. Each possible glyph substitution is defined by a 'glyphSub' child element. The first 'glyphSub' element which locates a substitute glyph will be applied.

```
<!ENTITY % altGlyphDefExt "" >
<!ELEMENT altGlyphDef (glyphSub %altGlyphDefExt;)* >
<!ATTLIST altglyphDef
  id ID #IMPLIED >
```

Attributes defined elsewhere:

id, %xlinkAttrs;.

The 'glyphSub' element defines a possible glyph substitution, consisting of a font name, a glyph identifier and a font format.

```
<!ELEMENT glyphSub EMPTY >
<!ATTLIST glyphSub
  id ID #IMPLIED
  font CDATA #REQUIRED
  glyphRef CDATA #REQUIRED
  format CDATA #REQUIRED >
```

Attribute definitions:

font-family = "<string>"

The identifier for a single font which might contain the substitute glyph. The <string> can contain any single font family name value as is allowed in [CSS2].

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 the [CSS2] specification (e.g., *TrueDoc™ Portable Font Resource* or *Embedded OpenType*), then the <string> must contain the corresponding font format string defined in the [CSS2] specification (e.g., *truedoc-pfr* or *embedded-opentype*).

Animatable: no.

Attributes defined elsewhere:

id.

10.13 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. **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. First, it will remove all carriage return and linefeed 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. It will convert all carriage returns, linefeeds 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 examples illustrate that line indentation can be important when using `xml:space="default"`. The fragments below show two pairs of equivalent 'text' elements. Each pair consists of two equivalent 'text' elements, with the

first 'text' element using xml:space='default' and the second using xml:space='preserve'. 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 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 not intermediate space characters).

The xml:space attribute is:

Animatable: no.

10.14 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 character 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 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 bi-directional text, the user agent must support text selection in lexical order, which will result in discontinuous highlighting of glyphs due to the bi-directional reordering of characters. User agents can provide an alternative ability to select bi-directional text in visual rendering order (i.e., after bi-directional text layout algorithms have been applied), with the result that selected character data might be discontinous lexically. In this case, if the user requests that bi-directional 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.

10.15 DOM interfaces

10.15.1 Interface SVGTextContentElement

The SVGTextContentElement interface is inherited by various text-related interfaces, such as SVGTextElement, SVGTSpanElement, SVGTRefElement and SVGTextpathElement.

```
interface SVGTextContentElement : SVGStyledElement {

  long            getNumberOfChars();   // Number of characters in 'text' element
  float           getLength(); // From start of first char to end of last char
  float           getSubStringLength(in unsigned long charnum, in unsigned long nchars);
  SVGPoint        getStartPositionOfChar(in unsigned long charnum); // 0-based indexing???
  SVGPoint        getEndPositionOfChar(in unsigned long charnum); // 0-based indexing???
  SVGRect         getExtentOfChar(in unsigned long charnum); // 0-based indexing???
  float           getRotationOfChar(in unsigned long charnum); // 0-based indexing???
  long            getCharNumAtPosition(in SVGPoint); // Returns -1 is no char found
  void            selectSubString(in unsigned long charnum, in unsigned long nchars);
};
```

10.15.2 Interface SVGTextElement

The SVGTextElement interface corresponds to the 'text' element.

```
interface SVGTextElement : SVGTextContentElement {
  // SVGTransformedElement attributes
  readonly attribute  SVGElement       viewportElement; // element that established current viewport
           attribute  SVGTransformList transform;

  SVGRect      getBBox(); // tight bounding box on geometry of all contained
                          // graphics elements, in userspace.
                          // Doesn't take into account stroke-width or filter effects, for example

  SVGMatrix    getNearestCTM(); // returns CTM (userspace to [nearest 'svg'] viewport transform matrix)
  SVGMatrix    getNearestCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)
  SVGMatrix    getFurthestCTM(); // returns CTM (userspace to [outermost 'svg'] viewport transform matrix)
  SVGMatrix    getFurthestCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)
  SVGMatrix    getScreenCTM(); // returns CTM (userspace to screen units transform matrix)
  SVGMatrix    getScreenCTMInverse()
               raises(SVGException); // returns inverse matrix (SVG_MATRIX_NOT_INVERTABLE)

  // Easy access to user units
  attribute SVGLength x;
  attribute SVGLength y;
};
```

10.15.3 Interface SVGTextPositioningElement

The SVGTextPositioningElement interface is inherited by text-related interfaces: SVGTSpanElement, SVGTRefElement and SVGTextpathElement.

```
interface SVGTextPositioningElement : SVGTextContentElement {
  attribute SVGLengthList x;
  attribute SVGLengthList y;
  attribute SVGLengthList dx;
  attribute SVGLengthList dy;
  attribute SVGLengthList rotate;
};
```

10.15.4 Interface SVGTSpanElement

The SVGTSpanElement interface corresponds to the 'tspan' element.

```
interface SVGTSpanElement : SVGTextPositioningElement {
};
```

10.15.5 Interface SVGTRefElement

The SVGTRefElement interface corresponds to the 'tref' element.

```
interface SVGTRefElement : SVGTextPositioningElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
};
```

10.15.6 Interface SVGTextpathElement

The SVGTextpathElement interface corresponds to the 'textPath' element.

```
interface SVGTextpathElement : SVGTextPositioningElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
  attribute SVGLength startOffset;
};
```

10.15.7 Interface SVGAltGlyphElement

The SVGAltGlyphElement interface corresponds to the 'altGlyph' element.

```
interface SVGAltGlyphElement : SVGTextContentElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
};
```

10.15.8 Interface SVGAltGlyphDefElement

The SVGAltGlyphDefElement interface corresponds to the 'altGlyphDef' element.

```
interface SVGAltGlyphDefElement : SVGElement {
};
```

10.15.9 Interface SVGSVGGlyphSubElement

The SVGSVGGlyphSubElement interface corresponds to the 'SVGGlyphSub' element.

```
interface SVGGlyphSub : SVGElement {
  attribute DOMString fontFamily;
  attribute DOMString glyphRef;
  attribute DOMString format;
};
```

---

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**. Gradients and patterns are just specific types of paint servers. For example, first you can define a linear gradient by including a 'linearGradient' element within a 'defs', assign an ID to that 'linearGradient' element, and then reference that ID in a 'fill' or 'stroke' property:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Linear gradient example
  </desc>
  <g>
    <defs>
      <linearGradient id="MyGradient">
        <stop offset="0%" style="color:#F60"/>
        <stop offset="70%" style="color:#FF6"/>
      </linearGradient>
    </defs>
    <rect style="fill: url(#MyGradient)" width="20" height="15.8"/>
  </g>
</svg>
```

Download this example

11.2 Specifying paint

Properties 'fill' and 'stroke' take on a value of type <paint>, which is specified as follows:

```
<paint>:    none |
            currentColor |
            <color> [icc-color(<name>,<icccolorvalue>[,<icccolorvalue>]*)] |
            <uri>
              [ none |
                currentColor |
                <color> [icc-color(<name>,<icccolorvalue>[,<icccolorvalue>]*)]] |
            inherit
```

none

Indicates that the object has no fill (i.e., the interior is transparent).

currentColor

Indicates that the object is. filled with the color specified by the 'color' property. This mechanism is provided to facilitate sharing of color attributes between parent grammars such as other (non-SVG) XML. This mechanism allows you to define a style in your HTML which sets the 'color' property and then pass that style to the SVG user agent so that your SVG text will draw in the same color.

<color>
[icc-color(<name>,<icccolorvalue>[,<icccolorvalue>]*)]

<color> is the explicit color (in the sRGB [SRGB] color space) to be used to fill the current object. SVG supports all of CSS2's <color> specifications. If an optional ICC color specification is provided, then the user agent searches the color profile description database for an @color-profile entry whose name descriptor matches <name> and uses the last matching entry that is found. (If no match is found, then the ICC color specification is ignored.) The list of **<icccolorvalue>**'s is a set of ICC-profile-specific color values, expressed as <number>s. On platforms which support ICC-based color management, the **icc-color** gets precedence over the <color> (which is in the sRGB color space). Percentages are not allowed on **<icccolorvalue>**'s. For more on ICC-based colors, refer to Color.

<uri>

[none |
currentColor |
<color> [icc-color(<name>,<icccolorvalue>[,<icccolorvalue>]*)]]

The <uri> is how you identify a fancy paint style such as a gradient, a pattern or a custom paint from extensibility. The <uri> provides. the ID of the paint server (e.g., a gradient or a pattern) to be used to paint the current object. If the URI reference is not valid (e.g., it points to an object that doesn't exist or the object is not a valid paint server), then the paint method following the <uri> (i.e., **none | currentColor | <color> [icc-color(<name>,<icccolorvalue>[,<icccolorvalue>]*)]| inherit**) is used if provided; otherwise, the document is in error (see Error processing).

11.3 Fill Properties

'fill'

Value:	<paint> (See Specifying paint)
Initial:	currentColor
Applies to:	all elements
Inherited:	see Inheritance of Painting Properties below
Percentages:	N/A
Media:	visual
Animatable:	yes

Note that graphical objects that are not closed (e.g., a 'path' without a closepath at the end or a 'polyline') still can be filled. The fill operation automatically closes all open subpaths by connecting the last point of the subpath with the first point of the subpath before painting the fill.

'fill-rule'

Value:	evenodd | nonzero | inherit
Initial:	evenodd
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

evenodd

nonzero

'fill-opacity'

Value:	<opacity-value> | inherit
Initial:	100%
Applies to:	all elements
Inherited:	yes
Percentages:	Allowed
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. If a <number> is provided, any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. If a percentage is provided, any value outside the range of 0% to 100% 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:	all elements
Inherited:	see Inheritance of Painting Properties below
Percentages:	N/A
Media:	visual
Animatable:	yes

'stroke-width'

Value:	<width> | inherit
Initial:	1
Applies to:	all elements
Inherited:	yes
Percentages:	Yes
Media:	visual
Animatable:	yes

<width>

The width of the stroke on the current object, expressed as a <length>. If a percentage is used, the <width> is expressed as a percentage of the current viewport
(See Processing rules for CSS units and percentages.)

'stroke-linecap'

Value:	butt | round | square | inherit
Initial:	butt
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

'stroke-linecap' specifies the shape to be used at the end of open subpaths when they are stroked.

butt

See drawing below.

round

See drawing below.

square

See drawing below.

'stroke-linejoin'

Value:	miter | round | bevel | inherit
Initial:	miter
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

'stroke-linejoin' specifies the shape to be used at the corners of paths (or other vector shapes) that are stroked. when they are stroked.

miter

See drawing below.

round

See drawing below.

bevel

See drawing below.

'stroke-miterlimit'

Value:	<miterlimit> | inherit
Initial:	8
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

When two line segments meet at a sharp angle and **miter** joins have been specified for 'stroke-linejoin', it is possible for the miter to extend far beyond the thickness of the line stroking the path. The 'stroke-miterlimit' imposes a limit on the ratio of the miter length to the 'stroke-linewidth'.

<miterlimit>

The limit on the ratio of the miter length to the 'stroke-linewidth'. The value of **<miterlimit>** must be a number greater than or equal to 1.

'stroke-dasharray'

Value:	none | <dasharray> | inherit
Initial:	none
Applies to:	all elements

Inherited: yes
Percentages: Yes. See below.
Media: visual
Animatable: yes

'stroke-dasharray' controls the pattern of dashes and gaps used to stroke paths. **<dasharray>** contains a list of space- or comma-separated numbers that specify the lengths of alternating dashes and gaps in user units. 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 space- or comma-separated <length>'s which can be in user units or in any of the CSS units, including percentages. A percentage represents a distance as a percentage of the current viewport. (See Processing rules for CSS units and percentages.)

'stroke-dashoffset'

Value: <dashoffset> | inherit
Initial: 0
Applies to: all elements
Inherited: yes
Percentages: Yes. See below.
Media: visual
Animatable: yes

'stroke-dashoffset' specifies the distance into the dash pattern to start the dash.

<dashoffset>

A <length>. If a percentage is used, the <width> is expressed as a percentage of the current viewport (See Processing rules for CSS units and percentages.)

'stroke-opacity'

Value: <opacity-value> | inherit
Initial: 100%
Applies to: all elements
Inherited: yes
Percentages: Allowed
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. If a <number> is provided, any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. If a percentage is provided, any value outside the range of 0% to 100% will be clamped to this range. (See Clamping values which are restricted to a particular range

Related properties: 'fill-opacity' and 'opacity'.

11.5 Markers

11.5.1 Introduction

To use a marker symbol for arrowheads or polymarkers, you need to define a **'marker'** element which defines the marker symbol and then refer to that 'marker' element using the various marker properties (i.e., 'marker-start', 'marker-end', 'marker-mid' or 'marker') on the given 'path' element or vector graphic shape. Here is an example which draws a triangular marker symbol that is drawn as an arrowhead at the end of a path:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="4in"
     viewBox="0 0 4000 4000" >
  <defs>
    <marker id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerWidth="1.25" markerHeight="1.75"
      orient="auto">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>
  <desc>Placing an arrowhead at the end of a path.
  </desc>
  <path d="M 1000 1000 L 2000 1000 L 3000 2000"
        style="fill:none; stroke:black; stroke-width:100;
        marker-end:url(#Triangle)" />
</svg>
```

Download this example

11.5.2 The 'marker' element

The **'marker'** element defines the graphics that is to be used for drawing arrowheads or polymarkers on a given 'path' element or vector graphic shape.

```
<!ENTITY % markerExt "" >
<!ELEMENT marker (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%markerExt;)*) >
<!ATTLIST marker
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  markerUnits (stroke-width | userSpace | userSpaceOnUse) "stroke-width"
  markerWidth  CDATA "3"
  markerHeight CDATA "3"
  orient CDATA "0" >
```

Attribute definitions:

markerUnits = "strokeWidth | userSpace | userSpaceOnUse"

markerUnits indicates how to interpret the values of **markerWidth** and **markerHeight** (described as

follows).
If **markerUnits="stroke-width"**, then **markerWidth** and **markerHeight** represent scale factors relative to the stroke width in place for graphic object referencing the marker.
If **markerUnits="userSpace"**, then **markerWidth** and **markerHeight** represent values in the user coordinate system in place for the graphic object referencing the marker.
If **markerUnits="userSpaceOnUse"**, then **markerWidth** and **markerHeight** represent values in the current user coordinate system in place at the time when the 'marker' element is referenced (i.e., the user coordinate system for the element referencing the 'marker' element via the 'marker', 'marker-start', 'marker-mid' or 'marker-end' property). represent values in the user coordinate system in place for the graphic object referencing the marker.
Animatable: yes.

markerWidth = "<length>"

Represents the width of the temporary viewport that is to be created when drawing the marker. Default value is "3".
Animatable: yes.

markerHeight = "<length>"

Represents the height of the temporary viewport that is to be created when drawing the marker. Default value is "3".
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 in a direction that is the average of the ending direction of path segment going into the vertex and the starting direction of the path segment going out of the vertex. (Refer to 'path' element implementation notes for a more thorough discussion directionality of path segments.) A value of *<angle>* represents a particular orient 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. The default value is an angle of zero.
Animatable: yes (non-additive, 'set' and 'animate' elements only).

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, refX, refY, viewBox, preserveAspectRatio.

Markers are drawn such that their reference point (i.e., attributes **ref-x** and **ref-y**) is positioned at the given vertex.

11.5.3 Marker properties

'marker-start' defines the arrowhead or polymarker that shall be drawn at the first vertex of the given **'path'** element or vector graphic 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).

'marker-start', 'marker-end', marker-mid'

Value:	none | inherit | <uri>
Initial:	none
Applies to:	all elements
Inherited:	see Inheritance of Painting Properties below
Percentages:	N/A
Media:	visual
Animatable:	yes

none

Indicates that no marker symbol shall be drawn at the given vertex (vertices).

<uri>

The <uri> is a URI reference to the ID of a 'marker' element which shall be used as the arrowhead symbol or polymarker at the given vertex (vertices). If the URI reference is not valid (e.g., it points to an object that is undefined or the object is not a 'marker' element), then the marker(s) shall not be drawn.

The **'marker'** property specifies the marker symbol that shall be used for all points on the sets the value for all vertices on the given **'path'** element or vector graphic shape. It is a short-hand for the three individual marker properties:

'marker'

Value:	see individual properties
Initial:	see individual properties
Applies to:	all elements
Inherited:	see Inheritance of Painting Properties below
Percentages:	N/A
Media:	visual
Animatable:	yes

11.5.4 Details on how markers are rendered

The following provides details on how markers are rendered:

- Markers are drawn after the given object is filled and stroked.
- Each marker is drawn on the path by first creating a temporary viewport such that the origin of the viewport coordinate system is at the given vertex and the axes are aligned according to the **orient** attribute on the **'marker'** element.
- The width and height of the viewport is established by evaluating the values of <markerUnits>, <markerWidth> and <markerHeight> and calculating temporary values **computed-width** and **computed-height** in the user coordinate system of the object referencing the markers. **computed-width** and **computed-height** are used to determine the dimensions of the temporary viewport.
- The marker is drawn into the viewport.

For illustrative purposes, we'll repeat the marker example shown earlier:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="4in"
     viewBox="0 0 4000 4000" >
  <defs>
    <marker id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerWidth="1.25" markerHeight="1.75"
      orient="auto">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>
  <desc>Placing an arrowhead at the end of a path.
  </desc>
  <path d="M 1000 1000 L 2000 1000 L 3000 2000"
        style="fill:none; stroke:black; stroke-width:100;
        marker-end:url(#Triangle)" />
</svg></svg>
```

Download this example

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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="4in"
     viewBox="0 0 4000 4000" >
  <defs>
    <!-- Note: to illustrate the effect of "marker",
      replace "marker" with "symbol" and remove the various
      marker-specific attributes -->
    <symbol id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </symbol>
  </defs>
  <desc>File which produces the same effect
      as the marker example file, but without
      using markers.
  </desc>
  <!-- The path draws as before, but without the marker properties -->
  <path d="M 1000 1000 L 2000 1000 L 3000 2000"
        style="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(3000 2000)" >

    <!-- Now, 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)" >

      <!-- Establish a new viewport with an <svg> element.
           The width/height of the viewport are 1.25 and 1.75 times
           the current stroke-width, respectively. Since the
           current stroke-width is 100, the viewport's width/height
           is 125 by 175. Apply the viewBox attribute
           from the <marker> element onto this <svg> element.
           Transform the marker symbol to align (refX,refY) with
           the origin of the viewport. -->
      <svg width="125" height="175"
           viewBox="0 0 10 10"
           transform="translate(0,-5)" >

        <!-- Expand out the contents of the <marker> element. -->
        <path d="M 0 0 L 10 5 L 0 10 z" />
      </svg>
    </g>
  </g>
</svg>
```

Download this example

11.6 Rendering properties

The SVG user agent performs color interpolations and compositing in the following cases:

- when rendering gradients
- when performing color animations (see 'animateColor')
- when performing alpha blending/compositing of graphics elements into the current background
- when performing various filter effects

The 'color-interpolation' property specifies whether color interpolations and compositing shall be performed in the sRGB [SRGB] color space or in a (light energy linear) linearized RGB color space.

The conversion formulas between sRGB color space and linearized RGB color space is can be found in [SRGB]. The following formula shows the conversion from sRGB to linearized RGB:

```
  R'[sRGB] = R[sRGB] / 255
  G'[sRGB] = G[sRGB] / 255
  B'[sRGB] = B[sRGB] / 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
```

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:	color interpolation and compositing operations
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

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 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'

Value:	auto | optimizeSpeed | optimizeQuality | inherit
Initial:	auto
Applies to:	color interpolation and compositing operations
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

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.

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:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	no

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.

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:	no

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.

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:	no

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. This option will sometimes cause the user agent to use a bilinear image resampling algorithm.

optimizeQuality

Indicates that the user agent shall emphasize quality over rendering speed. This option will sometimes cause the user agent to use a bicubic image resampling algorithm.

The 'visibility' indicates whether a given object shall be rendered at all.

'visibility'

Value:	visible | hidden | inherit
Initial:	visible
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

visible

The current object is drawn.

hidden

The current object is not drawn.

11.7 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 leaf-node individually, never at the 'g' level. Thus, for the following SVG, two distinct gradients are painted (one for each rectangle):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
```

```
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Gradients apply to leaf nodes
  </desc>
  <g>
    <defs>
      <linearGradient id="MyGradient">
        <stop offset="0%" style="color:#F60"/>
        <stop offset="70%" style="color:#FF6"/>
      </linearGradient>
    </defs>
    <g style="fill: url(#MyGradient)">
      <rect width="20" height="15.8"/>
      <rect width="35" height="8"/>
   </g>
  </g>
</svg>
```

Download this example

11.8 DOM interfaces

11.8.1 Interface SVGICCColor

The SVGICCColor expresses an ICC-based color specification and is a base class for interface SVGColor

```
interface SVGICCColor  {
  readonly attribute  unsigned long numberOfColorValues;
  float      getColorValue(in unsigned long index);

  void           clear(); // Clear all entries, giving an empty list
  float      insertBefore(in float colorValue,
                                      in unsigned long index)
                                      raises(DOMException);
  float      replace(in float colorValue,
                                  in unsigned long index)
                                  raises(DOMException);
  float      remove(in unsigned long index)
                                 raises(DOMException);
  float      append(in float colorValue)
                                 raises(DOMException);
```

11.8.2 Interface SVGColor

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

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

```
interface SVGColor  {
  // Paint Types
  const unsigned short kSVG_COLORTYPE_UNKNOWN           = 0; // invalid, must be retrieved as a string
  const unsigned short kSVG_COLORTYPE_RGBCOLOR          = 1;
  const unsigned short kSVG_COLORTYPE_RGBCOLOR_ICCCOLOR = 2;
  readonly attribute  unsigned short colorType;

  readonly attribute RGBColor rgbColor;
  readonly attribute SVGICCColor iccColor;

  void setRGBColor(in RGBColor rgbColor);
  void setRGBColorICCColor(in RGBColor rgbColor, in SVGICCColor iccColor);

  // Convenience creation routines.
  RGBColor      createRGBColor();      // Returns RGBColor=black
  SVGICCColor   createSVGICCColor();   // Returns empty unattached list

  // If this property currently is being animated, these properties
  // reflect the current animated value.
  // Otherwise, they reflect the static document properties.
  readonly attribute  unsigned short animatedColorType;
  readonly attribute RGBColor animatedRgbColor;
  readonly attribute SVGICCColor animatedIccColor;
}
```

11.8.3 Interface SVGPaint

The SVGPaint interface corresponds to basic type <paint> and represents the values of properties 'fill' and 'stroke'.

```
interface SVGPaint : SVGColor  {
  // Paint Types
  // First three match SVGColor's colorTypes.
  const unsigned short kSVG_PAINTTYPE_UNKNOWN    = 0; // invalid, must be retrieved as a string
  const unsigned short kSVG_PAINTTYPE_RGBCOLOR              = 1;
  const unsigned short kSVG_PAINTTYPE_RGBCOLOR_ICCCOLOR     = 2;
  const unsigned short kSVG_PAINTTYPE_NONE                  = 101;
  const unsigned short kSVG_PAINTTYPE_CURRENTCOLOR          = 102;
  const unsigned short kSVG_PAINTTYPE_URI_NONE              = 103;
  const unsigned short kSVG_PAINTTYPE_URI_CURRENTCOLOR      = 104;
  const unsigned short kSVG_PAINTTYPE_URI_RGBCOLOR          = 105;
  const unsigned short kSVG_PAINTTYPE_URI_RGBCOLOR_ICCCOLOR = 106;
  readonly attribute  unsigned short paintType;

  readonly attribute DOMString uri;

  void setUri(in DOMString uri);
  void setPaint(in unsigned short paintType,
                in DOMString uri,
                in RGBColor rgbColor,
                in SVGICCColor iccColor);

  // If this property currently is being animated, these properties
  // reflect the current animated value.
  // Otherwise, they reflect the static document properties.
  readonly attribute  unsigned short animatedPaintType;
  readonly attribute DOMString animatedUri;
};
```

11.8.4 Interface SVGMarkerElement

The SVGMarkerElement interface corresponds to the 'marker' element.

```
interface SVGMarkerElement : SVGStyledElement {
  // markerUnit Types
  const unsigned short kSVG_MARKERUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_MARKERUNITS_USERSPACE         = 1;
  const unsigned short kSVG_MARKERUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_MARKERUNITS_STROKEWIDTH       = 3;
  attribute  unsigned short markerUnits;

  attribute SVGLength refX;
  attribute SVGLength refY;
  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;
  attribute SVGLength markerWidth;
  attribute SVGLength markerHeight;

  // orient Types
  const unsigned short kSVG_MARKER_ORIENT_UNKNOWN = 0;
  const unsigned short kSVG_MARKER_ORIENT_AUTO    = 1;
  const unsigned short kSVG_MARKER_ORIENT_ANGLE   = 2;
  readonly attribute SVGAngle orientAngle;

  void setOrientToAuto();
  void setOrientToAngle(SVGAngle angle);
};
```

---

12 Color

Contents

12.1 Introduction

All SVG colors are specified in the sRGB color space (see [SRGB]). At a minimum, SVG user agents shall conform to the color behavior requirements specified in the Colors chapter of the CSS2 specification (see [CSS2]).

Additionally, SVG content can specify an alternate color specification using an ICC profiles (see [ICC32]). If ICC-based colors are provided and the SVG user agent support ICC color, then the ICC-based color takes precedence over the sRGB color specification.

For more on specifying color properties, refer to the descriptions of the 'fill' property and the 'stroke' property.

The 'color' property is used to provide a potential indirect value (currentColor) for the 'fill' and 'stroke' properties.

'color'

Value:	<color> | inherit
Initial:	depends on user agent
Applies to:	'fill' and 'stroke' properties
Inherited:	see Inheritance of Painting Properties
Percentages:	N/A
Media:	visual
Animatable:	yes

For a description of the parameters, refer to [CSS2].

12.2 Color profile descriptions and @color-profile

The International Color Consortium has established a standard, the ICC Profile [ICC32], for documenting the color characteristics of input and output devices. Using these profiles, it is possible to build a transform and correct visual data for viewing on different devices.

A color profile description provides the bridge between an ICC profile and references to that ICC profile within SVG content. The color profile description is added to the color profile database 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.

Color profile descriptions in CSS style sheets are specified via an *@color-profile* rule. The general form is:

```
@color-profile: { <color-profile-description> }
```

where the <color-profile-description> has the form:

```
descriptor: value;
[...]
descriptor: value;
```

Each @color-profile rule specifies a value for every color profile descriptor, either implicitly or explicitly. Those not given explicit values in the rule take the initial value listed with each descriptor in this specification. These descriptors apply solely within the context of the @color-profile rule in which they are defined, and do not apply to document language elements. Thus, there is no notion of which elements the descriptors apply to, or whether the values are inherited by child elements.

The following are the descriptors for a <color-profile-description>:

'src' (Descriptor)

> *Values:* sRGB | <uri> | inherit
> *Initial:* auto
> *Media:* visual

sRGB

> The source profile is assumed to be sRGB [SRGB]. This differs from **auto** in that it overrides an embedded profile inside an image.

<uri>

> The name or location of a standard ICC profile resource. Due to the size of profiles, the <uri> may specify a special name representing a standard profile. The name sRGB, being the standard WWW color space, is defined separately because of its significance, although the rules regarding application of any special profile shall be identical.

'name' (Descriptor)

> *Values:* <name>
> *Initial:* undefined
> *Media:* visual

<name>

> The name which is used as the first parameter for icc-color specifications within 'fill', 'stroke' and 'stop-color' property values to identify the color profile to use for the ICC color specification.

Note that if <name> is not provided, it will be impossible to reference the given @color-profile definition.

'rendering-intent' (Descriptor)

Values:	auto | perceptual | relative-colorimetric | saturation | absolute-colorimetric | inherit
Initial:	auto
Media:	visual
Animatable:	no

This property permits the specification of a color profile rendering intent other than the default. The behavior of values other than *auto* and *inherent* are defined by the International Color Consortium standard.

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 (based on the *color-profile* style) and force the intent, overriding any intent that might be stored in the profile itself.

---

13 Gradients and Patterns

Contents

13.1 Introduction

With SVG, you can fill (i.e., paint the interior) or stroke (i.e., paint the outline) of shapes and text using one of the following:

- color
- gradients (linear or radial)
- patterns (vector or image, possibly tiled)

SVG uses the general notion of a **paint server**. Gradients and patterns are just specific types of paint servers. For example, first you define a linear gradient by including a 'linearGradient' element within a 'defs', assign an ID to that 'linearGradient' element, and then reference that ID in a 'fill' or 'stroke' property:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
```

```
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Linear gradient example
  </desc>
  <g>
    <defs>
      <linearGradient id="MyGradient">
        <stop offset="0%" style="stop-color:#F60"/>
        <stop offset="70%" style="stop-color:#FF6"/>
      </linearGradient>
    </defs>
    <rect style="fill: url(#MyGradient)" width="20" height="15.8"/>
  </g>
</svg>
```

Download this example

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**.

Gradients are specified within a 'defs' element and are then referenced using 'fill' or 'stroke' or properties on a given graphics element to indicate that the given element shall be filled or stroked with the referenced gradient.

13.2.2 Linear gradients

Linear gradients are defined by a **'linearGradient'** element.

```
<!ENTITY % linearGradientExt "" >
<!ELEMENT linearGradient (stop|animate|set|animateTransform
                  %linearGradientExt;)* >
<!ATTLIST linearGradient
  id ID #IMPLIED
  gradientUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  x1 CDATA #IMPLIED
  y1 CDATA #IMPLIED
  x2 CDATA #IMPLIED
  y2 CDATA #IMPLIED
  spreadMethod (pad | reflect | repeat) "pad"
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >
```

Attribute definitions:

gradientUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

Defines the coordinate system for attributes x1, y1, x2, y2.
If gradientUnits="userSpace" (the default), x1, y1, x2, y2 represent values in the current user

coordinate system in place at the time when the 'linearGradient' element is defined.
If gradientUnits="userSpaceOnUse", x1, y1, x2, y2 represent values in the current user coordinate system in place at the time when the 'linearGradient' element is referenced (i.e., the user coordinate system for the element referencing the 'linearGradient' element via a 'fill' or 'stroke' property).
If gradientUnits="objectBoundingBox", then x1, y1, x2, y2 represent values in an abstract coordinate system where (0,0) is the (minx,miny) in user space of the bounding box of the object getting filled with the gradient, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
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., userSpace or objectBoundingBox). This allows for things such as skewing the gradient.
Animatable: yes.

x1 = "<coordinate>"

x1, y1, x2, y2 define a *gradient vector* for the linear gradient. This *gradient vector* provides starting and ending points onto which the gradient stops are mapped. The values of **x1, y1, x2, y2** can be either numbers or percentages whose meaning is determined by the value of attribute gradientUnits, as follows:

gradientUnits	Type of value	Meaning of value
"userSpace"	a number	The value represents a coordinate in the current user coordinate system
"userSpace"	a percentage	The value represents a percent distance along the X-axis of the current viewport (see Processing rules for CSS units and percentages)
"objectBoundingBox"	a number	The value represents a fractional position within the bounding box of the given shape, where (0,0) is the (minx,miny) of the shape and (1,1) is the (maxx,maxy) of the shape. (See discussion of gradientUnits="objectBoundingBox".)
"objectBoundingBox"	a percentage	The value represents a fractional position within the bounding box of the given shape, where (0%,0%) is the (minx,miny) of the shape and (100%,100%) is the (maxx,maxy) of the shape. (See discussion of gradientUnits="objectBoundingBox".)

Default value is "0%".
Animatable: yes.

y1 = "<coordinate>"

See x1. Default value is "0%".
Animatable: yes.

x2 = "<coordinate>"

See x1. Default value is "100%".
Animatable: yes.

y2 = "<coordinate>"

See x1. Default value is "0%".
Animatable: yes.

spreadMethod = "pad | reflect | repeat"

Indicates what happens if the 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.
Animatable: yes.

xlink:href = "<uri>"

A URI reference to a different 'linearGradient' or 'radialGradient' element within the current SVG document fragment. Any 'linearGradient' attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no defined gradient stops, and the referenced element does (possibly due to its own href attribute), then this element inherits the gradient stop from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attribute or gradient stops due to its own href attribute, then the current element can inherit those attributes or gradient stops.
Animatable: yes.

Attributes defined elsewhere:

id, %xlinkAttrs;.

Percentages are allowed for **x1, y1, x2, y2**. For gradientUnits="userSpace", percentages represent values relative to the current viewport. For gradientUnits="objectBoundingBox", percentages represent values relative to the bounding box for the object.

13.2.3 Radial gradients

Radial gradients are defined by a **'radialGradient'** element.

```
<!ENTITY % radialGradientExt "" >
<!ELEMENT radialGradient (stop|animate|set|animateTransform
                    %radialGradientExt;)* >
<!ATTLIST radialGradient
  id ID #IMPLIED
  gradientUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  cx CDATA #IMPLIED
  cy CDATA #IMPLIED
  r CDATA #IMPLIED
  fx CDATA #IMPLIED
  fy CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >
```

Attribute definitions:

gradientUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

Defines the coordinate system for attributes cx, cy, r, fx, fy.
If **gradientUnits="userSpace"** (the default), cx, cy, r, fx, fy represent values in the current user coordinate system in place at the time when the 'linearGradient' element is defined.
If gradientUnits="userSpaceOnUse", cx, cy, r, fx, fy represent values in the current user coordinate system in place at the time when the 'radialGradient' element is referenced (i.e., the user coordinate system for the element referencing the 'radialGradient' element via a 'fill' or 'stroke' property).
If **gradientUnits="objectBoundingBox"**, then cx, cy, r, fx, fy represent values in an abstract coordinate system where (0,0) is the (minx,miny) in user space of the bounding box of the object getting filled with the gradient, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
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., userSpace or objectBoundingBox). This allows for things such as skewing the gradient.
Animatable: yes.

cx = "<coordinate>"

cx, cy, r define the largest/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/outermost circle.
Default value is "50%".
Animatable: yes.

cy = "<coordinate>"

See cx. Default value is "50%".
Animatable: yes.

r = "<length>"

See cx. Default value is "50%".
Animatable: yes.

fx = "<coordinate>"

fx, fy define the focal point for the radial gradient. The gradient will be drawn such that the 0% gradient stop is mapped to (fx, fy). The default value is 50%.
Animatable: yes.

fy = "<coordinate>"

See fx. Default value is "50%".
Animatable: yes.

xlink:href = "<uri>"

A URI reference to a different 'linearGradient' or 'radialGradient' element within the current SVG document fragment. Any 'radialGradient' attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has no defined gradient stops, and the referenced element does (possibly due to its own href attribute), then this element inherits the gradient stop from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attribute or gradient stops due to its own href attribute, then the current element can inherit those attributes or gradient stops.
Animatable: yes.

Attributes defined elsewhere:

id, %xlinkAttrs;.

Percentages are allowed for **cx, cy, r, fx, fy**. For gradientUnits="userSpace", percentages represent values relative to the current viewport. For gradientUnits="objectBoundingBox", percentages represent values relative to the bounding box for the object.

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. Here is an example of the definition of a linear gradient that consists of a smooth transition from white-to-red-to-black:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Radial gradient example with three gradient stops
  </desc>
  <g>
    <defs>
      <radialGradient id="MyGradient">
         <stop offset="0%" style="stop-color:white"/>
         <stop offset="50%" style="stop-color:red"/>
         <stop offset="100%" style="stop-color:black"/>
      </radialGradient>
    </defs>
    <circle style="fill: url(#MyGradient)" r="42"/>
  </g>
</svg>
```

Download this example

```
<!ENTITY % stopExt "" >
<!ELEMENT stop (animate|set|animateColor
                  %stopExt;)* >
<!ATTLIST stop
  id ID #IMPLIED
  style CDATA #IMPLIED
  offset CDATA #REQUIRED >
```

Attribute definitions:

offset = "*length*"

The **offset** attribute is either a <number> (usually ranging from 0 to 1) or a percentage (correspondingly usually ranging from 0% to 100%) which indicates where the gradient stop is placed. For linear gradients, the offset attribute represents a location along the *gradient vector*. For radial gradients, it represents a percentage distance from (fx,fy) to the edge of the outermost/largest circle.
Animatable: yes.

Attributes defined elsewhere:

id, style.

The 'stop-color' property indicates what color to use at that gradient stop. ICC colors can be specified in the same manner as for the 'fill' and 'stroke' properties.

'stop-color'

Value:	<color> | inherit
Initial:	black
Applies to:	'stop' elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

The 'stop-opacity' property defines the opacity of a given gradient stop.

'stop-opacity'

Value:	<alphavalue> | inherit
Initial:	1
Applies to:	'stop' elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

Some notes on gradients:

- Gradient offset values less than 0 (or less than 0%) are rounded up to 0%. Gradient offset values greater than 1 (or greater than 100%) are rounded down to 100%.
- There needs to be 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.

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:**.

```
<!ENTITY % patternExt "" >
<!ELEMENT pattern (%descTitleDefs;,
                   (path|text|rect|circle|ellipse|line|polyline|polygon|
                    use|image|svg|g|switch|a
                    %ceExt;%patternExt;)*) >
<!ATTLIST pattern
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  patternUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  patternTransform CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >
```

Attribute definitions:

patternUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

Defines the coordinate system for attributes x, y, width, height and the contents of the 'pattern'. If patternUnits="userSpace" (the default), x, y, width, height and the contents of the 'pattern' represent values in the current user coordinate system in place at the time when the 'mask' element is defined.
If patternUnits="userSpaceOnUse", x, y, width, height and the contents of the 'pattern' represent values in 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).

If patternUnits="objectBoundingBox", x, y, width, height and the contents of the 'pattern' represent values in the abstract coordinate system where (0,0) is the (minx,miny) in user space of the tight bounding box of the object referencing the mask, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
Animatable: yes.

patternTransform = "<transform-list>"

Contains the definitions of an optional additional transformation from the pattern coordinate system onto the target coordinate system (i.e., userSpace or objectBoundingBox). This allows for things such as skewing the pattern tiles.
Animatable: yes.

x = "<coordinate>"

x, y, width, height indicate how the pattern tiles are placed and spaced and represent coordinates and values in the coordinate space specified by **patternUnits**. Default value is "0%".
Animatable: yes.

y = "<coordinate>"

See x. Default value is "0%".
Animatable: yes.

width = "<length>"

See x. Default value is "100%".
Animatable: yes.

height = "<length>"

See x. Default value is "100%".
Animatable: yes.

xlink:href = "<uri>"

A URI reference to a different 'pattern' element within the current SVG document fragment. Any attributes which are defined on the referenced element which are not defined on this element are inherited by this element. If this element has children, and the referenced element does (possibly due to its own href attribute), then this element inherits the children from the referenced element. Inheritance can be indirect to an arbitrary level; thus, if the referenced element inherits attributes or children due to its own href attribute, then the current element can inherit those attributes or gradient stops.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style, refX, refY, viewBox, preserveAspectRatio, %xlinkAttrs;.

An example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in" >
  <defs>
```

```
    <pattern id="TrianglePattern"
             patternUnits="userSpace"
             x="0" y="0" width="25" height="25"
             patternTransform="skewX(45)"
             viewBox="0 0 10 10" >
      <path d="M 0 0 L 10 0 L 5 10 z" />
    </pattern>
  </defs>
  <!-- Fill this ellipse with the above pattern -->
  <ellipse style="fill: url(#TrianglePattern)" rx="40" ry="27" />
</svg>
```

Download this example

13.4 DOM interfaces

13.4.1 Interface SVGGradientElement

The SVGGradientElement interface is a base interface used by SVGLinearGradientElement and SVGRadialGradientElement.

```
interface SVGGradientElement : SVGElement {
  // gradientUnits Types
  const unsigned short kSVG_GRADIENTUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_GRADIENTUNITS_USERSPACE         = 1;
  const unsigned short kSVG_GRADIENTUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_GRADIENTUNITS_OBJECTBOUNDINGBOX = 3;
  attribute unsigned short gradientUnits;
  attribute SVGTransformList gradientTransform;
};
```

13.4.2 Interface SVGLinearGradientElement

The SVGLinearGradientElement interface corresponds to the 'linearGradient' element.

```
interface SVGLinearGradientElement : SVGGradientElement {
  attribute SVGLength x1;
  attribute SVGLength y1;
  attribute SVGLength x2;
  attribute SVGLength y2;

  // spreadMethod Types
  const unsigned short kSVG_SPREADMETHOD_PAD     = 0;
  const unsigned short kSVG_SPREADMETHOD_REFLECT = 1;
  const unsigned short kSVG_SPREADMETHOD_REPEAT  = 2;
  attribute  unsigned short spreadMethod;
};
```

13.4.3 Interface SVGRadialGradientElement

The SVGRadialGradientElement interface corresponds to the 'radialGradient' element.

```
interface SVGRadialGradientElement : SVGGradientElement {
  attribute SVGLength cx;
  attribute SVGLength cy;
  attribute SVGLength r;
  attribute SVGLength fx;
  attribute SVGLength fy;
};
```

13.4.4 Interface SVGStopElement

The SVGStopElement interface corresponds to the 'stop' element.

```
interface SVGStopElement : SVGStyledElement {
  attribute float offset;
};
```

13.4.5 Interface SVGPatternElement

The SVGPatternElement interface corresponds to the 'pattern' element.

```
interface SVGPatternElement : SVGStyledElement {
  // patternUnits Types
  const unsigned short kSVG_PATTERNUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_PATTERNUNITS_USERSPACE         = 1;
  const unsigned short kSVG_PATTERNUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_PATTERNUNITS_OBJECTBOUNDINGBOX = 3;
  attribute unsigned short patternUnits;
  attribute SVGTransformList patternTransform;
  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
  attribute SVGLength refX;
  attribute SVGLength refY;
  attribute SVGRect viewBox;
  attribute SVGPreserveAspectRatio preserveAspectRatio;
};
```

---

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 absense of antialiasing) 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 antialiasing 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 Blending/Compositing).

(Insert drawings showing a clipping path, a grayscale imagemask, simple alpha blending and more complex blending.)

14.2 Simple alpha blending/compositing

Graphics elements are blended into the elements already rendered on the canvas using simple alpha blending/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 opacity/alpha value
Cr, Cg, Cb    - Canvas color value (before blending)
Ca            - Canvas opacity/alpha value (before blending)
Cr', Cg', Cb' - Canvas color value (after blending)
Ca'           - Canvas opacity/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 1-bit mask.

14.3.2 The initial clipping path

When an 'svg' element is encountered by a CSS user agent, the CSS user agent needs to establish an initial clipping path for the SVG document fragment. The 'overflow' and 'clip' properties from CSS2 along with additional SVG user agent processing rules determine the initial clipping path which the CSS user agent establishes for the SVG document fragment:

14.3.3 The 'overflow' and 'clip' properties

'overflow'

Value:	visible | hidden | scroll | auto | inherit
Initial:	visible (see notes below)
Applies to:	elements which establish a new viewport
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	N/A

The 'overflow' property has the same parameter values and has the same meaning as defined in [CSS2-overflow]; however, the following additional points apply:

- The 'overflow' property only applies to elements that establish new viewports, such as 'svg' elements. (See the discussion of the elements which establish a new viewport.)
- When an outermost SVG 'svg' element is embedded inline within a parent XML grammar which uses CSS layout [CSS2-LAYOUT] or XSL formatting [XSL], if the 'overflow' property has the value hidden, then the SVG user agent will establish an initial clipping path equal to the bounds of the initial viewport; otherwise, the initial clipping path is set according to the clipping rules as defined in [CSS2-overflow].
- When an outermost SVG 'svg' element is standalone or embedded inline within a parent XML grammar which does not use CSS layout [CSS2-LAYOUT] or XSL formatting [XSL], the 'overflow' property on the outermost 'svg' element is ignored for the purposes of visual rendering and the initial clipping path is set to the bounds of the initial viewport.
- For 'svg' elements that are embedded inside of an ancestor SVG document fragment (i.e., without a 'foreignObject' element between the inner 'svg' and the nearest ancestor 'svg') or for any other elements which establish a new viewport, the 'overflow' property determines whether an additional new clipping path is established around the bounds of the viewport established by the given element. If the value of the the 'overflow' property is hidden, then a new clipping path is established; otherwise, no new clipping path is established.
- The initial value for 'overflow' as defined in [CSS2-overflow] is 'visible'; however, the Default styles sheet for SVG specifies that the 'overflow' property on all elements within an SVG document fragment has 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.

For stand-alone SVG viewers or in situations where an SVG document fragment is embedded inline within a parent XML grammar which does not use CSS layout or XSL formatting, then the initial clipping path must be set to the bounds of the viewing region in which the SVG document fragment is rendered, even if the and the 'overflow' property is set to a value other than hidden.

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
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	N/A

The 'clip' property only applies to elements which establish a new viewport. The 'clip' property has the same parameter values as defined in [CSS2-clip]. Unitless values, which indicates current user coordinates, are permitted on the coordinate values on the <shape>. The value of "auto" indicates 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 Default style sheet for SVG 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 on a viewport-creating element, it is sometime desirable that the initial viewport be set to the bounds of the viewBox instead of the viewport, 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 instead of the viewport, set the bounds of 'clip' property to the same rectangle as specified on the viewBox attribute. (Note that the parameters don't match. 'clip' takes values <top>, <right>,<bottom> and <left>, whereas viewBox takes values <min-x>, <min-y>, <width> and <height>.)

14.3.5 Establishing a new clipping path

A clipping path is defined with a 'clipPath' element. A clipping path is used/referenced using the 'clip-path' property.

A 'clipPath' element can contain 'path' elements, 'text' elements, other vector graphic 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 vector graphic shape elements. Indirect references are an error (see Error processing). 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.

It is an error if the 'clip-path' property references a non-existent object or if the referenced object is not a 'clipPath' element (see Error processing).

For a given graphics element, the actual clipping path used will be the intersection of the clipping path specified by its 'clip-path' property (if any) with any clipping paths on its ancestors, as specified by the 'clip-path' property on the ancestor elements.

A couple of notes:

- The 'clipPath' element itself and its child elements do *not* inherit clipping paths from the

ancesotors of the 'clipPath' element.

- The 'clipPath' element or any of its children can specify property 'clip-path'.
  If a valid 'clip-path' reference is placed on a 'clipPath' element, the resulting clipping path is the intersection of the contents of the 'clipPath' element with the referenced clipping path.
  If a valid 'clip-path' reference is placed on one of the children of a 'clipPath' element, then the given child element is clipped by the referenced clipping path before OR'ing the silhouette of the child element with the silhouettes of the other child elements.

```
<!ENTITY % clipPathExt "" >
<!ELEMENT clipPath (%descTitle;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%clipPathExt;)*) >
<!ATTLIST clipPath
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  clipPathUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace" >
```

Attribute definitions:

clipPathUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

Defines the coordinate system for the contents of the 'clipPath'.
If clipPathUnits="userSpace" (the default), the contents of the 'clipPath' represent values in the current user coordinate system in place at the time when the 'clipPath' element is defined.
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", the contents of the 'clipPath' represent values in the abstract coordinate system where (0,0) is the (minx,miny) in user space of the tight bounding box of the object referencing the mask, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style.

'clip-path'

Value:	<uri> | none | inherit
Initial:	See The initial clipping path: 'overflow' and 'clip' properties
Applies to:	all elements
Inherited:	no

Percentages: N/A
Media: visual
Animatable: yes

<uri>

A URI reference to another graphical object within the same SVG document fragment which will be used as the clipping path.

'clip-rule'

Value: evenodd | nonzero | inherit
Initial: evenodd
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Animatable: yes

evenodd

nonzero

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 grouping 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. 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.

The following processing rules apply:

- If all of the child elements of the 'mask' consist of the same type of one-channel image (i.e., a grayscale image or an image consisting only of an alpha channel), then the child elements will be processed as single channel images into a resulting single channel image result, and that single channel result will be used as the mask.
- If all of the child elements of the 'mask' consist of three-channel RGB images, then the child elements will be processed as RGB images into a resulting RGB image result, and the luminance from the resulting RGB image will be used as the mask, where the luminance is calculated using the luminance-to-alpha formulas as defined in the 'feColorMatrix' filter effect.
- Otherwise, the child elements of the 'mask' will will be processed and will result in a four-channel RGBA image, and the alpha channel from this resulting RGBA image will be used as the mask.

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.

In the following example, an image is used to mask a rectangle:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>Example of using a mask
  </desc>
  <g>
    <defs>
      <mask id="MyMask">
        <image xlink:href="transp.png" />
      </mask>
    </defs>
    <rect style="mask: url(#MyMask)" width="12.5" height="30" />
  </g>
</svg>
```

Download this example

A <mask> element can define a region on the canvas for the mask using the following attributes:

```
<!ENTITY % maskExt "" >
<!ELEMENT mask (%descTitleDefs;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|image|svg|g|switch|a|
                     animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%maskExt;)*) >
<!ATTLIST mask
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  maskUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED >
```

Attribute definitions:

maskUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

Defines the coordinate system for attributes x, y, width, height and the contents of the 'mask'. If maskUnits="userSpace" (the default), x, y, width, height and the contents of the 'mask' represent values in the current user coordinate system in place at the time when the 'mask' element is defined.
If maskUnits="userSpaceOnUse", x, y, width, height and the contents of the 'mask' represent values in the current user coordinate system in place at the time when the 'mask' element is referenced (i.e., the user coordinate system for the element referencing the 'mask' element via the 'mask' property).
If maskUnits="objectBoundingBox", x, y, width, height and the contents of the 'mask' represent values in the abstract coordinate system where (0,0) is the (minx,miny) in user space of the tight

bounding box of the object referencing the mask, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
Animatable: yes.

x = "<coordinate>"

The x coordinate of one corner of the rectangle for the largest possible offscreen buffer, where the values are either relative to the current user coordinate system (if maskUnits="userSpace") or relative to the current object (if maskUnits="objectBoundingBox"). Note that the clipping path used to render any graphics within the mask will consists of the intersection of the current clipping path associated with the given object and the rectangle defined by x, y, width, height. The default value for x is 0%.
Animatable: yes.

y = "<coordinate>"

The y coordinate of one corner of the rectangle for the largest possible offscreen buffer.The default value for y is 0%.
Animatable: yes.

width = "<length>"

The width of the largest possible offscreen buffer, where the values are either relative to the current user coordinate system (if maskUnits="userSpace") or relative to the current object (if maskUnits="objectBoundingBox"). Note that the clipping path used to render any graphics within the mask will consists of the intersection of the current clipping path associated with the given object and the rectangle defined by x, y, width, height. The default value for width is 100%.
Animatable: yes.

height = "<length>"

The height of the largest possible offscreen buffer. The default value for height is 100%.
Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, class, style.

The following is a description of the 'mask' property.

'mask'

Value:	<uri> | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

<uri>

A URI reference to another graphical object which will be used as the mask.

14.5 Object and group opacity: the 'opacity' property

There are several opacity properties within SVG:

- Fill opacity
- Stroke opacity
- Gradient stop opacity
- Object/group opacity (described here)

Except for object/group opacity (described just below), all other opacity properties are involved in intermediate rendering operations. Object/group opacity can be thought of conceptually as a postprocessing operation. Conceptually, after the object/group is rendered into an RGBA offscreen image, the object/group opacity setting specifies how to blend the offscreen image into the current background.

'opacity'

Value:	<alphavalue> | inherit
Initial:	1
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

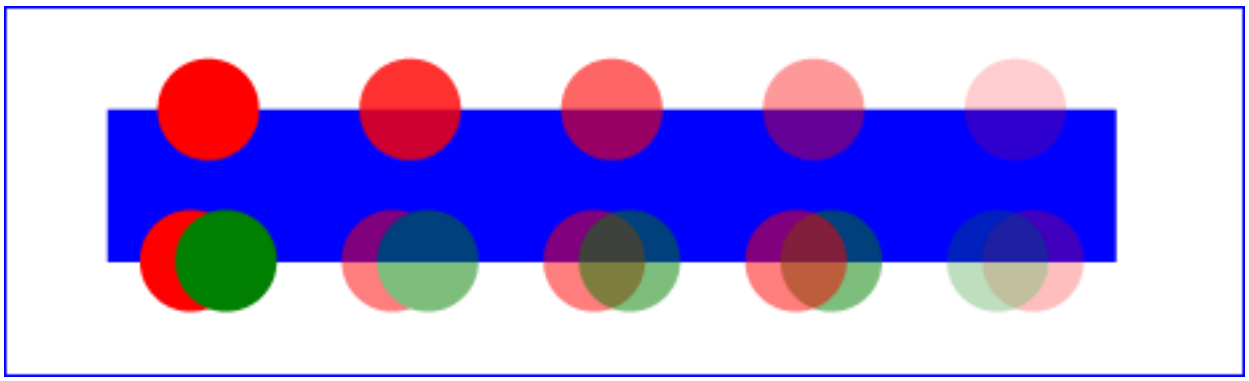
<alphavalue>

The uniform opacity setting to be applied across an entire object. If a <number> is provided, any values outside the range 0.0 (fully transparent) to 1.0 (fully opaque) will be clamped to this range. If a percentage is provided, any value outside the range of 0% to 100% will be clamped to this range. (See Clamping values which are restricted to a particular range If the object is a container element such as a 'g', then the effect is as if the contents of the 'g' were blended against the current background using a mask where the value of each pixel of the mask is <alphavalue>. (See Simple alpha blending/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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="12cm" height="3.5cm">
  <desc>Example opacity01 - opacity property</desc>

  <!-- Background blue rectangle -->
  <rect x="1cm" y="1cm" width="10cm" height="1.5cm" style="fill:#0000ff" />

  <!-- Red circles going from opaque to nearly transparent -->
  <circle cx="2cm" cy="1cm" r=".5cm" style="fill:red; opacity:1" />
  <circle cx="4cm" cy="1cm" r=".5cm" style="fill:red; opacity:.8" />
  <circle cx="6cm" cy="1cm" r=".5cm" style="fill:red; opacity:.6" />
  <circle cx="8cm" cy="1cm" r=".5cm" style="fill:red; opacity:.4" />
  <circle cx="10cm" cy="1cm" r=".5cm" style="fill:red; opacity:.2" />

  <!-- Opaque group, opaque circles -->
```

```
  <g style="opacity:1">
    <circle cx="1.825cm" cy="2.5cm" r=".5cm" style="fill:red; opacity:1" />
    <circle cx="2.175cm" cy="2.5cm" r=".5cm" style="fill:green; opacity:1" />
  </g>
  <!-- Group opacity: .5, opacity circles -->
  <g style="opacity:.5">
    <circle cx="3.825cm" cy="2.5cm" r=".5cm" style="fill:red; opacity:1" />
    <circle cx="4.175cm" cy="2.5cm" r=".5cm" style="fill:green; opacity:1" />
  </g>
  <!-- Opaque group, semi-transparent green over red -->
  <g style="opacity:1">
    <circle cx="5.825cm" cy="2.5cm" r=".5cm" style="fill:red; opacity:.5" />
    <circle cx="6.175cm" cy="2.5cm" r=".5cm" style="fill:green; opacity:.5" />
  </g>
  <!-- Opaque group, semi-transparent red over green -->
  <g style="opacity:1">
    <circle cx="8.175cm" cy="2.5cm" r=".5cm" style="fill:green; opacity:.5" />
    <circle cx="7.825cm" cy="2.5cm" r=".5cm" style="fill:red; opacity:.5" />
  </g>
  <!-- Group opacity .5, semi-transparent green over red -->
  <g style="opacity:.5">
    <circle cx="10.175cm" cy="2.5cm" r=".5cm" style="fill:red; opacity:.5" />
    <circle cx="9.825cm" cy="2.5cm" r=".5cm" style="fill:green; opacity:.5" />
  </g>
</svg>
```



Example opacity01

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

In the example above, the top row of circles have differing opacities, ranging from 1.0 to 0.2. The bottom row illustrates five 'g' elements, each of which contains overlapping red and green circles, as follows:

- The first group shows the opaque case for reference. The group has opacity of 1, as do the circles.
- The second group shows group opacity when the elements in the group are opaque.
- The third and fourth group show that opacity is not commutative. In the third group (which has opacity of 1), a semi-transparent green circle is drawn on top of a semi-transparent red circle, whereas in the fourth group a semi-transparent red circle is drawn on top of a semi-transparent green circle. Note that area where the two circles intersect display different colors. The third group shows more green color in the intersection area, whereas the fourth group shows more red color.
- The fifth group shows the multiplicative effect of opacity settings. Both the circles and the group itself have opacity settings of .5. The result is that the portion of the red circle which does not overlap with the green circle (i.e., the top/right of the red circle) will blend into the blue rectangle

with accumulative opacity of .25 (i.e., .5*.5), which, after blending into the blue rectangle, results in a blended color which is 25% red and 75% blue.

14.6 DOM interfaces

14.6.1 Interface SVGClipPath

The SVGClipPath interface corresponds to the clipPath element.

```
interface SVGClipPath : SVGStyledElement {
  // clipPathUnit Types
  const unsigned short kSVG_CLIPPATHUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_CLIPPATHUNITS_USERSPACE         = 1;
  const unsigned short kSVG_CLIPPATHUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_CLIPPATHUNITS_OBJECTBOUNDINGBOX = 3;
  attribute  unsigned short clipPathUnits;
};
```

14.6.2 Interface SVGMask

The SVGMask interface corresponds to the 'mask' element.

```
interface SVGMask : SVGStyledElement {
  // maskUnit Types
  const unsigned short kSVG_MASKUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_MASKUNITS_USERSPACE         = 1;
  const unsigned short kSVG_MASKUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_MASKUNITS_OBJECTBOUNDINGBOX = 3;
  attribute  unsigned short maskUnits;

  attribute  SVGLength x;
  attribute  SVGLength y;
  attribute  SVGLength width;
  attribute  SVGLength height;
};
```

---

15 Filter Effects

Contents

15.1 Introduction

A model for adding declarative raster-based rendering effects to a 2D graphics environment is presented. As a result, the expressiveness of the traditional 2D rendering model is greatly enhanced, while still preserving the device independence, scalability, and high level geometric description of the underlying graphics.

15.2 Background

On the Web, many graphics are presented as bitmap images in gif, jpg, or png format. Among the many disadvantages of this approach is the general difficulty of keeping the raster data in sync with the rest of the Web site. Many times, a web site designer must resort to a bitmap editor to simply change the title of a button. As the Web gets more dynamic, we desire a way to describe the "piece parts" of a site in a more flexible format. This chapter describes SVG's declarative filter effects model, 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.

15.3 Basic Model

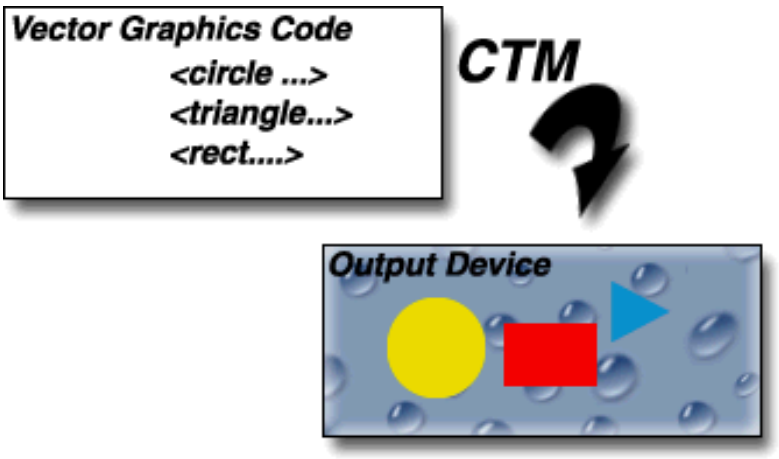
The filter effects model consists of a set of filtering operations (called "processing nodes" in the descriptions below) on one or more graphic primitives. Each processing node takes a set of graphics primitives as input, performs some processing, and generates revised graphics primitives as output. Because nearly all of the filtering operations are some form of image processing, in almost all cases the output from most processing nodes consists of a single RGBA image.

For example, a simple filter could replace one graphic by two -- by adding a black copy of original offset to create a drop shadow. In effect, there are now two layers of graphics, both with the same original set of graphics primitives. In this example, the bottommost shadow layer could be blurred and become a raster layer, while the topmost layer could remain as

higher-order graphics primitives (e.g., text or vector objects). Ultimately, the two layers are composited together and rendered into the background.

Filter effects introduce an additional step into the traditional 2D graphics pipeline. Consider the traditional 2D graphics pipeline:

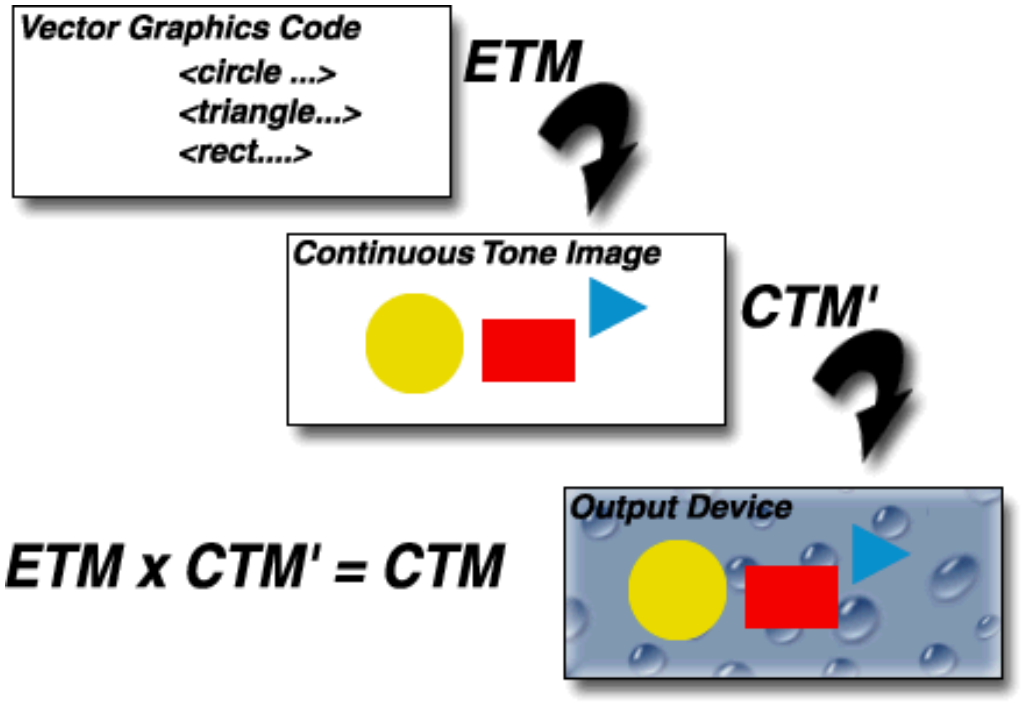
Traditional 2D graphics pipeline



Vector graphics primitives are specified abstractly and rendered onto the output device through a geometric transformation called the *current transformation matrix,* or *CTM*. The CTM allows the vector graphics code to be specified in a device independent coordinate system. At rendering time, the CTM accounts for any differences in resolution or orientation of the input vector description space and the device coordinate system. According to the "painter's model", areas on the device which are outside of the vector graphic shapes remain unchanged from their previous contents (in this case the droplet pattern).

Consider now, altering this pipeline slightly to allow rendering the graphics to an intermediate continuous tone image which is then rendered onto the output device in a second pass:

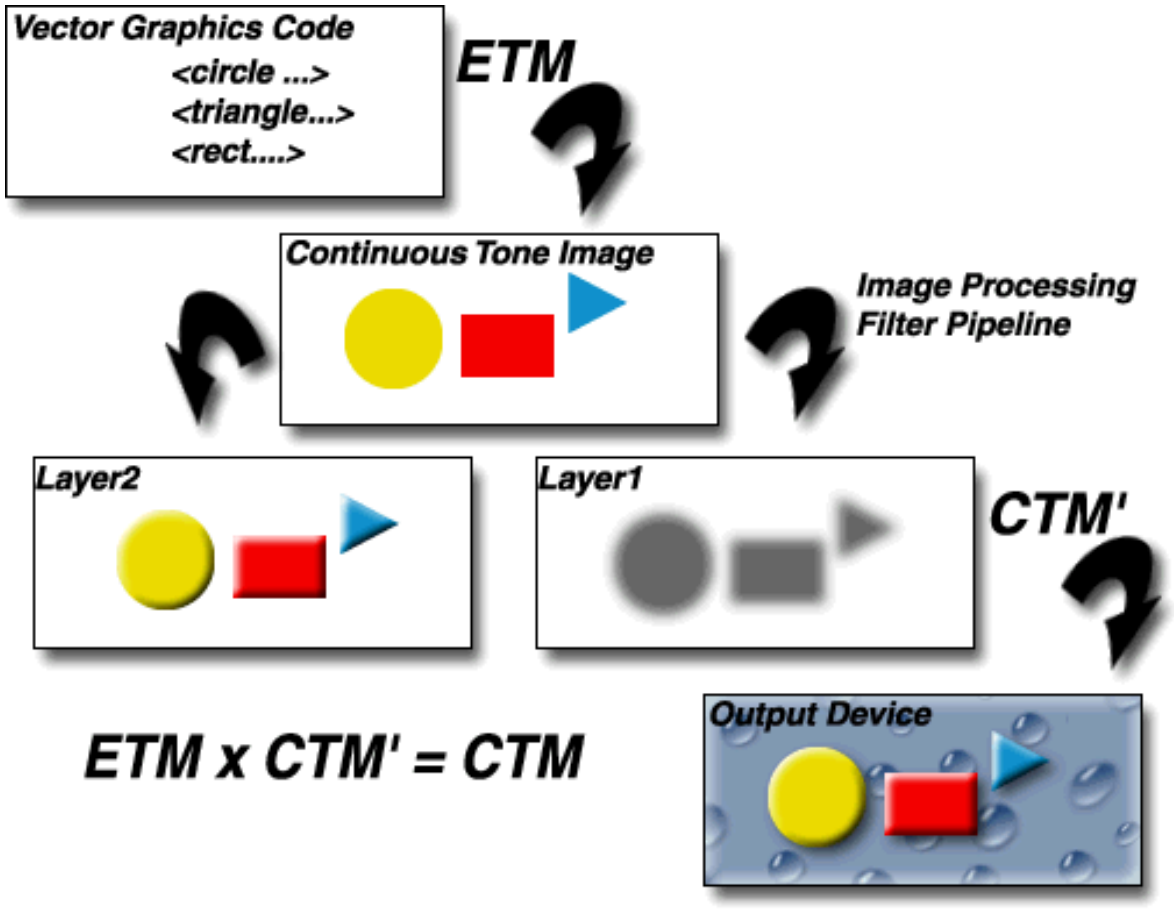
Rendering via Continuous Tone Intermediate Image



We introduce a new transformation matrix called the *Effect Transformation Matrix,* or *ETM*. Vector primitives are rendered via the ETM onto an intermediate continuous tone image. This image is then rendered onto the output device using the standard 2D imaging path via a modified transform, *CTM',* such that the net effect of *ETM* followed by *CTM'* is equivalent to the original CTM. It is important to note that the intermediate continuous tone image contains coverage information so that non rendered parts of the original graphic are transparent in the intermediate image and remain unaffected on the output device, as required by the painter's model. A physical analog to this process is to imagine rendering the vector primitives onto a sheet of clear acetate and then transforming and overlaying the acetate sheet onto the output device. The resulting imaging model remains as device-independent as the original one, except we are now using the 2D imaging model itself to generate images to render.

So far, we really haven't added any new expressiveness to the imaging model. What we *have* done is reformulated the traditional 2D rendering model to allow an intermediate continuous tone rasterization phase. However, now we can extend this further by allowing the application of image processing operations on the intermediate image, still without sacrificing device independence. In our model, the intermediate image can be operated on by a number of image processing operations which can effect both the color and coverage channels. The resulting image(s) are then rendered onto the device in the same way as above.

Rendering via Continuous Tone Intermediate Step with Image Processing



In the picture above, the intermediate set of graphics primitives was processed in two ways. First a simple bump map lighting calculation was applied to add a 3D look, then another copy of the original layer was offset, blurred and colored black to form a shadow. The resulting transparent layers were then rendered via the painter's model onto the output device.

15.4 Defining and invoking a filter effect

Filter effects are defined by a **<filter>** element with an associated ID. Filter effects are applied to elements which have a **filter:** property which reference a **<filter>** element. Here is an example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs>
    <filter id="CoolTextEffect">
      <!-- Definition of filter goes here -->
    </filter>
  </defs>
  <text style="filter:url(#CoolTextEffect)">Text with a cool effect</text>
</svg>
```

Download this example

When applied to grouping elements such as 'g', the **filter:** property applies to the contents of the group as a whole. The effect is as if the group's children did not render to the screen directly but instead just added their resulting graphics primitives to the group's *graphics display list (GDL)*, which is then passed to the filter for processing. After the group filter is processed, then the result of the filter is rendered to the target device (or passed onto a parent grouping element for further processing in cases such as when the parent has its own group filter).

The **<filter>** element consists of a sequence of *processing nodes* which take a set of graphics primitives as input, apply

filter effects operations on the graphics primitives, and produce a modified set of graphics primitives as output. The processing nodes are executed in sequential order. The resulting set of graphics primitives from the final processing node (*feMerge* in the example below) represents the result of the filter.

Example filters02 renders some text with a shadowing effect.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
        "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="278px" height="118px">
  <defs>
    <filter id="Shadow">
      <feGaussianBlur in="SourceAlpha"
                stdDeviation="3"
                result="blurredAlpha" />
      <feOffset in="blurredAlpha"
                dx="2" dy="1"
                result="offsetBlurredAlpha" />
      <feDiffuseLighting in="blurredAlpha"
                diffuseConstant=".5"
                surfaceScale="5"
                resultScale="5"
                lightColor="white"
                result="bumpMapDiffuse" >
        <feDistantLight azimuth="135" elevation="60"/>
      </feDiffuseLighting>
      <feComposite in="bumpMapDiffuse" in2="SourceGraphic"
                operator="arithmetic" k1="1"
                result="litPaint" />
      <feSpecularLighting in="blurredAlpha"
                surfaceScale="5"
                specularConstant=".5"
                specularExponent="10"
                lightColor="white"
                result="bumpMapSpecular" >
        <feDistantLight azimuth="135" elevation="60"/>
      </feSpecularLighting>
      <feComposite in="litPaint" in2="bumpMapSpecular"
                operator="arithmetic" k2="1" k3="1"
                result="litPaint" />
      <feComposite in="litPaint"
                in2="SourceAlpha"
                operator="in"
                result="litPaint" />
      <feMerge>
        <feMergeNode in="offsetBlurredAlpha" />
        <feMergeNode in="litPaint" />
      </feMerge>
    </filter>
  </defs>
  <desc>Example filters02 - text with shadowing effect</desc>

  <text style="font-size:36px; fill:red; filter:url(#Shadow)"
                x="10px" y="70px">Shadowed Text</text>
</svg>
```

Example filters02

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

For most processing nodes, the **in** (and sometimes **in2**) attribute identifies the graphics which serve as input and the **result** attribute gives a name for the resulting output. The **in** and **in2** attributes can point to either:

- A built-in keyword. In the example above, the <feGaussianBlur> processing node specifies keyword *SourceGraphic*, which indicates that the original set of graphics primitives available at the start of the filter is used as input to the processing node.
- A reference to an earlier **result**. In the example above, the <feOffset> processing node refers to the most recent processing node whose **result** is *blurredAlpha*. (In this case, that would be the <feGaussianBlur> processing node.)

The default value for **in** is the output generated from the previous processing node. In those cases where the output from a given processing node is used as input only by the next processing node, it is not necessary to specify the **result** on the previous processing node or the **in** attribute on the next processing node. In the example above, there are a few cases (show highlighted) where **result** and **in** did not have to be provided.

Filters *do not* use XML IDs for **result**s; instead, **result**s can be any arbitrary attribute string value. **result**s only are meaningful within a given <filter> element and thus have only local scope. If a **result** appears multiple times within a given <filter> element, then a reference to that **result** will use the closest preceding processing node with the given **result**. Forward references to results are invalid.

The description of the 'filter' elements is as follows:

```
<!ENTITY % filterExt "" >
<!ELEMENT filter (feBlend|feFlood|
  feColorMatrix|feComponentTransfer|
  feComposite|feDiffuseLighting|feDisplacementMap|
  feGaussianBlur|feImage|feMerge|
  feMorphology|feOffset|feSpecularLighting|
  feTile|feTurbulence|
  animate|set
  %filterExt;)* >
<!ATTLIST filter
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  filterUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  filterRes CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >
```

Attribute definitions:

filterUnits = "*userSpace* | *userSpaceOnUse* | *objectBoundingBox*"

See Filter effects region.

x = "*x-coordinate*"

See Filter effects region.

y = "*y-coordinate*"

See Filter effects region.

width = "*length*"

See Filter effects region.

height = "*length*"

See Filter effects region.

filterRes = "*<number> [<number>]*"

See Filter effects region.

xlink:href = "<uri>"

A URI reference to another 'filter' element within the current SVG document fragment. Any attributes which are defined on the referenced 'filter' element which are not defined on this element are inherited by this element. If this

element has no defined filter nodes, and the referenced element has defined filter nodes (possibly due to its own href attribute), then this element inherits the filter nodes defined from the referenced 'filter' element. Inheritance can be indirect to an arbitrary level; thus, if the referenced 'filter' element inherits attribute or its filter node specification due to its own href attribute, then the current element can inherit those attributes or filter node specifications. Animatable: yes.

Attributes defined elsewhere:

id, xml:lang, xml:space, %xlinkAttrs;.

The description of the 'filter' property is as follows:

'filter'

Value:	<uri> | none
Initial:	none
Applies to:	graphics and container elements
Inherited:	no
Percentages:	N/A
Media:	visual

<uri>

A URI reference to a 'filter' element which defines the filter effects that shall be applied to this element.

15.5 Filter effects region

A <filter> element can define a region on the canvas on which a given filter effect applies and can provide a resolution for any intermediate continuous tone images used in to process any raster-based processing nodes. The <filter> element has the following attributes:

- **filterUnits={ userSpace | userSpaceOnUse | objectBoundingBox }**. Defines the coordinate system for attributes **x, y, width, height**.
  If **filterUnits="userSpace"** (the default), **x, y, width, height** and 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 defined.
  If **filterUnits="userSpaceOnUse"**, **x, y, width, height** and 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 **filterUnits="objectBoundingBox"**, then **x, y, width, height** and any length values within the filter definitions represent values in an abstract coordinate system where (0,0) is the (minx,miny) in user space of the tight bounding box of the object referencing the filter, and (1,1) is the (maxx,maxy) corner of the bounding box. (Note: the bounding box represents the maximum extent of the shape of the object in X and Y with respect to the user coordinate system of the object exclusive of stroke-width.)
- **x, y, width, height**, which indicate the rectangle for the largest possible offscreen buffer, where the values are either relative to the current user coordinate system (if filterUnits="userSpace") or relative to the current object (if filterUnits="target-object"). Note that the clipping path used to render any graphics within the filter will consists of the intersection of the current clipping path associated with the given object and the rectangle defined by **x, y, width, height**. The default values for **x, y, width, height** are 0%, 0%, 100% and 100%, respectively.
- **filterRes** (which has the form `x-pixels [y-pixels])` indicates the width/height of the intermediate images in pixels. If not provided, then a reasonable default resolution appropriate for the target device will be used. (For displays, an appropriate display resolution, preferably the current display's pixel resolution, is the default. For printing, an appropriate common printer resolution, such as 400dpi, is the default.)

For performance reasons on display devices, it is recommended that the filter effect region is designed to match pixel-for-pixel with the background.

It is often necessary to provide padding space because the filter effect might impact bits slightly outside the tight-fitting bounding box on a given object. For these purposes, it is possible to provide negative percentage values for **x, y** and percentages values greater than 100% for **width, height**. For example, x="-10%" y="-10%" width="120%"

height="120%".

15.6 Common attributes

The following two attributes are available for all processing nodes (the exception is feMerge and feImage, which do not have an **in** attribute):

Common Attributes	*result* Assigned name for this node. If supplied, then the GDL resulting after processing the given node is saved away and can be referenced as input to a subsequent processing node. *in* If supplied, then indicates that this processing node uses either the output of previous node as input or use some standard keyword to specify alternate input. (For the first processing node, the default *in* is SourceGraphic.) Available keywords representing pseudo image inputs include: • **SourceGraphic**. This built-in input represents the graphics elements that were the original input into the <filter> element. For raster effects processing nodes, 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**. Same as SourceGraphic except only the alpha channel is specified. The color channels of this image are implicitly black and are unaffected by any image processing operations. Again, pixels unpainted by the SourceGraphic will be 0. The SourceAlpha image will also reflects any Opacity settings in the 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 built-in input 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 specified. See Accessing the background image. • **FillPaint** This image represents the color data specified by the current SVG rendering state, transformed to image space. The FillPaint image has conceptually infinite extent in image space. (Since it is usually either just a constant color, or a tile). Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of an alpha gradient or transparent pattern. For the simple case where the source graphic represents a simple filled object, it is guaranteed that: `SourceGraphic = In(FillPaint,SourceAlpha)` where In(A,B) represents the resulting image of Porter-Duff compositing operation A in B (see below). • **StrokePaint** Similar to FillPaint, except for the stroke color as specified in SVG. Again for the simple case where the source graphic represents stroked path, it is guaranteed that: `SourceGraphic = In(StrokePaint,SourceAlpha)` where In(A,B) represents the resulting image of Porter-Duff compositing operation A in B (see below). *x*, *y*, *width*, *height*The sub-region which restricts calculation and rendering of the given filter node. These attributes would be defined according to the same rules as other filter effects coordinate and length attributes. These subregion attributes make 'feImage' consistent to the 'image' element and to provide enough

	information so that feTile can figure out how to stitch together tiles. x,y,width,height default to the union of the subregions defined for all referenced nodes. If there are no referenced nodes (e.g., for feImage or feTurbulence, which have no "in", or if in="SourceGraphic") or for feTile (which is special), the default subregion is 0%,0%,100%,100%, where percentages are relative to the dimensions of the filter region. x,y,width,height act as a hard clip. All intermediate offscreens are defined to not exceed the intersection of x,y,width,height with the filter region. The filter region and any of the x,y,width,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 of the x,y,width,height subregions. feImage scales the referenced image to fit exactly into x,y,width,height. feTile references a previous filter node and then stitches the tiles together based on the x,y,width,height values of the referenced filter node.

15.7 Accessing the background image

Two possible pseudo input images for filter effects are BackgroundImage and BackgroundAlpha, which each represent an image snapshot of the canvas under the filter region at the time that the <filter> element is invoked. BackgroundImage represents both the color values and alpha channel of the canvas (i.e., RGBA pixel values), whereas BackgroundAlpha represents only the alpha channel.

Implementations of SVG user agents often will need to maintain supplemental background image buffers in order to support the BackgroundImage and BackgroundAlpha pseudo input images. Sometimes, the background image buffers will contain an in-memory copy of the accumulated painting operations on the current canvas.

Because in-memory image buffers can take up significant system resources, SVG content must explicitly indicate to the SVG user agent that the document needs access to the background image before BackgroundImage and BackgroundAlpha pseudo input images can be used. The property which enables access to the background image is **'enable-background'**:

'enable-background'

Value:	accumulate | new [(<x> <y> <width> <height>)] | inherit
Initial:	accumulate
Applies to:	container elements
Inherited:	no
Percentages:	N/A
Media:	visual

'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 fully transparent) 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 fully transparent image shall be provided

in response to the request.

The optional **(<x>,<y>,<width>,<height>)** parameters on the **new** value indicate the sub-region of user space where access to the background image is allowed to happen. These parameters enable the SVG user agent potentially to allocate smaller temporary image buffers than the default values, which might require the SVG user agent to allocate buffers as large as the current viewport. Thus, the values <x>,<y>,<width>,<height> act as a clipping rectangle on the background image canvas.

15.8 Filter processing nodes

The following is a catalog of the individual processing nodes. Unless otherwise stated, all image filters operate on linear 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.

<table>
<tr><th>NodeType</th><th>feBlend</th></tr>
<tr><td>Processing Node-Specific Attributes</td><td>mode, One of the image blending modes (see table below). Default is: normal

in2, The second image ("B" in the formulas) for the compositing operation.</td></tr>
<tr><td>Description</td><td>This filter composites two objects together using commonly used high-end imaging software blending modes. Performs the combination of the two input images pixel-wise in image space.</td></tr>
<tr><td>Implementation Notes</td><td>The compositing formula, expressed using premultiplied colors:

<pre>qr = 1 - (1-qa)*(1-qb)
cr = (1-qa)*cb + (1-qb)*ca + qa*qb*(Blend(ca/qa,cb/qb))
where:
  qr = Result opacity
  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
  Blend = Image compositing function, depending on the compositing mode</pre>
The following table provides the list of available image blending modes:
<table>
<tr><td>Image Blending Mode</td><td>Blend() function</td></tr>
<tr><td>normal</td><td>Ca</td></tr>
<tr><td>multiply</td><td>(ca/qa)*(cb/qb)</td></tr>
<tr><td>screen</td><td>1-(1-(ca/qa))*(1-(cb/qb))</td></tr>
<tr><td>darken</td><td>(to be provided later)</td></tr>
<tr><td>lighten</td><td>(to be provided later)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>NodeType</th><th>feColorMatrix</th></tr>
<tr><td>Processing Node-Specific Attributes</td><td>type, string (one of: matrix, saturate, hueRotate, luminanceToAlpha)

values

For matrix: space-separated list of 20 element color transform (a00 a01 a02 a03 a04 a10 a11 ... a34). For example, the identity matrix could be expressed as:
<pre>type="matrix"
values="1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0"</pre>
For saturate: one real number value (0 to 1) or one percentage value (e.g., 50%)
For hueRotate: one real number value (degrees)
Not applicable for luminanceToAlpha
</td></tr>
</table>

Description

This filter performs

```
| 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 |
```

for every pixel. The RGBA and R'G'B'A' values are automatically *non-premultiplied* temporarily for this operation.

The following shortcut definitions are provide for compactness. The following tables show the mapping from the shorthand form to the corresponding longhand (i.e., matrix with 20 values) form:

```
saturate value (0..1)
   (can be expressed as a percentage value, such as "50%")


   | 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 |

hueRotate value (0..360)

   | 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:

      | a01 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) *  [-0.212 +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


luminanceToAlpha

   | R' |     |      0        0        0  0  0 |   | R |
   | G' |     |      0        0        0  0  0 |   | G |
   | B' |  =  |      0        0        0  0  0 | * | B |
   | A' |     |  0.299    0.587    0.114  0  0 |   | A |
   | 1  |     |      0        0        0  0  1 |   | 1 |
```

Implementation issues	These matrices often perform an identity mapping in the alpha channel. If that is the case, an implementation can avoid the costly undoing & redoing of the premultiplication for all pixels with A = 1.

NodeType	**feComponentTransfer**
Processing Node-Specific Attributes	None.
Processing Node-Specific Sub-Elements	Each 'feComponentTransfer' element needs to have at most one each of the following sub-elements, each of which is an empty element: *'feFuncR'*, transfer function for red component *'feFuncG'*, transfer function for green component *'feFuncB'*, transfer function for blue component *'feFuncA'*, transfer function for alpha component Each of these sub-elements (i.e., *'feFuncR'*, *'feFuncG'*, *'feFuncB'*, *'feFuncA'*) can have the following attributes: Common parameters to all transfer modes: *type*, string (one of: identity, table, linear, gamma) Parameters specific to particular transfer modes: For *table*: *tableValues*, list of real number values *v0,v1,...vn*. For *linear*: *slope*, real number value giving slope of linear equation. *intercept*, real number value giving Y-intercept of linear equation. For *gamma* (see descriptiong below for descriptions): *amplitude*, real number value. *exponent*, real number value. *offset*, real number value.

Description	This filter performs component-wise remapping of data as follows:
	`R' = feFuncR( R )`
	`G' = feFuncG( G )`
	`B' = feFuncB( B )`
	`A' = feFuncA( A )`
	for every pixel. The RGBA and R'G'B'A' values are automatically *non-premultiplied* temporarily for this operation.
	When *type="table"*, the transfer function consists of a linearly interpolated lookup table.
	k/N <= C < (k+1)/N => C' = vk + (C - k/N)*N * (vk+1 - vk)
	When *type="linear"*, the transfer function consists of a linear function describes by the following equation:
	C' = slope*C + offset
	When *type="gamma"*, the transfer function consists of the following equation:
	C' = amplitude*pow(C, exponent) + offset

Comments	This filter allows operations like brightness adjustment, contrast adjustment, color balance or thresholding. We might want to consider some predefined transfer functions such as identity, gamma, sRGB [SRGB] transfer, sine-wave, etc.

Implementation issues	Similar to the feColorMatrix filter, the undoing and redoing of the premultiplication can be avoided if feFuncA is the identity transform and A = 1.

NodeType	**feComposite**
Processing Node-Specific Attributes	*operator*, one of (*over, in, out, atop, xor, arithmetic*). Default is: over. *arithmetic-constants*, k1,k2,k3,k4 *in2*, The second image ("B" in the formulas) for the compositing operation.
Description	This filter performs the combination of the two input images pixel-wise in image space. *over, in, atop, out, xor* use the Porter-Duff compositing operations. For these operations, the extent of the resulting image can be affected. In other words, even if two images do not overlap in image space, the extent for over will essentially include the union of the extents of the two input images. *arithmetic* evaluates k1*i1*i2 + k2*i1 + k3*i2 + k4, using componentwise arithmetic with the result clamped between [0..1].
Comments	*arithmetic* are useful for combining the output from the feDiffuseLighting and feSpecularLighting filters with texture data. *arithmetic* is also useful for implementing *dissolve*.

NodeType	**feDiffuseLighting**
Processing Node-Specific Attributes	*resultScale* (Multiplicative scale for the result. This allows the result of the feDiffuseLighting nodeto represent values greater than 1) *surfaceScale* height of surface when Ain = 1. *diffuseConstant* kd in Phong lighting model. Range 0.0 to 1.0 *lightColor* RGB

Processing Node-Specific Sub-Elements	One of
	`<feDistantLight azimuth= elevation= >`
	`<fePointLight x= y= z= >`
	`<feSpotLight x= y= z= pointsAtX= pointsAtY= pointsAtZ= specularExponent=>`

	Light 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 caculation 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. Color or texture is mean to be applied via a multiply (mul) composite operation.
	`Dr = (kd * N.L * Lr) / resultScale`
	`Dg = (kd * N.L * Lg) / resultScale`
	`Db = (kd * N.L * Lb) / resultScale`
	`Da = 1.0 / resultScale`
	
	where
	
	kd = 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
	Lr,Lg,Lb = RGB components of light, a function of x and y in the spot light case
	resultScale = overall scaling factor

Description	N is a function of x and y and depends on the surface gradient as follows:
	
	The surface described by the input alpha image Ain (x,y) is:
	`Z (x,y) = surfaceScale * Ain (x,y)`
	
	Surface normal is calculated using the Sobel gradient 3x3 filter:
	`Nx (x,y)= - surfaceScale * 1/4*(( I(x+1,y-1) + 2*I(x+1,y) + I(x+1,y+1))`
	`- (I(x-1,y-1) + 2*I(x-1,y) + I(x-1,y+1)))`
	`Ny (x,y)= - surfaceScale * 1/4*(( I(x-1,y+1) + 2*I(x,y+1) + I(x+1,y+1))`
	`- (I(x-1,y-1) + 2*I(x,y-1) + I(x+1,y-1)))`
	`Nz (x,y) = 1.0`
	
	`N = (Nx, Ny, Nz) / Norm((Nx,Ny,Nz))`
	
	L, the unit vector from the image sample to the light is calculated as follows:
	
	For Infinite light sources it is constant:
	`Lx = cos(azimuth)*cos(elevation)`
	`Ly = -sin(azimuth)*cos(elevation)`
	`Lz = sin(elevation)`
	
	For Point and spot lights it is a function of position:
	`Lx = Lightx - x`
	`Ly = Lighty - y`
	`Lz = Lightz - Z(x,y)`
	
	`L = (Lx, Ly, Lz) / Norm(Lx, Ly, Lz)`
	
	where Lightx, Lighty, and Lightz are the input light position.
	
	Lr,Lg,Lb, 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. (Lr = Lg = Lb = 0)

Comments	This filter produces a light map, which 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.

NodeType	**feDisplacementMap**
Processing Node-Specific Attributes	*scale* *xChannelSelector* one of *R,G,B* or *A*. *yChannelSelector* one of *R,G,B* or *A* *in2*, The second image ("B" in the formulas) for the compositing operation.

Description	Uses Input2 to spatially displace Input1, (similar to the Photoshop displacement filter). 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 source image, Input1, and P'(x,y) is the destination. XC(x,y) and YC(x,y) are the component values of the designated by the *xChannelSelector* and *yChannelSelector*. For example, to use the R component of Image2 to control displacement in x and the G component of Image2 to control displacement in y, set *xChannelSelector* to "R" and *yChannelSelector* to "G".
Comments	The displacement map defines the inverse of the mapping performed.
Implementation issues	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 displacement in either x or y.

NodeType	**feFlood**
Processing Node-Specific Attributes	*style*, to specify the 'flood-color' and 'flood-opacity' properties (both non-inheritable) to specify an RGBA color
Description	Creates an image with infinite extent filled with *color*

NodeType	**feGaussianBlur**
Processing Node-Specific Attributes	*stdDeviation.*

Description	Perform gaussian blur on the input image.
	The Gaussian blur kernel is an appoximation of the normalized convolution:
	`H(x) = exp(-x2/ (2s2)) / sqrt(2* pi*s2)`
	where 's' is the standard deviation specified by 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 one pixel to the left, the second one centered one pixel to the right of the output pixel one box blur of size 'd+1' centered on the output pixel.

Implementation Issues	Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, SourceAlpha. The implementation may notice this and optimize the single channel case. If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

NodeType	**feImage**
Processing Node-Specific Attributes	*href,* reference to external image data. *transform,* supplemental transformation specification

Description	Refers to an external image which is loaded or rendered into an RGBA raster. If *imaging-matrix* is not specified, the image takes on its natural width and height and is positioned at 0,0 in image space. The imageref could refer to an external image, or just be a reference to another piece of SVG. This node produces an image similar to the builtin image source *SourceGraphic* except from an external source.

NodeType	**feMerge**
Processing Node-Specific Attributes	none
Processing Node-Specific Sub-Elements	Each 'feMerge' element can have any number of 'feMergeNode' subelements, each of which has an **in** attribute.
Description	Composites input image layers on top of each other using the *over* operator with *Input1* on the bottom and the last specified input, *InputN*, on top.
Comments	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 (see below).
Implementation issues	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.

NodeType	**feMorphology**
Processing Node-Specific Attributes	*operator,*one of *erode* or *dilate.* *radius,* extent of operation
Description	This filter is intended to have a similar effect as the min/max filter in Photoshop and the width layer attribute in ImageStyler. It is useful for "fattening" or "thinning" an alpha channel, The dilation (or erosion) kernel is a square of side 2**radius* + 1.
Implementation issues	Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, *SourceAlpha.* In that case, the implementation might want to optimize the single channel case. If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

NodeType	**feOffset**
Processing Node-Specific Attributes	*dx,dy*
Description	Offsets an image relative to its current position in the image space by the specified vector.
Comments	This is important for effects like drop shadow etc.

NodeType	

	feSpecularLighting
Processing Node-Specific Attributes	*surfaceScale* height of surface when Ain = 1. *specularConstant* ks in Phong lighting model. Range 0.0 to 1.0 *specularExponent* exponent for specular term, larger is more "shiny". Range 1.0 to 128.0. *lightColor* RGB

Processing Node-Specific Sub-Elements	One of
	`<feDistantLight azimuth= elevation= >`
	`<fePointLight x= y= z= >`
	`<feSpotLight x= y= z= pointsAtX= pointsAtY= pointsAtZ= specularExponent=>`

Description	Light an image using the alpha channel as a bump map. The resulting image is an RGBA image based on the light color. The lighting caculation 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. We assume that the viewer is at infinity the z direction (i.e the unit vector in the eye direction is (0,0,1) everywhere.
	
	`Sr = ks * pow(N.H, specularExponent) * Lr`
	`Sg = ks * pow(N.H, specularExponent) * Lg`
	`Sb = ks * pow(N.H, specularExponent) * Lb`
	`Sa = max(Sr, Sg, Sb)`
	
	`where`
	`ks = specular lighting constant`
	`N = surface normal unit vector, a function of x and y`
	`H = "halfway" unit vectorbetween eye unit vector and light unit vector`
	`Lr,Lg,Lb = RGB components of light`
	
	See feDiffuseLighting for definition of N and (Lr, Lg, Lb).
	
	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 specular result (Sr,Sg,Sb,Sa) 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.

Comments	This filter 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* Composite method. Multiple light sources can be simulated by adding several of these light maps before applying it to the texture image.
Implementation issues	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.

NodeType	**feTile**
Processing Node-Specific Attributes	none
Description	Creates an image with infinite extent by replicating source image in image space.

NodeType	feTurbulence
Processing Node-Specific Attributes	*baseFrequencyX* *baseFrequencyY* *numOctaves* *stitchTiles* *type,* one of*fractalNoise* or *turbulence*.
Description	Adds noise to an image using the Perlin turbulence-function. It is possible to create bandwidth-limited noise by synthesizing only one octave. For a detailed description the of the Perlin turbulence-function, see "Texturing and Modeling", Ebert et al, AP Professional, 1994. If the input image is infinite in extent, as is the case with a constant color or a tile, the resulting image will have maximal size in image space. If one of baseFrequencyX or baseFrequencyY attributes is not provided but the other is, then the missing attribute takes on the value from the other attribute. *stitchTiles* can take on the values "stitch" or "noStitch", where "noStitch" is the default. If stitchTiles="stitch", then automatically adjust baseFrequencyX such that the feTurbulence node's width (i.e., the width of the current subregion) contains an integral number of the Perlin tile width for the first octave. Do the corresponding adjustment for baseFrequencyY. The baseFrequency will be adjusted up or down depending on which way has the smallest relative (not absolute) change. Here's how: 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.
Comments	This filter allows the synthesis of artificial textures like clouds or marble.
Implementation issues	It might be useful to provide an actual implementation for the turbulence function, so that consistent results are achievable.

15.9 DOM interfaces

15.9.1 Interface SVGFilterElement

The SVGFilterElement interface corresponds to the 'filter' element.

```
interface SVGFilterElement : SVGElement {
  // filterUnit Types
  const unsigned short kSVG_FILTERUNITS_UNKNOWN           = 0;
  const unsigned short kSVG_FILTERUNITS_USERSPACE         = 1;
  const unsigned short kSVG_FILTERUNITS_USERSPACEONUSE    = 2;
  const unsigned short kSVG_FILTERUNITS_OBJECTBOUNDINGBOX = 3;
  attribute  unsigned short filterUnits;

  attribute SVGLength x;
  attribute SVGLength y;
  attribute SVGLength width;
  attribute SVGLength height;
  attribute SVGNumber filterRes;
};
```

15.9.2 Interface SVGStandardFilterNodeElement

The SVGStandardFilterNodeElement interface is the base interface for the DOM interfaces for the most of the elements that can be children of a 'filter' element.

```
interface SVGStandardFilterNodeElement : SVGElement {
  attribute DOMString in;
  attribute DOMString result;
};
```

??? Need to do all of the filter elements

??? Special interface for filter property might be necessary. feColorMatrix will be complicated. tableValues will be complicated

---

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) or keyboard events 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 enableZoomAndPanControls 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 user interface events, including under which circumstances user interface 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 User interface events

On user agents which support interactivity, it is common for authors to define SVG document such that they are responsive to user interface events. Among the set of possible user events are pointer events, keyboard events, and document events.

In response to user interface (UI) events, the author might start an animation, perform a hyperlink to another web page, highlight part of the document (e.g., change the color of the graphics elements which are under the pointer), initiate a "roll-over" (e,g., cause some previously hidden graphics elements to appear near the pointer) or launch a script which communicates with a remote database.

For all UI event-related features defined as part of the SVG language via event attributes or animation, the event model corresponds to the *event bubbling* model described in DOM2 [DOM2-EVBUBBLE]. The *event capture* model from DOM2 [DOM2-EVCAPTURE] can only be established from DOM method calls.

16.3 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 graphics content is under the pointer at the time of the event. (See property 'pointer-events' for a description of how to determine the situations in which a graphic element receives pointer events.)

The event is either initially dispatched to the *target element*, to one of the target element's ancestors, or not dispatched, depending on the following:

- If there are no graphics elements whose relevant graphics content is under the pointer (i.e., there

is no target element), the event is not dispatched.

- Otherwise, there is a target element. If there is an ancestor of the target element which has specified an event handler with event capturing [DOM2-EVCAPTURE] for the given event, then the event is dispatched to that ancestor element.
- Otherwise, if the target element has an appropriate event handler for the given event, the event is dispatched to the target element.
- Otherwise, each ancestor of the target element (starting with its immediate parent) is checked to see if it has an appropriate event handler. If an ancestor is found with an appropriate event handler, the event is dispatched to that ancestor element.
- Otherwise, the event is discarded.

When event bubbling [DOM2-EVBUBBLE] is active, descendant elements receive events before their ancestors. Thus, if a 'path' element is a child of a 'g' element and they both have event listeners for **click** events, then the event will be dispatched to the 'path' element before the 'g' element.

When event capturing [DOM2-EVCAPTURE] is active, ancestor elements receive events before their descendants.

After an event is initially dispatched to a particular element, unless an appropriate action has been taken to prevent further processing (e.g., by invoking the preventCapture() or preventBubble() DOM method call), the event will be passed to the appropriate event handlers (if any) for that element's ancestors (in the case of event bubbling) or that element's descendants (in the case of event capture) for further processing.

16.4 Processing order for user interface events

The processing order for user interface events is as follows:

- Event handlers assigned to the topmost graphics element under the pointer (and the various ancestors of that graphics element) receive the event first. If none of the activation event handlers take an explicit action to prevent further processing of the given event (e.g., by invoking the preventDefault() DOM method), then the event is passed on for:
- Processing of any relevant CSS2's dynamic pseudo-classes (i.e., :hover, :active and :focus) [CSS2-DYNPSEUDO], after which the event is passed on for:
- (For those user interface events which invoke hyperlinks, such as mouse clicks in some user agents) Link processing. If a hyperlink is invoked in response to a user interface event, the hyperlink typically will disable further activation event processing (e.g., often, the link will define a hyperlink to another web page). If link processing does not disable further processing of the given event, then the event is passed on for:
- (For those user interface events which can select text, such as mouse clicks and drags on 'text' elements) Text selection processing. When a text selection operation occurs, typically it will disable further processing of the given event; otherwise, the event is passed on for:
- Document-wide event processing, such as user agent facilities to allow zooming and panning of an SVG document fragment.

16.5 The 'pointer-events' property

In different circumstances, authors may or may not want events to be triggered when the pointer is over the unfilled interior of a graphics element or the pointer is over an invisible graphics element. The 'pointer-events' property specifies under what circumstances a given graphics element receive pointer events.

'pointer-events'

Value:	visiblePainted | visibleFill | visibleStroke | visibleFillStroke | visible | painted | fill | stroke | fillstroke | all | none | inherit
Initial:	visiblePainted
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

visiblePainted

The given element receives pointer events when the 'visibility' property is set to visible and when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the 'fill' property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the 'stroke' property is set to a value other than 'none'.

visibleFill

The given element receives pointer events when the 'visibility' property is set to visible and when the pointer is over the interior (i.e., fill) of the element. The value of the 'fill' property does not effect event processing.

visibleStroke

The given element receives pointer events when the 'visibility' property is set to visible and when the pointer is over the perimeter (i.e., stroke) of the element. The value of the 'stroke' property does not effect event processing.

visibleFillStroke

The given element receives pointer events when the 'visibility' property is set to visible and when 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' properties do not effect event processing.

visible

The given element receives pointer in all cases when the 'visibility' property is set to visible. The values of the 'fill' and 'stroke' do not effect event processing.

painted

The given element receives pointer events when the pointer is over a "painted" area. The pointer is over a painted area if it is over the interior (i.e., fill) of the element and the 'fill' property is set to a value other than 'none' or it is over the perimeter (i.e., stroke) of the element and the 'stroke' property is set to a value other than 'none'. The value of the 'visibility' property does not effect event processing.

fill

The given element receives pointer events when the pointer is over the interior (i.e., fill) of the

element. The values of the 'fill' and 'visibility' properties do not effect event processing.

stroke

The given element receives pointer events when the pointer is over the perimeter (i.e., stroke) of the element. The values of the 'stroke' and 'visibility' properties do not effect event processing.

fillStroke

The given element receives pointer events when the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element. The values of the 'fill', 'stroke' and 'visibility' properties do not effect event processing.

all

The given element receives pointer in all cases. The values of the 'fill', 'stroke' and 'visibility' properties do not effect event processing.

none

The given element does not receive pointer events.

For text elements, hit detection is performed on a character cell basis. The values visiblePainted, visibleFill, visibleStroke and visibleFillStroke are all defined to be equivalent to the value visible, and the values painted, fill, stroke and fillStroke are all defined to be equivalent to the value all.

For raster elements, hit detection can be defined to be dependent on whether the pixel under the pointer is fully transparent. For any of the values visiblePainted, visibleFill, visibleStroke and visibleFillStroke, the raster element receives the event if the 'visibility' property is set to visible and the pixel under the pointer is not fully transparent. For a value of visible, the raster element receives the event if the 'visibility' property is set to visible even if the pixel under the pointer is fully transparent. For any of the values painted, fill, stroke and fillStroke, the raster element receives the event if the the pixel under the pointer is not fully transparent, no matter what the value is for the 'visibility' property. For a value of all, the raster element receives the event even if the pixel under the pointer is fully transparent, no matter what the value is for the 'visibility' property.

16.6 Zooming panning and magnification

Zooming represents a (potentially) non-uniform scale transformation on an SVG document fragment in response to a user interface action. All elements which are specified in user coordinates will scale uniformly, but elements which use CSS unit identifiers to define coordinates or lengths may be transformed differently. A zoom operation has the effect of a supplemental scale and translate transformation inserted into the transformation hierarchy between the outermost 'svg' element and its children, as if an extra 'g' element enclosed all of the children and that 'g' element specified a transformation to achieve the desired zooming effect.

Panning represents a translation (i.e., a shift) transformation on an SVG document fragment in response to a user interface action.

Magnification represents complete, uniform transformation on an 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)..

Some ability to zoom and pan SVG document fragments are required for SVG user agents that operate

in interaction-capable user environments. Document-level magnification capabilities are recommended for SVG user agents to enable accessibilty to those who are partially visually impaired.

The outermost 'svg' element in an SVG document fragment has attribute enableZoomAndPanControls, which takes the possible values of *true*and *false*, with the default being *true*. If true, in environments that support user interactivity, the user agent shall provide user interface controls to allow the user to zoom in, zoom out and pan around the given document fragment. If false, the user agent shall disable these controls and not allow the user to zoom and pan on the given document fragment. If a enableZoomAndPanControls attribute is assigned to an inner 'svg' element, the enableZoomAndPanControls setting on the inner 'svg' element will have no effect on the SVG user agent.

Animatable: no.

16.7 Cursors

16.7.1 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 references via a <uri> and can point to either an external resource such as a platform-specific cursor file or to a 'cursor' element, which can be used to define a platform-independent cursor.

16.7.2 The 'cursor' property

'cursor'

Value:	[[<uri> ,]* [auto | crosshair | default | pointer | move| e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help]] | inherit
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual, interactive
Animatable:	yes

This property specifies the type of cursor to be displayed for the pointing device. Values have the following meanings:

auto

The UA determines the cursor to display based on the current context.

crosshair

A simple crosshair (e.g., short line segments resembling a "+" sign).

default

The platform-dependent default cursor. Often rendered as an arrow.

pointer

The cursor is a pointer that indicates a link.

move

Indicates something is to be moved.

e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize

Indicate that some edge is to be moved. For example, the 'se-resize' cursor is used when the movement starts from the south-east corner of the box.

text

Indicates text that can be selected. Often rendered as an I-bar.

wait

Indicates that the program is busy. Often rendered as a watch or hourglass.

help

Help is available for the object under the cursor. Often rendered as a question mark or a balloon.

<uri>

The user agent retrieves the cursor from the resource designated by the URI. If the user agent cannot handle the first cursor of a list of cursors, it shall. attempt to handle the second, etc. If the user agent cannot handle any user-defined cursor, it must use the generic cursor at the end of the list.

Example(s):

```
P { cursor : url("mything.cur"), url("second.csr"), text; }
```

The 'cursor' property for SVG is identical to the 'cursor' property defined in the "Cascading Style Sheets (CSS) level 2" specification [CSS2], with the exception that SVG user agents must support cursors defined by the 'cursor' element.

16.7.3 The 'cursor' element

The 'cursor' element can be used to define a platform-independent custom cursor. A recommended approach for defining a platform-independent custom cursor is to create a PNG [PNG01] image and define a 'cursor' element that references the PNG image and identifies the exact position within the image which is the pointer position (i.e., the hot spot).

```
<!ELEMENT cursor (%descTitle;) >
<!ATTLIST cursor
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA "0"
  y CDATA "0"
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >
```

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.
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.
Animatable: yes.

xlink:href = "<uri>"

A URI reference to the file or element which provides the image of the cursor.
Animatable: yes.

Attributes defined elsewhere:

id, system-required, system-language, %xlinkAttrs;.

SVG user agents are required to support PNG format images as targets of the xlink:href property.

16.8 DOM interfaces

16.8.1 Interface SVGCursorElement

The SVGCursorElement interface corresponds to the 'cursor' element.

```
interface SVGCursorElement : SVGElement {
  attribute SVGLength x;
  attribute SVGLength y;

  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
};
```

16.8.2 Interface SVGViewElement

The SVGViewElement interface corresponds to the 'view' element.

```
interface SVGViewElement : SVGElement {
  // Object-oriented access
  SVGRect viewBox;
  SVGPreserveAspectRatio preserveAspectRatio;
  boolean enableZoomAndPanControls;
  SVGElement viewTarget;
};
```

---

17 Linking

Contents

17.1 Links out of SVG content: the 'a' element

SVG provides an **'a'** element, analogous to like HTML's 'a' element, to indicate hyperlinks; those parts of the drawing which when clicked on will cause the current browser frame to be replaced by the contents of the URL specified in the *href* attribute.

The 'a' element uses Xlink. (Note that the XLink specification is currently under development and is subject to change. The SVG working group will track and rationalize with XLink as it evolves.)

The following is a valid example of a hyperlink attached to a path (which in this case draws a triangle):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <desc>This valid svg document draws a triangle which is a hyperlink
  </desc>
  <a xlink:href="http://www.w3.org">
    <path d="M 0 0 L 200 0 L 100 200 z"/>
  </a>
</svg>
```

Download this example

This is the well-formed equivalent example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <desc>This well formed svg document draws a triangle which is a hyperlink
  </desc>
  <a xmlns:xlink="http://www.w3.org/XML/XLink/0.9"
     xlink:type="simple" xlink:show="replace" xlink:actuate="user"
     xlink:href="http://www.w3.org">
    <path d="M 0 0 L 200 0 L 100 200 z"/>
  </a>
</svg>
```

Download this example

In both examples, if the path is clicked on, then the current browser frame will be replaced by the W3C home page.

```
<!ENTITY % aExt "" >
<!ELEMENT a (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%aExt;)*) >
<!ATTLIST a
  id ID #IMPLIED
  xmlns:xlink CDATA #FIXED "http://www.w3.org/XML/XLink/0.9"
  xlink:type (simple|extended|locator|arc) #FIXED "simple"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|embed|replace) 'replace'
  xlink:actuate (user|auto) #FIXED 'user'
  xlink:href CDATA #REQUIRED
  target CDATA #IMPLIED >
```

xmlns [:*prefix*] = "*resource-name*"

Standard XML attribute for identifying an XML namespace. This attribute makes the XLink [XLink] namespace available to the current element. Refer to the "Namespaces in XML" Recommendation [XML-NS].

Animatable: no.

xlink:type = 'simple'

Identifies the type of XLink being used. For hyperlinks in SVG, only simple links are available. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:role = '<string>'

A generic string used to describe the function of the link's content. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:title = '<string>'

Human-readable text describing the link. Refer to the "XML Linking Language (XLink)" [XLink].

Animatable: no.

xlink:show = 'replace'

Indicates that upon activation of the link the referenced document will. replace the entire contents of the current document. Refer to the "XML Linking Language (XLink)" [XLink].
Animatable: no.

xlink:actuate = 'user'

Indicates that the contents of the referenced object are incorporated into the current document upon user action. Refer to the "XML Linking Language (XLink)" [XLink].
Animatable: no.

xlink:href = "<uri>"

The location of the referenced object, expressed as a URI reference. Refer to the "XML Linking Language (XLink)" [XLink].
Animatable: yes.

target = "*<frame-target>*"

This attribute has applicability when the current SVG document is used as part of an HTML [HTML40] or XHTML [XHTML10] parent document which defines multiple frames. This attribute specifies the name of an HTML or XHTML frame into which a document is to be opened when the hyperlink is activated. For more information, refer to the appropriate HTML or XHTML specifications.
Animatable: yes.

17.2 Linking into SVG content: URI fragments and SVG views

17.2.1 Introduction: URI fragments and SVG views

On the Internet, resources are identified using URIs (Uniform Resource Identifiers) [URI]. For example, an SVG file called MyDrawing.svg located at http://www.MyCompany.com might have the following URI:

```
http://www.MyCompany.com/MyDrawing.svg
```

A URI can also address a particular element within an XML document by including a URI fragment identifier as part of the URI. A URI which includes a URI fragment identifier consists of an optional base URI, followed by a "#" character, followed by the URI fragment identifier. For example, the following URI can be used to specify the element whose ID is "Lamppost" within file MyDrawing.svg:

```
http://www.MyCompany.com/MyDrawing.svg#Lamppost
```

Because SVG content often represent a picture or drawing of something, a common need is to link into a particular view of the document, where a view indicates the initial transformations so as to present a closeup of a particular section of the document.

17.2.2 SVG fragment identifiers

To link into a particular view of an SVG document, the URI fragment identifier needs to be a correctly formed SVG fragment identifier. An SVG fragment identifier, which defines the meaning of the "selector" or "fragment identifier" portion of URIs that locate resources of MIME media type "image/svg".

An SVG fragment identifier can come in three forms:

- Shorthand *bare name* form of addressing (e.g., MyDrawing.svg#MyView). This form of addressing, which allows addressing an SVG element by its ID, is compatible with the fragment addressing mechanism for older versions of HTML and the shorthand bare name formulation in "XML Pointer Language (XPointer)" [XPTR]. (The bare name form of addressing #MyElement is equivalent to the XPointer formulation #xptr(id('MyView')).)
- XPointer-compatible ID reference (e.g., MyDrawing.svg#xptr(id('MyView'))). This form of addressing, which also allows addressing an SVG element by its ID, is compatible with the syntax for referencing IDs in "XML Pointer Language (XPointer)" [XPTR].
- 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(...), enableZoomAndPanControls(...) and viewTarget(...), whose parameters have the same meaning as the corresponding attributes on a 'view' element.

An SVG fragment identifier is defined as follows:

```
SVGFragmentIdentifier ::= BareName |
                          XPointerIDRef |
                          SVGViewSpec

BareName ::= XML_Name

XPointerIDRef ::= 'xptr(id(' XML_Name '))'

SVGViewSpec ::= 'svgView(' SVGViewAttributes ')'

SVGViewAttributes ::= SVGViewAttribute |
                      SVGViewAttribute ';' SVGViewAttributes

SVGViewAttribute ::= viewBoxSpec |
                     preserveAspectRatioSpec |
                     transformSpec |
                     enableZoomAndPanControlsSpec |
                     viewTargetSpec

viewBoxSpec ::= 'viewBox(' X ',' Y ',' Width ',' Height ')'

X ::= Number

Y ::= Number

Width ::= Number

Height ::= Number

preserveAspectRatioSpec = 'preserveAspectRatio(' AspectParams ')'

AspectParams ::= AspectValue |
                 AspectValue ',' MeetOrSlice
```

```
AspectValue ::= 'none' | 'xMinYMin' | 'xMinYMid' | 'xMinYMax' |
                'xMidYMin' | 'xMidYMid' | 'xMidYMax' |
                'xMaxYMin' | 'xMaxYMid' | 'xMaxYMax'

MeetOrSlice ::= 'meet' | 'slice'


Height ::= Number

transformSpec ::= 'transform(' TransformParams ')'

transformSpec ::= 'enableZoomAndPanControls(' TrueOrFalse ')'

TrueOrFalse ::= 'true' | 'false'

viewTargetSpec ::= 'viewTarget(' XML_Name ')'
```

where:

- XML_Name is an XML name (i.e., matches the name formulation rules in XML 1.0).
- Number is a real number.
- The parameter values for viewBoxSpec corresponds to to the parameter values for the viewBox attribute on the 'svg' element. For example, viewBox(0,0,200,200).
- The parameter values for preserveAspectRatioSpec corresponds to to the parameter values for the preserveAspectRatio attribute on the 'svg' element. For example, preserveAspectRatio(xMidYMid).
- The parameter values for transformSpec corresponds to to the parameter values for the transform attribute that is available on many SVG elements. For example, transform(matrix(2 0 0 2 10 15)).
- The parameter values for transformSpec corresponds to to the parameter values for the transform attribute that is available on many SVG elements. For example, transform(matrix(2 0 0 2 10 15)).
- The parameter values for enableZoomAndPanControlsSpec corresponds to to the parameter values for the enableZoomAndPanControls attribute on the 'svg' element. For example, enableZoomAndPanControls(false).
- The parameter values for viewTargetSpec corresponds to to the parameter values for the viewTarget attribute on the 'view' element. For example, viewTarget(MyElementID).

Spaces are not allowed in fragment specifications; thus, commas are. used to separate numeric values within an SVG view specification (e.g., #svgView(viewBox(0,0,200,200))) and semicolons are. used to separate attributes (e.g., #svgView(viewBox(0,0,200,200);preserveAspectRatio(none))).

When a source document performs a hyperlink into an SVG document via an HTML [HTML40] linking element (i.e., <a href=...> element in HTML) or an XLink specification [XLINK], then the SVG fragment identifier specifies the initial view into the SVG document, as follows:

- If no SVG fragment identifier is provided (e.g, the specified URI did not contain a "#" character, such as MyDrawing.svg), then the initial view into the SVG document is established using the view specification attributes (i.e., viewBox, etc.) on the outermost 'svg' element.
- If the SVG fragment identifier addresses a 'view' element within an SVG document (e.g., MyDrawing.svg#MyView or MyDrawing.svg#xptr(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 any element other than a 'view' element, then the document defined by the closest ancestor 'svg' element is displayed in the viewport using the view specification attributes on that 'svg' element.

17.2.3 Predefined views: the 'view' element

The 'view' element is defined as follows:

```
<!ENTITY % viewExt "" >
<!ELEMENT view (%descTitle;%viewExt;) >
<!ATTLIST view
  id ID #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  enableZoomAndPanControls (true | false) "true"
  viewTarget CDATA #IMPLIED >
```

Attribute definitions:

viewTarget = "*XML_Name [XML_NAME]**"

Indicates the target object associated with the view. If provided, then the target element(s) will. be highlighted.
Animatable: no.

Attributes defined elsewhere:

class, viewBox, preserveAspectRatio, enableZoomAndPanControls.

17.3 DOM interfaces

17.3.1 Interface SVGAElement

The SVGAElement interface corresponds to the 'a' element.

```
interface SVGAElement : SVGElement {
  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
  attribute DOMString target;
};
```

17.3.2 Interface SVGViewSpec

This interface corresponds to an SVG View Specification.

```
interface SVGViewSpec {
  // Object-oriented access
  SVGRect viewBox;
  SVGPreserveAspectRatio preserveAspectRatio;
  SVGTransformList transform;
  boolean enableZoomAndPanControls;
  SVGElement viewTarget;

  // String-oriented access.
  // (Necessary because this information is part
  // of the original document and thus not available
  // in XML DOM.)
  DOMString viewBoxString;
  DOMString preserveAspectRatioString;
  DOMString transformString;
  DOMString enableZoomAndPanControlsString;
  DOMString viewTargetString;
};
```

---

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 all scripts in the given document.

. contentScriptType = "<contentType>"

Identifies the default scripting language for all scripts in the given document. The term contentType has the same meaning and usage as the term "content type" has in the HTML 4.0 Specification [HTML40].

Animatable: no.

In the absense of a contentScriptType attribute, the default can be set by a "Content-Script-Type" HTTP header:

```
Content-Script-Type: <contentType>
```

User agents shall determine the default scripting language for an SVG document fragment according to the following steps (highest to lowest priority):

1. If a contentType attribute is provided on the 'svg' element, then the value of that attribute determines the default scripting language.

2. Otherwise, if any HTTP headers specify the "Content-Script-Type", the last one in the character stream determines the default scripting language.

Documents that do not specify a default scripting language shall set the default scripting lanuage to "text/ecmascript".

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 language attribute on the 'script' element.

18.2 The 'script' element

A **'script'** element can appear as a subelement to any 'defs' 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.

The following is an example of defining an ECMAScript function and defining an event handler that invokes that function:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in">
  <defs>
    <script><![CDATA[
      /* Beep on mouseclick */
      MouseClickHandler() { beep(); }
     ]]>
    </script>
  </defs>
  <circle onclick="MouseClickHandler()" r="85"/>
</svg>
```

Download this example

```
<!ELEMENT script (#PCDATA)* >
<!ATTLIST script
  language CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >
```

Attribute definitions:

language = "<contentType>"

Identifies the scripting language for the given 'script' element. The term contentType has the same meaning and usage as the term "content type" has in the HTML 4.0 Specification [HTML40]. If this attribute is not provided, the default scripting language is set as described under Specifying the default scripting language.
Animatable: no.

Attributes defined elsewhere:

%xlinkAttrs;, href.

18.3 Event handling

Events can cause scripts to execute when either of the following has occurred:

- Event attributes such as "onclick" or "onload" are assigned to particular elements, where the value of the event attributes contains script which is executed when the given event occurs.
- Event listeners as described in [DOM2-EVENTS] are defined which are invoked when a given event happens on a given object

Related sections of the spec:

- User interface events describes how an SVG user agent handles events such as pointer movements events (e.g., mouse movement) and activation events (e.g., mouse click).
- Relationship with DOM2 events describes what parts of DOM are supported by SVG and how to register event listeners

18.4 Event attributes

The following event attributes are available on many SVG elements, including its graphics elements and its container elements.

Mouse Events

- **onfocusin**
- **onfocusout**
- **ongainselection**
- **onloseselection**
- **onactivate**
- **onmousedown**
- **onmouseup**
- **onclick**
- **ondblclick**
- **onmouseover**
- **onmousemove**
- **onmouseout**

Animatable: no.

Keyboard Events

- **onkeydown**
- **onkeypress**

- **onkeyup**

Animatable: no.

State Change Events

- **onload** (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 such as images and style sheets are not necessarily loaded, parsed and ready to render)
- **onresize** (only applicable to outermost 'svg' elements which are to be mapped into a rectangular region/viewport. Corresponds to DOM level 2 resize event.)
- **onscroll** (only applicable to outermost 'svg' elements which are to be mapped into a rectangular region/viewport. Corresponds to DOM level 2 scroll event.)
- **onunload** (only applicable to outermost 'svg' elements which are to be mapped into a rectangular region/viewport)
- **onzoom** (only applicable to outermost 'svg' elements which are to be mapped into a rectangular region/viewport)
- **onerror** (corresponds to DOM level 2 error event)
- **onabort** (corresponds to DOM level 2 abort event)

Animatable: no.

A **load** event is dispatched only to the element to which the event applies; it is not dispatched to its ancestors. For example, if an 'image' element and its parent 'g' element both have event listeners for **load** events, when the 'image' element has been loaded, only its event listener will be invoked. (The 'g' element's event listener will indeed get invoked, but the invocation will happen when the 'g' itself has been loaded.)

Additionally, SVG's scripting engine needs to have the *altKey*, *ctrlKey* and *shiftKey* properties available.

18.5 DOM interfaces

18.5.1 Interface SVGScriptElement

The SVGScriptElement interface corresponds to the 'script' element.

```
interface SVGScriptElement : SVGElement  {
  attribute DOMString language;

  attribute DOMString role;
  attribute DOMString title;
  attribute DOMString show;
  attribute DOMString actuate;
  attribute DOMString href;
  attribute DOMString target;
};
```

18.5.2 Interface SVGZoomEvent

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.

```
interface SVGZoomEvent : UIEvent  {
  // Information about the specified zoom rectangle in screen units.
  attribute SVGRect zoomRectScreen;

  // Information about the previous zoom and pan factors
  attribute float previousScale;
  attribute SVGPoint previousTranslate;

  // Information about the new zoom and pan factors which will
  // be applied upon normal return from the event handler.
  attribute float newScale;
  attribute SVGPoint newTranslate;
};
```

The UI event type for a zoom event is:

zoom

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.

- Bubbles: Yes
- Cancelable: No
- Context Info: zoomRectScreen, previousScale, previousTranslate, newScale, newTranslate, screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedNode.
  (screenX, screenY, clientX, clientY indicate the center of the zoom area, with clientX, clientY in viewport coordinates for the corresponding 'svg' element. relatedNode is the corresponding 'svg' element.)

---

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 do motion paths, fade-in/fade-out effects and objects that grow, shrink, spin or change color.
- Using the SVG DOM. The SVG DOM conforms to key aspects of the "Document Object Model (DOM) level 1" [DOM1] and "Document Object Model (DOM) level 2" [DOM2] specifications. Every attribute and style sheet setting is accessible to scripting, and SVG offers a set of additional DOM interfaces to support efficient animation via scripting. As a result, virtually any kind of animation can be achieved.

The timer facilities in scripting languages such as ECMAScript can be used to start up and control the animations. (See example below.)

- SVG has been designed to allow future versions of SMIL [SMIL1] to use animated or static SVG content as media components.
- In the future, it is expected that future versions of SMIL will be modularized and that components of it could be used in conjunction with SVG and other XML grammars to achieve animation effects.

19.2 Animation elements

19.2.1 Relationship to SMIL Animation

SVG's animation elements were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [SMIL1].

The SYMM working group, in collaboration with the SVG working group, has authored the SMIL Animation specification [SMILAnim], which represents a general-purpose XML animation feature set. SVG incorporates the animation features defined in the SMIL Animation specification and provides some SVG-specific extensions.

SVG supports the following four animation elements which are defined in the SMIL Animation specification:

'animate'	allows scalar attributes and properties to be assigned different values over time
'set'	a convenient shorthand for 'animate', which is useful for assigning animation values to non-numeric attributes and properties, such as the 'visibility' property
'animateMotion'	moves an element along a motion path
'animateColor'	modifies the color value of particular attributes or properties over time

Additionally, SVG includes the following compatible extensions to SMIL Animation:

'animateTransform'	modifies one of SVG's transformation attributes over time, such as the transform attribute
path attribute	SVG allows the 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.)
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

SMIL Animation requires that the host language define how identify the elements which are to be animated. For compatibility with other aspects of the language, SVG uses URI references via an xlink:href attribute to identify the elements which are to be targets of the animations.

SMIL Animation requires that the host language define the meaning for document begin and the document end. Since an 'svg' is sometimes the root of the XML document tree and other times can be a component of a parent XML grammar, SVG defines an effective begin and effective end for an SVG document fragment. The effective begin of an SVG document fragment is the exact time at which the 'svg' element's onload event is triggered. The effective 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.

The term presentation time indicates the effective position in time relative to the effective begin of a document fragment. Presentation time behaves like the timecode shown on a counter of a tape-deck that advances at the speed of the presentation. It reflects that the presentation can be stopped, and that its speed can be decreased or increased either by user actions, or by the animation engine itself.

SVG defines more constrained error processing that is defined in the SMIL Animation [SMILAnim] specification. SMIL Animation defines error processing behavior where the document continues to run in certain error situations, whereas all animations within an SVG document fragment will stop in the event of any error within the document (see Error processing).

The SMIL Animation specification was jointly developed by the SYMM and SVG working groups. SVG is a host language in terms of SMIL Animation and therefore introduces additional constraints and features as permitted by that specification. Except as specifically noted, all animation elements and attributes described for SVG conform to the SMIL Animation [SMILAnim] specification.

19.2.2 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 December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="8cm" height="3cm"  viewBox="0 0 800 300">
  <desc>Example anim01 - demonstrate animation elements</desc>

  <!-- 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"
       style="fill:rgb(255,255,0)" >
    <animate attributeName="x" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="300" to="0" />
    <animate attributeName="y" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="100" to="0" />
    <animate attributeName="width" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="300" to="800" />
    <animate attributeName="height" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="100" to="300" />
  </rect>

  <!-- Set up a new user coordinate system so that
       the text string's origin is at (0,0), allowing
       rotation and scale relative to the new origin -->
  <g transform="translate(100,100)" >
    <!-- The following illustrates the use of the 'set', 'animateMotion',
         'animateColor' and 'animateTransform' elements. The 'text' element
         below starts off hidden (i.e., invisible). At 3 seconds, it:
           * becomes visible
           * continuously moves diagonally across the viewport
           * changes color from blue to dark red
           * rotates from -30 to zero degrees
           * scales by a factor of three. -->
    <text id="TextElement" x="0" y="0"
          style="font-family:Verdana; font-size:35.27; visibility:hidden" >
      It's alive!
      <set attributeName="visibility" attributeType="CSS" to="visible"
           begin="3s" dur="6s" fill="freeze" />
      <animateMotion path="M 0 0 L 100 100"
           begin="3s" dur="6s" fill="freeze" />
      <animateColor attributeName="fill" attributeType="CSS"
           from="rgb(0,0,255)" to="rgb(128,0,0)"
           begin="3s" dur="6s" fill="freeze" />
      <animateTransform attributeName="transform" attributeType="XML"
```

```
            type="rotate" from="-30" to="0"
            begin="3s" dur="6s" fill="freeze" />
        <animateTransform attributeName="transform" attributeType="XML"
            type="scale" from="1" to="3"
            begin="3s" dur="6s" fill="freeze" />
    </text>
  </g>
</svg>
```

At zero seconds

At three seconds

At six seconds

At nine seconds

Example anim01

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

The sections below describe the various animation attributes and elements.

19.2.3 Attributes to identify the target of an animation

The following attributes are common to all animation elements and identify the target element and the attribute or property whose value changes over time.

```
<!ENTITY % animTargetAttrs
 "%xlinkRefAttrs;
  xlink:href CDATA #IMPLIED
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED" >
```

Attribute definitions:

xlink:href = "<uri>"

A URI reference to the element which is the target of this animation and which therefore will be modified over time.
The target element must be part of the current SVG document fragment.
<uri> must point to exactly one target element which is capable of being the target of the given animation. If <uri> points to multiple target elements or the given target element is not capable of being a target of the given animation or 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.
For more information, see [SMILANIM-TARGET].

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 target element.
For more information, see [SMILANIM-TARGET].

attributeType = "**CSS | XML | auto**"

Specifies the namespace in which the target attribute and its associated values are defined. The attribute value is one of the following:

"XML"

This specifies that the value of "attributeName" is the name of an XML attribute on the target element. The attribute must be defined as animatable in this specification.

"CSS"

This specifies that the value of "attributeName" is the name of a CSS property defined as animatable in this specification.

"auto"

This specifies that the user agent will automatically check to determine if there is an animatable SVG property whose name matches the value of attributeName; if not, the user agent will check to determine if there is an animatable XML attribute on the target element whose name matches the value of attributeName.

The default value is "auto".
For more information, see [SMILANIM-TARGET].

Attributes defined elsewhere:

%xlinkRefAttrs;.

19.2.4 Attributes to control the timing of the animation

The following attributes are common to all animation elements and control the timing of the animation, including what causes the animation to start and end, whether the animation runs repeatedly, and whether to retain the end state the animation once the animation ends.

```
<!ENTITY % animTimingAttrs
 "begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  endActive CDATA #IMPLIED
  restart (always | never | whenNotActive) 'always'
  repeatCount CDATA #IMPLIED
  repeatDur CDATA #IMPLIED
  fill (remove | freeze) 'remove'" >
```

Attribute definitions:

begin

Defines when the element begins (i.e. become active).
The attribute value can be one of the following types of values:

clock-value

Specifies the presentation time at which the animation begins. The begin is thus defined relative to the document begin.

syncbase-value : (id-ref ".") ("begin" | "end") ("+"clock-value)?

Describes a syncbase and an offset from that syncbase. The element begin is defined relative to the begin **B** or active end **AE** of another animation.

event-value : (id-ref ".")? (event-ref) ("+"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. (See Event handling for a discussion of the various event attributes that can be used as Event-symbols.) Details of event-based timing are described in [SMILANIM-UNIFY].

"indefinite"

The begin of the animation will be determined by a "beginElement()" method call or a hyperlink targeted to the animation element.
The SMIL Animation DOM methods are described in DOM interfaces.
Hyperlink-based timing is described in [SMILANIM-LINKS].

For more information, see [SMILANIM-ATTR-BEGIN].

dur

Specifies the simple duration.
The attribute value can be one of the following types of values:

clock-value

Specifies the length of the simple duration in presentation time.

"indefinite"

Specifies the simple duration **d** as indefinite.

For more information, see [SMILANIM-ATTR-DUR].

end

Specifies the simple duration as the difference between an end time and the begin time of the animation **B**.
The attribute value can be one of the following types of values:

clock-value

Specifies the presentation time at which the simple duration ends. The end of the simple duration is thus defined relative to the document begin.

syncbase-value : (id-ref ".") ("begin" | "end") ("+"clock-value)?

Describes a syncbase and an offset from that syncbase. The animation end is defined relative to the begin **B** or active end **AE** of another animation.

For more information, see [SMILANIM-ATTR-END].

endActive

Defines the active end **AE** of the animation (i.e. the end of the active duration).
The attribute value can be one of the following types of values:

clock-value

Specifies the presentation time of the active end. The active end is thus defined relative to the document begin.

syncbase-value : (id-ref) (".begin" | ".end")? ("+"clock-value)?

Describes a syncbase and an offset from that syncbase. The active end is defined relative to the begin **B** or active end **AE** of another animation.

event-value : (id-ref ".")? (event-ref) ("+"clock-value)?

Describes an event and an optional offset that determine the active end. The active end is defined relative to the time that the event is raised. The event must be raised after the animation begins, and before the active duration otherwise ends (e.g. as defined by `repeatDur`). 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. (See Event handling for a discussion of the various event attributes that can be used as Event-symbols.) Details of event-based timing are described in [SMILANIM-UNIFY].

"indefinite"

The active end of the animation will be determined by an "endElement()" method call. The SMIL Animation DOM methods are described in the Supported Methods section.

For more information, see [SMILANIM-ATTR-ENDACTIVE].

restart

always

The animation can be restarted at any time.
This is the default value.

never

The animation cannot be restarted for the remainder of the document duration.

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.

For more information, see [SMILANIM-ATTR-RESTART].

repeatCount

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 **d**. Values must be greater than 0.

"indefinite"

The animation is defined to repeat indefinitely (i.e. until the document ends).

For more information, see [SMILANIM-ATTR-REPEATCOUNT].

repeatDur

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).

For more information, see [SMILANIM-ATTR-REPEATDUR].

fill

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 Restarting Animations).

remove

The animation effect is removed (no longer applied) when the active duration of the animation is over. After the active end **AE** of the animation, the animation no longer affects the target (unless the animation is restarted - see Restarting Animations).

This is the default value.

For more information, see [SMILANIM-ATTR-FILL].

If both `repeatCount` or `repeatDur` are specified (and the simple duration is not indefinite), the active duration is defined as the minimum of the specified repeatDur, and the simple duration multiplied by repeatCount. For the purposes of this comparison, a defined value is considered to be "less than" a value of "indefinite". If the simple duration is indefinite, and both `repeatCount` or `repeatDur` are specified, the `repeatCount` will be ignored, and the `repeatDur` will be used (refer to the examples below describing `repeatDur` and an indefinite simple duration). These rules are included in [SMILANIM-D].

Timing Attribute Values

In the syntax specifications that follow, allowed white space is indicated as "S", defined as follows (taken from the [XML] definition for "S"):

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

Clock values

Clock values have the following syntax:

```
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
```

Syncbase values

A syncbase value has the following syntax:

```
Syncbase-value   ::= ( Syncbase-element "." Time-symbol )
                     ( S "+" S Clock-value )?
Syncbase-element ::= Id-value
Time-symbol      ::= "begin" | "end"
```

A syncbase value starts with a Syncbase-element term defining the value of an "id" attribute of an animation element referred to as the *syncbase element*. This element must be another animation element contained in the host document.

The syncbase element is qualified with one of the following *time symbols:*

begin

Specifies the begin time of the syncbase element.

end

Specifies the Active End **AE** of the syncbase element.

The time symbol can be followed by a clock value. The clock value specifies a presentation time offset from the time (i.e. the begin or end) specified by the syncbase and time symbol. If the clock value is omitted, it defaults to "0".

No embedded white space is allowed between a syncbase element and a time-symbol. White space will be ignored before and after a "+" for a clock value. Leading and trailing white space characters (i.e. before and after the entire syncbase value) will be ignored.

Examples:

```
begin="x.end+45s"   : Begin 45 seconds after "x" ends
begin=" x.begin "   : Begin when "x" begins
end="x.begin + 1m"  : End 1 minute after "x" begins
```

Event values

An event value has the following syntax:

```
Event-value        ::= ( Eventbase-element "." )? Event-symbol
                       ( S "+" S Clock-value )?
Eventbase-element  ::= Id-value
```

An Event value starts with an Eventbase-element term that specifies the *event-base element*. The event-base element is the element on which the event is observed. Given DOM event bubbling, the event-base element may be either the element that raised the event, or it may be an ancestor element on which the bubbled event can be

observed. Refer to DOM-Level2-Events [DOM2-EVENTS] for details.
The "Id-value" is the value of an attribute declared to be an "id" in the host language, for the event-base element. This element must be another animation element contained in the host document.
If the Eventbase-element term is missing, the event-base element defaults to the target element of the animation.

The event value must specify an Event-symbol. This term specifies the name of the event that is raised on the Event-base element. . 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. (See Event handling for a discussion of the various event attributes that can be used as Event-symbols.)

The last term specifies an optional clock-value that is a presentation time offset from the event. If this term is omitted, the offset is 0.

No embedded white space is allowed between an eventbase element and an event-symbol. White space will be ignored before and after a "+" for a clock value. Leading and trailing white space characters (i.e. before and after the entire eventbase value) will be ignored.

Note that it is not considered an error to specify an event that cannot be raised on the Event-base element.

Examples:

```
begin=" x.onload "    : Begin when "onload" is observed on "x"
begin="x.onfocus+3s" : Begin 3 seconds after an "onfocus" event on "x"
```

The defaults for the event and target element syntax make it easy to define simple interactive behavior. The following example sets the `rect` element color to be red for 5 seconds, when the user clicks on the element.

```
<rect ...>
  <set begin="onclick" dur="5s" to="red"
       attributeName="fill" attributeType="CSS" />
  ...
</rect>
```

19.2.5 Attributes that define animation values over time

The following attributes are common to elements 'animate', 'animateMotion', 'animateColor' and 'animateTransform'. These attributes define the values that are assigned to the target attribute or property over time. The attributes below provide control over the relative timing of keyframes and the interpolation method between discrete values.

```
<!ENTITY % animValueAttrs
 "calcMode (discrete | linear | evenPace | spline) 'linear'
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED" >
```

Attribute definitions:

The animation is described either as a list of *values*, or in a simplified form that describes the *from*, *to* and *by*

values.

from = "**<value>**"

Specifies the starting value of the animation.

to = "**<value>**"

Specifies the ending value of the animation.

by = "**<value>**"

Specifies a relative offset value for the animation.

values = "**<list>**"

A semicolon-separated list of one or more values. Vector-valued attributes are supported using the vector syntax of the `attributeType` domain.

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 semi-colon 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 animation will have no effect.

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 simpler syntax provides for several variants. Note that `from` is optional, but that one of `by` or `to` must be used (unless of course a list of `values` is provided). It is not legal to specify both `by` and `to` attributes - if both are specified, only the `to` attribute will be used (the `by` will be ignored). The combinations of attributes yield the following classes of animation:

from-to animation

Specifying a `from` value and a `to` value defines a simple animation, equivalent to a `values` list with 2 values. The animation function is defined to start with the `from` value, and to finish with the `to` value.

from-by animation

Specifying a `from` value and a `by` value defines a simple animation in which the animation function is defined to start with the `from` value, and to change this over the course of the simple duration **`d`** by a *delta* specified with the `by` attribute. This can only be used with attributes that support addition (e.g. most numeric attributes).

by animation

Specifying only a by value defines a simple animation in which the animation function is defined to offset the underlying value for the attribute, using a delta that varies over the course of the simple duration **`d`**, starting from a delta of 0 and ending with the delta specified with the `by` attribute. This can only be used with attributes that support addition.

to animation

This describes an animation in which the animation function is defined to start with the underlying value for the attribute, and finish with the value specified with the `to` attribute. Using this form, an author can describe an animation that will start with whatever value the attribute has originally, and will end up at the desired `to` value.

For more information on these attributes, see [SMILANIM_VALUES].

The last two forms "*by animation*" and "*to animation*" have additional semantic constraints when combined with other animations. The details of this are described in [SMILANIM_FROMTOBY-ADD].

Examples

The following example using the `values` syntax animates the width of a 'rect' over the course of 10 seconds

from a width of 40 to a width of 100 and back to 40.

```
<rect ...>
   <animate attributeName="width" values="40;100;40" dur="10s"/>
</rect>
```

The following "*from-to animation*" example animates the width of a 'rect' over the course of 10 seconds from a width of 50 to a width of 100.

```
<rect ...>
   <animate attributeName="width" from="50" to="100" dur="10s"/>
</rect>
```

The following "*from-by animation*" example animates the width of a 'rect' over the course of 10 seconds from a width of 50 to a width of 75.

```
<rect ...>
   <animate attributeName="width" from="50" by="25" dur="10s"/>
</rect>
```

The following "*by animation*" example animates the width of a 'rect' over the course of 10 seconds from the original width of 40 to a width of 70.

```
<rect width="40"...>
   <animate attributeName="width" by="30" dur="10s"/>
</rect>
```

The following "*to animation*" example animates the width of a 'rect' over the course of 10 seconds from the original width of 40 to a width of 100.

```
<rect width="40"...>
   <animate attributeName="width" to="100" dur="10s"/>
</rect>
```

By default, a simple linear interpolation is performed over the values, evenly spaced over the duration of the animation. Additional attributes can be used for finer control over the interpolation and timing of the values. The `calcMode` attribute defines the basic method of applying values to the attribute. The `keyTimes` attribute provides additional control over the timing of the animation function, associating a time with each value in the `values` list. Finally, the `keySplines` attribute provides a means of controlling the pacing of interpolation *between* the values in the `values` list.

`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), this attribute is ignored and discrete interpolation is always 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. This is the default `calcMode`.

`"paced"`

Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). If "`paced`" is specified, any `keyTimes` or `keySplines` will be ignored.

`"spline"`

Interpolates from one value in the `values` list to the next according to a time function defined by a cubic Bezier 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.

`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.
Each successive time value must be greater than or equal to the preceding time value.
The first time value in the list must be 0, and the last time value in the list must be 1.
If a list of `keyTimes` is specified, there must be exactly as many values in the `keyTimes` list as in the `values` list.
If there are any errors in the `keyTimes` specification (bad values, too many or too few values), the animation will have no effect.
If the simple duration is indefinite, any `keyTimes` specification will be ignored.

`keySplines` = "**<list>**"

A set of Bezier control points associated with the `keyTimes` list, defining a cubic Bezier function that controls interval pacing. The attribute value is a semi-colon separated list of control point descriptions. Each control point description is a set of four values: `x1 y1 x2 y2`, describing the Bezier control points for one time segment. The `keyTimes` values that define the associated segment are the Bezier "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 animation will have no effect.

If the `keyTimes` attribute is not specified, the values in the `values` attribute are assumed to be equally spaced through the animation duration, according to the calcMode:

- For discrete animation, the duration is divided into equal time periods, one per value. The animation function takes on the values in order, one value for each time period.
- For linear and spline animation, the duration is divided into `n-1` even periods, and the animation function is a linear interpolation between the values at the associated times. Note that a linear animation will be a nicely closed loop if the first value is repeated as the last.

Note that for the shorthand forms *to animation* and *from-to animation*, there are only 1 and 2 values respectively. Thus a discrete *to animation* will simply set the "to" value for the simple duration. A discrete *from-to animation* will set the "from" value for the first half of the simple duration and the "to" value for the second half of the simple duration.

Note that if the `calcMode` is set to "paced", the `keyTimes` attribute is ignored, and the values in the `values` attribute are spaced to produce a constant rate of change as the target attribute value is interpolated.

If the argument values for `keyTimes` or `interSpline` are not legal (including too few or too many values for either attribute), the animation will have no effect.

In the `calcMode`, `keyTimes` and `keySplines` attribute values, leading and trailing white space and white space before and after semi-colon separators will be ignored.

Examples

This example describes a somewhat unusual usage: "*from-to animation*" with discrete animation. The 'text' element supports the font-style property, which takes a string, and so implies a `calcMode` of discrete. The animation will set the font-style to "normal" for 5 seconds (half the simple duration) and then set the font-style to "italic" for 5 seconds.

```
<text style="font-style:normal"...>
  <animate attributeName="font-style"
```

```
      from="normal" to="italic" dur="10s"/>
</text>
```

This example illustrates the use of `keyTimes`:

```
<animate attributeName="x" dur="10s" values="0; 50; 100"
     keyTimes="0; .8; 1" calcMode="linear"/>
```

The `keyTimes` values causes the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds (at 80% into the simple duration) and "100" at the end of the animation. The value will change more slowly in the first half of the animation, and more quickly in the second half.

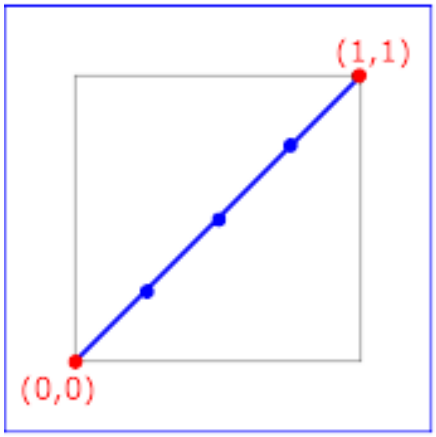
Extending this example to use `keySplines`:

```
<animate attributeName="x" dur="10s" values="0; 50; 100"
     keyTimes="0; .8; 1" calcMode="spline"
     keySplines=".5 0 .5 1; 0 0 1 1" />
```

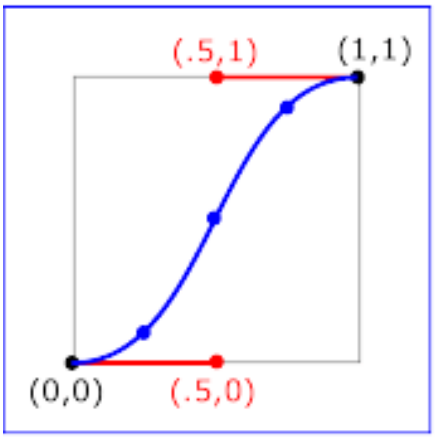
The `keyTimes` still causes the "x" attribute to have a value of "0" at the start of the animation, "50" after 8 seconds and "100" at the end of the animation. However, the `keySplines` values define a curve for pacing the interpolation between values. In the example above, the spline causes an ease-in and ease-out effect between time 0 and 8 seconds (i.e. between `keyTimes` 0 and .8, and `values` "0" and "50"), but a strict linear interpolation between 8 seconds and the end (i.e. between `keyTimes` .8 and 1, and `values` "50" and "100"). See the figure below for an illustration of the curves that these `keySplines` values define.

For some attributes, the *pace* of change might not be easily discernable by viewers. However for animations like motion, the ability to make the *speed* of the motion change gradually, and not in abrupt steps can be important. The `keySplines` attribute provides this control.

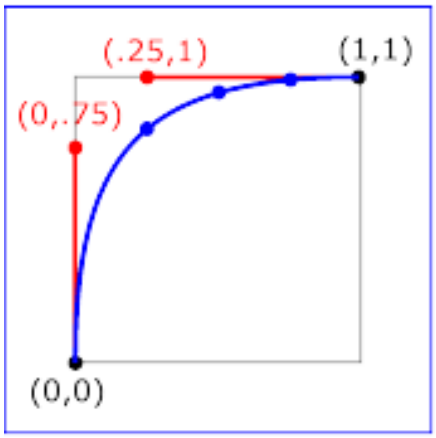
The following figure illustrates the interpretation of the `keySplines` attribute. Each diagram illustrates the effect of `keySplines` settings for a single interval (i.e. between the associated pairs of values in the `keyTimes` and `values` lists.). The horizontal axis can be thought of as the input value for the *unit progress* of interpolation within the interval - i.e. the pace with which interpolation proceeds along the given interval. The vertical axis is the resulting value for the *unit progress*, yielded by the `keySplines` function.



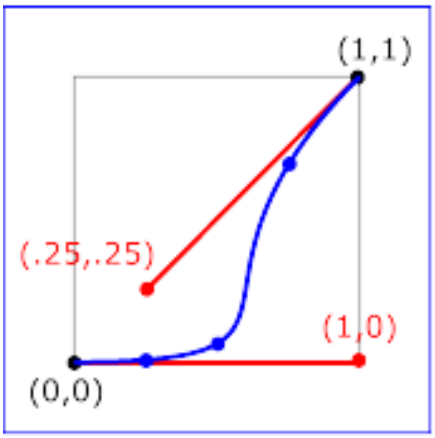
keySplines="0 0 1 1" (the default)



keySplines=".5 0 .5 1"



keySplines="0 .75 .25 1"



keySplines="1 0 .25 .25"

Examples of keySplines

To illustrate the calculations, consider the simple example:

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
     calcMode="spline" keySplines={as in table} />
```

Using the keySplines values for each of the four cases above, the approximate interpolated values as the animation proceeds are:

keySplines values	Initial value	After 1s	After 2s	After 3s	Final value
0 0 1 1	10.0	12.5	15.0	17.5	20.0
.5 0 .5 1	10.0	11.0	15.0	19.0	20.0
0 .75 .25 1	10.0	18.0	19.3	19.8	20.0
1 0 .25 .25	10.0	10.1	10.6	16.9	20.0

For a formal definition of Bezier spline calculation, see [FOLEY-VANDAM].

19.2.6 Combining animations

At a particular moment in time, an attribute can be animated by several animations, i.e. animations can overlap in time. The effect of this depends on whether the animations combined are additive or non-additive. An additive animation function will take the initial value of the attribute as defined by the animations that are already running, and use it as its begin value. A non-additive animation will replace the initial value with a new begin value

When there are multiple animations defined for a given attribute with complete durations that overlap at any moment, the two either add together or one overrides the other. The active animations are prioritized according to their begin. The animation first activated (i.e. begun by scheduled timing or by an event) has lowest priority and the most recently begun animation has highest priority. Higher priority animations that are not additive will override all earlier animations, and simply set the attribute value. Animations that are additive apply (i.e. add to) to the result of the earlier-activated animations. When two animations have the same begin, the first in lexical order has lower priority.

For more information, see [SMILAnim-ADD] and [SMILAnim-ACCUM].

19.2.7 Attributes that control whether animations are additive

The following attributes are common to elements 'animate', 'animateMotion', 'animateColor' and 'animateTransform'.

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>
```

The width begins at 20 pixels, and increases to 30 pixels over the course of 10 seconds. If the animation were declared to be not additive, the same from and to values would make the width go from 0 to 10 pixels over 10 seconds.

When there are multiple animations defined for a given attribute that are active at a given moment, the two either add together or one overrules the other. The active animations are prioritized according to the *activation* time of each. The animation first activated (i.e. begun by scheduled timing or by an event) has lowest priority and the most recently begun animation has highest priority. Higher priority animations that are not additive will overrule all earlier animations, and set the attribute value. Animations that are additive apply (i.e. add to) to the result of the earlier-activated animations. When two animations start at the same point in time, the first in lexical order is applied first.

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 80 pixels.

```
<!ENTITY % animAdditionAttrs
 "additive        (true | false) 'false'
  accumulate      (true | false) 'false'" >
```

Attribute definitions:

additive = "**replace | sum**"

Controls whether or not the animation is additive.
If `"sum"`, the animation will add to the underlying value of the attribute and other lower priority animations.
If `"replace"`, 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 [SMILANIM_FROMTOBY-ADD].

accumulate = "**none | sum**"

Controls whether or not the animation is cumulative.
If `"sum"`, each repeat iteration after the first builds upon the last value of the previous iteration.
If `"none"`, 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.

19.2.8 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.

19.2.9 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="text/css" attributeName="opacity"
         from="1" to="0" dur="5s" repeatCount="indefinite" />
</rect>
```

```
<!ENTITY % animateExt "" >
<!ELEMENT animate (%descTitle;%animateExt;) >
<!ATTLIST animate
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >
```

Attributes defined elsewhere:

id, system-required, system-language, %animTargetAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.

For a list of attributes and properties that can be animated using the 'animate' element, see Elements, attributes and properties that can be animated.

19.2.10 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 effect of repeatCount and repeatDur attributes are just to extend the defined duration. In addition, using fill="freeze" will have the same effect as an indefinite duration.

```
<!ENTITY % setExt "" >
<!ELEMENT set (%descTitle;%setExt;) >
<!ATTLIST set
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  to CDATA #IMPLIED >
```

Attribute definitions:

to = "<value>"

Specifies the value for the attribute during the duration of the 'set' element. The <value> must be appropriate to the target attribute or property.

Attributes defined elsewhere:

id, system-required, system-language, %animTargetAttrs;, %animTimingAttrs;.

For a list of attributes and properties that can be animated using the 'set' element, see Elements, attributes and properties that can be animated.

19.2.11 The 'animateMotion' element

The 'animateMotion' element causes a referenced element to move along a motion path.

```
<!ENTITY % animateMotionExt "" >
<!ELEMENT animateMotion (%descTitle;%animateMotionExt;) >
<!ATTLIST animateMotion
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  path CDATA #IMPLIED
  rotate CDATA #IMPLIED
  origin CDATA #IMPLIED >
```

For 'animateMotion', the specified values for from, by, to and values consists of x, y coordinate pairs, with a comma 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, where commas separate the x coordinate from the y coordinate and semicolons separate the various coordinate pairs, such as 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.

For more flexibility in controlling the motion path, the path attribute provides the ability to specify a motion path using any of SVG's path data commands. If a path is specified, it will override the motion path provided by

the values or from/to/by attributes. (Note that a path can only contain values in user space, whereas from, by, to and values can specify coordinates in user space or using CSS unit identifiers. For more information on CSS units, see Processing rules for CSS units and percentages.)

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 axis 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 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 "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) does not count as an additional point for the purpose of keyTimes and keySplines, and does not define an additional "segment" for the purposes of timing or interpolation. 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 path attribute overrides values, which overrides from/by/to. Regarding determining the points which correspond to the keyTimes attributes, the keyPoints attribute overrides path, which overrides values, which overrides from/by/to.

Attribute definitions:

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 axis 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), the animation will have no effect.

rotate = "<angle> | auto | auto-reflect"

> 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-reflect indicates that the object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees. An actual angle value can also be given, which represents an angle relative to X-axis of current user coordinate system. The rotate attribute adds a supplemental transformation matrix onto the CTM to apply a rotation transformation

about the origin of the current user coordinate system. The rotation transformation is applied after the supplemental translation transformation that is computed due to the path attribute. The default value is 0.

origin = "default"

The origin attribute is defined in the SMIL Animation specification [SMILANIM-ATTR-ORIGIN]. It has no effect in SVG.

Attributes defined elsewhere:

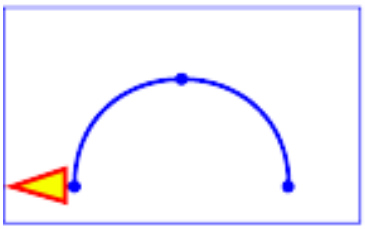
id, system-required, system-language, %animTargetAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.

At any time *t* within a motion path animation of effective duration *dur*, the computed coordinate (x,y) along the motion path is determined by finding the point (x,y) which is *t/dur* distance along the motion path using the user agent's distance along the path algorithm.

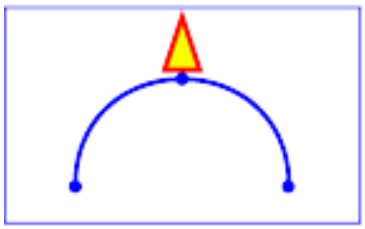
The following example demonstrates the supplemental transformation matrices that are computed during a motion path animation.

Example animMotion01 shows a triangle moving along a motion path.

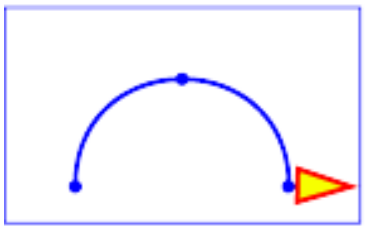
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="5cm" height="3cm"  viewBox="0 0 500 300">
  <desc>Example animMotion01 - demonstrate motion animation computations</desc>

  <!-- Draw the outline of the motion path in blue, along
         with three small circles at the start, middle and end. -->
  <path d="M100,250 C 100,50 400,50 400,250"
        style="fill:none; stroke:blue; stroke-width:7.06" />
  <circle cx="100" cy="250" r="17.64" style="fill:blue" />
  <circle cx="250" cy="100" r="17.64" style="fill:blue" />
  <circle cx="400" cy="250" r="17.64" style="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"
        style="fill:yellow; stroke:red; stroke-width:7.06" >

    <!-- Define the motion path animation -->
    <animateMotion dur="6s" repeatCount="indefinite"
                   path="M100,250 C 100,50 400,50 400,250" rotate="auto" />
  </path>
</svg>
```



At zero seconds



At three seconds



At six seconds

Example animMotion01

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

The following table shows the supplemental transformation matrices that are applied to achieve the effect of the motion path animation.

	After 0s	After 3s	After 6s
Supplemental transform due to movement along motion path	translate(100,250)	translate(250,100)	translate(400,250)
Supplemental transform due to rotate="auto"	rotate(-90)	rotate(0)	rotate(90)

For a list of elements that can be animated using the 'animateMotion' element, see Elements, attributes and properties that can be animated.

19.2.12 The 'animateColor' element

The 'animateColor' element specifies a color transformation over time.

```
<!ENTITY % animateColorExt "" >
<!ELEMENT animateColor (%descTitle;%animateColorExt;) >
<!ATTLIST animateColor
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >
```

Attributes defined elsewhere:

id, system-required, system-language, %animTargetAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.

The from, by and to attributes take color values, where each color value is expressed using the same syntax that is available for the target attribute or property.

The values attribute for the 'animateColor' element consists of a semicolon-separated list of color values, where each individual color value is expressed using the same syntax that is available for the target attribute or property.

Out of range color values can be provided, but user agent processing will be implementation dependent. User agents should clamp color values to allow color range values as late as possible, but note that system differences might preclude consistent behavior across different systems.

The 'color-interpolation' property applies to color interpolations that result from 'animateColor' animations.

For a list of attributes and properties that can be animated using the 'animateColor' element, see Elements, attributes and properties that can be animated.

19.2.13 The 'animateTransform' element

The 'animateTransform' element adds a supplemental transformation onto a target element so that it can be translated, scaled, rotated or skewed.

```
<!ENTITY % animateTransformExt "" >
<!ELEMENT animateTransform (%descTitle;%animateTransformExt;) >
<!ATTLIST animateTransform
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  type (translate | scale | rotate | skewX | skewY) "translate" >
```

Attribute definitions:

type = "translate | scale | rotate | skewX | skewY"

Indicates the type of transformation which is to have its values change over time.

Attributes defined elsewhere:

id, system-required, system-language, %animTargetAttrs;, %animTimingAttrs;, %animValueAttrs;, %animAdditionAttrs;.

The from, by and to attributes take a value expressed using the same syntax that is available for the given transformation type:

- For a type="translate", each individual value is expressed as <tx> [,<ty>].
- For a type="scale", each individual value is expressed as <sx> [,<sy>].
- For a type="rotate", each individual value is expressed as <rotate-angle>.
- For a type="skewX" and type="skewY", each individual value is expressed as <skew-angle>.

(See Modifying the User Coordinate System: the transform attribute.)

The values attribute for the 'animateTransform' element consists of a semicolon-separated list of values, where each individual value is expressed as described above for from, by and to.

If calcMode has the value paced, then a total "distance" for each component of the transformation is calculated (e.g., for a translate operation, a total distance is calculated for both *tx* and *ty*) consisting of the sum of the absolute values of the differences between each pair of values, and the animation runs to produce a constant distance movement for each individual component.

The effect of additive 'animateTransform' elements which are animating the same attribute or property is equivalent to nesting the corresponding transformation matrices (i.e., the notion of "additive" corresponds to transformation matrix multiplication).

For a list of attributes and properties that can be animated using the 'animateTransform' element, see Elements, attributes and properties that can be animated.

19.2.14 Elements, attributes and properties that can be animated

The following lists all of the elements which can be animated by an 'animateMotion' element:

- 'svg' ('animateMotion' has no effect on outermost 'svg' elements)
- 'g'
- 'path'
- 'rect'
- 'circle'
- 'ellipse'
- 'line'
- 'polyline'
- 'polygon'
- 'text'
- 'use'
- 'image'
- 'clipPath'
- 'mask'
- 'switch'

Each attribute or property within this specification indicates whether or not it can be animated by SVG's animation elements. Animatable attributes and properties are designated as follows:

Animatable: yes.

whereas attributes and properties that cannot be animated are designated:

Animatable: no.

SVG has a defined set of basic data types for its various supported attributes and properties. For those attributes and properties that can be animated, the following table indicates which animation elements can be used to animate each of the basic data types.

Basic data type	Additive?	'animate'	'set'	'animateColor'	'animateTransform'	Notes
<angle>	yes	yes	yes	no	no	
<color>	yes	yes	yes	yes	no	Only RGB color values are additive.
<coordinate>	yes	yes	yes	no	no	
<frequency>	no	no	no	no	no	
<integer>	yes	yes	yes	no	no	
<length>	yes	yes	yes	no	no	
<list of xxx>	no	yes	yes	no	no	

<number>	yes	yes	yes	no	no	
<paint>	yes	yes	yes	yes	no	Only RGB color values are additive.
<percentage>	yes	yes	yes	no	no	
<time>	no	no	no	no	no	
<transform-list>	yes	no	no	no	yes	
<uri>	yes	yes	yes	no	no	
All other animatable attributes and properties	no	yes	yes	no	no	

19.3 Animation using the SVG DOM

The following example shows a simple animation:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG December 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19991203.dtd">
<svg width="4in" height="3in"
     viewBox="0 0 400 300"
     onload="StartAnimation()" >

  <defs>
    <script><![CDATA[
      var timer_increment = 50.
      var max_time = 10000
      var text_element
      StartAnimation() {
        text_element = document.getElementById("TextElement");
        ShowAndGrowElement(0);
      }
      ShowAndGrowElement(timevalue) {
        timevalue = timevalue + timer_increment
        if (timevalue > max_time)
          timevalue = timevalue - floor(timevalue/max_time) * max_time

        // 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.getStyle().setProperty("opacity", "opacity:" + opacityfactor, "")

        // Call ShowAndGrowElement again <timer_increment> milliseconds later.
        setTimeout("ShowAndGrowElement(" + timer_increment + ")")
      }
    ]]></script>
  </defs>

  <g transform="translate(50,300)" style="fill:red; font-size:10">
    <text id="TextElement">SVG</text>
  </g>
</svg>
```

Download this example

The above SVG file contains a single graphics element, a text string that says "SVG". The animation loops continuously. 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 'svg' element's `width` and `height` attributes indicate that the viewport is a rectangle of size

4inches by 3inches. The `viewBox` attribute indicates that the initial coordinate system has (0,0) at its top left and (400,300) at its bottom right. (Thus, 1 inch equals 100 user units.) The `onload="StartAnimation()"` attribute indicates that when the document has been fully loaded and processed, then 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

19.4.1 Interface SVGAnimationElement

The SVGAnimationElement interface is the base interface for all of the animation element interfaces: SVGAnimateElement, SVGSetElement, SVGAnimateMotionElement and SVGTextpathElement.

The SVGAnimationElement interface implements the TimeControl interface defined in the SMIL Animation specification [SMILANIM-DOM-METHODS]. A DOM application can use the `hasFeature` method of the **DOMImplementation** [DOM2-CORE] interface to determine whether the `ElementTimeControl` interface is supported or not. The feature string for this interface is "ElementTimeControl".

Calling `beginElement()` causes the animation to begin in the same way that an animation with event-based begin timing begins. The effective begin time is the current presentation time at the time of the DOM method call. Note that `beginElement()` is subject to the `restart` attribute in the same manner that event-based begin timing is. If an animation is specified to disallow restarting at a given point, `beginElement()` methods calls must fail.

Calling `beginElementAtTimeOffset(seconds)` has the same behavior as `beginElement()`, except that the animation starts midway into the animation (i.e., a given number of `seconds` offset from the normal start of the animation.) If the offset value goes beyond the simple duration or the simple duration is undefined, (e.g., the end time is indefinite), then the animation does not start.

Similarly, `beginElementAtFractionOffset(fraction)` causes the animation to start at `fraction` from the start, where `fraction` is a number between 0 and 1 which represents a fraction of the simple duration. If the fraction is outside the range of 0 to 1 or if the simple duration is undefined (e.g., the end time is indefinite), then the animation does not start.

Calling `endElement()` causes an animation to end the active duration, just as `endActive` does. Depending upon the value of the `fill` attribute, the animation effect may no longer be applied, or it may be frozen at the current effect. If an animation is not currently active (i.e. if it has not yet begun or if it is frozen), the `endElement()` method will fail.

Unlike other SVG DOM interfaces, the SVG DOM does not specify convenience DOM attributes corresponding to the various language attributes on SVG's animation elements. Specification of these convenience properties in a way that will be compatible with future versions of SMIL Animation [SMILAnim] is expected in a future version of SVG. The current method for accessing and modifying the attributes on the animation elements is to use the standard `getAttribute`, `setAttribute`, `getAttributeNS` and `setAttributeNS` defined in [DOM2-CORE].

```
interface SVGAnimationElement  {
  boolean            beginElement()
                                 raises(DOMException);
  boolean            beginElementAtTimeOffset(in float seconds)
                                 raises(DOMException);
  boolean            beginElementAtFractionOffset(in float fraction)
                                 raises(DOMException);
  boolean            endElement()
                                 raises(DOMException);
};
```

Methods

`beginElement`

Causes this element to begin the local timeline (subject to sync constraints).

No Parameters

Return Value

`boolean` `true` if the method call was successful and the element was begun. `false` if the method call failed. Possible reasons for failure include:

- The element is already active and can't be restart when it is active. (the `restart` attribute is set to `"whenNotActive"`)
- The element is active or has been active and can't be restart. (the `restart` attribute is set to `"never"`).

Exceptions

`DOMException` SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `beginElement` calls.

`beginElementAtTimeOffset`

Causes this element to begin the local timeline (subject to sync constraints), but the animation starts midway as defined by parameter `seconds`.

Parameters:

`seconds` Real number value indicating the number of seconds from the start of the animation at which the animation should start.

Return Value: (same as beginElement)

Exceptions: (same as beginElement)

`beginElementAtFractionOffset`

Causes this element to begin the local timeline (subject to sync constraints), but the animation starts midway as defined by parameter `fraction`.

Parameters:

`fraction` Real number value indicating an offset from the start of the animation expressed as a fraction of the simple duration at which the animation should start.

Return Value: (same as beginElement)

Exceptions: (same as beginElement)

`endElement`

Causes this element to end the local timeline (subject to sync constraints).

No Parameters

Return Value

`boolean` `true` if the method call was successful and the element was endeed. `false` if method call failed. Possible reasons for failure include:

- The element is not active.

Exceptions

`DOMException` SYNTAX_ERR: The element was not defined with the appropriate syntax to allow `endElement` calls.

19.4.2 Interface SVGAnimateElement

The SVGAnimateElement interface corresponds to the 'animate' element.

```
interface SVGAnimateElement : SVGAnimationElement {
};
```

19.4.3 Interface SVGSetElement

The SVGSetElement interface corresponds to the 'set' element.

```
interface SVGSetElement : SVGAnimationElement {
};
```

19.4.4 Interface SVGAnimateMotionElement

The SVGAnimateMotionElement interface corresponds to the 'animateMotion' element.

```
interface SVGAnimateMotionElement : SVGAnimationElement {
};
```

19.4.5 Interface SVGAnimateTransformElement

The SVGAnimateTransformElement interface corresponds to the 'animateTransform' element.

```
interface SVGAnimateTransformElement : SVGAnimationElement {
};
```

19.4.5 Interface SVGAnimationEvent

```
interface SVGAnimationEvent : Event {
    // Transition Types
  const unsigned short kSVG_ANIMTRANSITION_UNKNOWN = 0; // unknown. Not supported
  const unsigned short kSVG_ANIMTRANSITION_START   = 1;
  const unsigned short kSVG_ANIMTRANSITION_FREEZE  = 2;
  const unsigned short kSVG_ANIMTRANSITION_RESTART = 3;
  const unsigned short kSVG_ANIMTRANSITION_STOP    = 4;
  readonly attribute  unsigned short transitionType;

  readonly attribute SVGElement timeContainer; // The 'svg' element.
  readonly attribute SVGAnimationElement animationElement;
  readonly attribute SVGElement targetElement;

  float        getStartTime()
                              raises(DOMException);
  float        getCurrentTime()
                              raises(DOMException);
  float        getSimpleDuration()
                              raises(DOMException);
};
```

Attributes

`transitionType`

Indicates the type of animation transition that caused the event to be generated. For a discussion of these transition types, refer to [SMILANIM-TRANSITION].

`timeContainer`

The 'svg' element which is the time container for the animation which generated this event.

`animationElement`

The animation element which generated this event.

`targetElement`

The element which is the target of the animation (i.e., the element whose attributes and/or properties are being modified over time).

Methods

`getStartTime`

Returns the start time in seconds for the given `animationElement`

No Parameters

Return Value

`float` The start time in seconds for the given `animationElement` relative to the start time of the `timeContainer`

No Exceptions

`getCurrentTime`

Returns the current time in seconds relative to time zero for the given `timeContainer`

No Parameters

Return Value

`float` The current time in seconds relative to time zero for the given `timeContainer`

No Exceptions

`getSimpleDuration`

Returns the number of seconds for the simple duration for the current `animationElement`. If the simple duration is undefined (e.g., the end time is indefinite), then an exception is raised.

No Parameters

Return Value

`float` The number of seconds for the simple duration for the current `animationElement`.

Exceptions

`DOMException` NO_SIMPLE_DURATION_ERR: The animation element's simple duration is undefined.

---

20 Fonts

Contents

20.1 Introduction

Reliable delivery of fonts is considered a critical requirement for SVG. Designers require the ability to create SVG graphics with whatever fonts they care to use and then have the same fonts appear in the end user's browser when viewing an SVG drawing, even if the given end user hasn't purchased the fonts in question. This parallels the print world, where the designer uses a given font when authoring a drawing for print, but when the end user views the same drawing within a magazine the text appears with the correct font.

SVG utilizes the web font facility defined in the "Cascading Style Sheets (CSS) level 2" specification [CSS2] as a key mechanism for reliable delivery of font data to end users. A common scenario is that SVG authoring applications will generate compressed, subsetted web fonts for all text elements used by a given SVG document fragment. Typically, the web fonts will be saved in a location relative to the

referencing document.

One disadvantage to CSS2's Webfont facility to date is that CSS2 did not specify particular font formats that were required to be supported. The result was that different implementations supported different web font formats, thereby making it difficult for web site creators to post a single web site that is supported by a large percentage of installed browsers.

To provide a common font format that will exist in all conforming SVG user agents, SVG includes elements which allow for fonts to be defined in SVG.

20.2 SVG fonts

20.2.1 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 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 attribute 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 web font 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 web font 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 "Cascading Style Sheets (CSS) level 2" specification [CSS2].

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 SVG document fragment that uses the font or saved as an external file and referenced via a URI reference.

Here is an example of how you might embed an SVG font inside of an SVG document:

```
<?xml version="1.0" standalone="yes"?>
<svg width="400px" height="300px"
  xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <defs>
    <font id="MyFont" font-face-name="Super Sans"
              units-per-em="1000" cap-height="600" x-height="400"
                  ascent="700" descent="300" horiz-adv-x="1000"
                  text-bottom="-300" baseline="0" centerline="350"
                  mathline="350" ideographic="400" hanging="500"
                  topline="700" text-top="700">
      <missing-glyph><path d="M0,0h200v200h-200z"/></glyph>
      <glyph unicode="33"><path d="M0,0L200,200L400,0z"/></glyph>
      <glyph unicode="34"><path d="M0,0L200,200L400,0z"/></glyph>
      <!-- more glyphs -->
    </font>
    <style>
      <![CDATA[
        @font-face {
          font-family: "MyFont";
          src: url("#MyFont") format(svg)
        }
      ]]>
    </style>
  </defs>
  <text style="font-family: MyFont, Helvetica, sans-serif">Text
    using embedded font</text>
</svg>
```

Download this example

Here is an example of how you might reference an SVG font which is saved in an external file. First referenced SVG font file:

```
<?xml version="1.0" standalone="yes"?>
<svg width="100%" height="100%"
 xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <defs>
    <font id="MyFont" font-face-name="Super Sans"
              units-per-em="1000" cap-height="600" x-height="400"
                  ascent="700" descent="300" horiz-adv-x="1000"
                  text-bottom="-300" baseline="0" centerline="350"
                  mathline="350" ideographic="400" hanging="500"
                  topline="700" text-top="700">
      <missing-glyph><path d="M0,0h200v200h-200z"/></glyph>
      <glyph unicode="33"><path d="M0,0L200,200L400,0z"/></glyph>
      <glyph unicode="34"><path d="M0,0L200,200L400,0z"/></glyph>
      <!-- more glyphs -->
    </font>
  </defs>
</svg>
```

Download this example

The SVG file which uses/references the above SVG font

```
<?xml version="1.0" standalone="yes"?>
<svg width="400px" height="300px"
 xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>  <defs>
    <style>
      <![CDATA[
        @font-face {
          font-family: "MyFont";
```

```
          src: url("myfont.svg#MyFont") format(svg)
        }
      ]]>
   </style>
  </defs>
  <text style="font-family: MyFont">Text using embedded font</text>
</svg>
```

Download this example

20.2.2 The 'font' element

The 'font' element defines an SVG font.

```
<!ENTITY % fontExt "" >
<!ELEMENT font (%descTitle;,missing-glyph,(glyph|hkern|vkern
                    %fontExt;)*) >
<!ATTLIST font
  id ID #IMPLIED
  font-style CDATA #IMPLIED
  font-variant CDATA #IMPLIED
  font-weight CDATA #IMPLIED
  font-stretch CDATA #IMPLIED
  unicode-range CDATA #IMPLIED
  units-per-em CDATA #REQUIRED
  panose-1 CDATA #IMPLIED
  slope CDATA #IMPLIED
  cap-height CDATA #REQUIRED
  x-height CDATA #REQUIRED
  accent-height CDATA #IMPLIED
  ascent CDATA #REQUIRED
  descent CDATA #REQUIRED
  horiz-origin-x CDATA #IMPLIED
  horiz-origin-y CDATA #IMPLIED
  horiz-adv-x CDATA #REQUIRED
  vert-origin-x CDATA #IMPLIED
  vert-origin-y CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED
  text-bottom CDATA #REQUIRED
  baseline CDATA #REQUIRED
  centerline CDATA #REQUIRED
  mathline CDATA #REQUIRED
  ideographic CDATA #REQUIRED
  hanging CDATA #REQUIRED
  topline CDATA #REQUIRED
  text-top CDATA #REQUIRED
  font-face-name CDATA #IMPLIED
  underline-position CDATA #IMPLIED
  underline-thickness CDATA #IMPLIED
  strikethrough-position CDATA #IMPLIED
  strikethrough-thickness CDATA #IMPLIED
  overline-position CDATA #IMPLIED
  overline-thickness CDATA #IMPLIED >
```

Attribute definitions:

font-style = "all | [normal | italic | oblique] [, [normal | italic | oblique]]*"

The style of a font. Takes on the same values as the 'font-style' property, except that a comma-separated list is permitted. The default value is all.
Animatable: no.

font-variant = "[normal | small-caps] [,[normal | small-caps]]*"

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. The default value is normal.
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]]*"

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:

1. relative keywords (bolder, lighter) are not permitted
2. a comma-separated list of values is permitted, for fonts that contain multiple weights
3. 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.

The default value is all.
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]]*"

Indication of the condensed or expanded nature of the face relative to others in the same font family. Takes on the same values as the 'font-stretch' property except that:

- relative keywords (wider,narrower) are not permitted
- a comma-separated list is permitted
- the keyword 'all' is permitted

The default value is normal.
Animatable: no.

unicode-range = "<urange> [, <urage>]*"

The range of ISO 10646 characters [UNICODE] covered by the font. For more information, see the description of the 'unicode-range' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2]. The default value is U+0-7FFFFFFF.
Animatable: no.

units-per-em = "<number>"

The number of coordinate units on the em square, the size of the design grid on which glyphs are laid out. For more information, see the description of the 'units-per-em' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

panose-1 = "[<integer>]{10}"

The Panose-1 number, consisting of ten decimal integers, separated by whitespace. For more

information, see the description of the 'panose-1' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2]. The default value is 0 0 0 0 0 0 0 0 0 0.
Animatable: no.

slope = "<number>"

The vertical stroke angle of the font. For more information, see the description of the 'slope' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2]. The default value is 0.
Animatable: no.

cap-height = "<number>"

The height of uppercase glyphs in the font within the font coordinate system. For more information, see the description of the 'cap-height' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

x-height = "<number>"

The height of lowercase glyphs in the font within the font coordinate system. For more information, see the description of the 'x-height' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

accent-height = "<number>"

The distance from the baseline to the top of accent characters, measure by a distance within the font coordinate system. The default value is the value of the ascent attribute.
Animatable: no.

ascent = "<number>"

The maximum unaccented height of the font within the font coordinate system. For more information, see the description of the 'ascent' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

descent = "<number>"

The maximum unaccented depth of the font within the font coordinate system. For more information, see the description of the 'descent' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

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. The default value is 0.
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. The default value is 0.
Animatable: no.

horiz-adv-x = "<number>"

The default horizontal advance after rendering a glyph in horizontal orientation. Glyph widths are required to be positive, 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 a glyph to be used when drawing vertically oriented text. The default value is half of the value of attribute horiz-adv-x.
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. The default value is 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. The default value is the sum of the values of attributes ascent and. descent.
Animatable: no.

text-bottom = "<number>"

The bottom of the font within the font coordinate system. For more information, see the description of the 'text-bottom' value for the 'vertical-align' property in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

baseline = "<number>"

The lower baseline of a font within the font coordinate system. For more information, see the description of the 'baseline' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

centerline = "<number>"

The central baseline of a font within the font coordinate system. For more information, see the description of the 'centerline' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

mathline = "<number>"

The mathematical baseline of a font within the font coordinate system. For more information, see the description of the 'mathline' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

ideographic = "<number>"

The ideographic baseline of a font within the font coordinate system.
Animatable: no.

hanging = "<number>"

The hanging baseline of a font within the font coordinate system.
Animatable: no.

topline = "<number>"

The top baseline of a font within the font coordinate system. For more information, see the

description of the 'topline' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

text-top = "<number>"

The top of the font within the font coordinate system. For more information, see the description of the 'text-top' value for the 'vertical-align' property in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
Animatable: no.

font-face-name = "<string>"

The full name of a particular face of a font family. It typically includes a variety of non-standardized textual qualifiers or *adornments* appended to the font family name. For more information, see the description of full font names in the "Cascading Style Sheets (CSS) level 2" specification [CSS2].
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.

Attributes defined elsewhere:

id.

20.2.3 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 contents of a 'glyph' can be any SVG graphics elements. However, in some implementations, faster

font rendering (and possibly improved quality) might occur when glyph definitions consist of a single 'path' element.

```
<!ENTITY % glyphExt "" >
<!ELEMENT glyph (%descTitleDefs;,
                 (path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|g|switch
                  %glyphExt;)*) >
<!ATTLIST glyph
  id ID #IMPLIED
  unicode CDATA #REQUIRED
  glyph-name CDATA #IMPLIED
  vert-text-orient CDATA #IMPLIED
  arabic CDATA #IMPLIED
  han CDATA #IMPLIED
  horiz-adv-x CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED >
```

Attribute definitions:

unicode = "<string>"

One or more Unicode characters indicating the sequence of Unicode characters which corresponds to this glyph. If a character is provided, then this glyph corresponds to the given Unicode character. If a list of characters is provided, then this glyph corresponds to the given sequence of Unicode characters. One use of a list of numbers is for ligatures. For example, if unicode="ffl", then the given glyph will be used to render the sequence of characters "f", "f", and "l". (This could alternatively have been expressed using character entities, using XML character references expressed in hexadecimal notation: unicode="ffl", or XML character references expressed in decimal notation: unicode="ffl".) 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 lexical order to see if the upcoming sequence of Unicode characters to be rendered match the sequence of Unicode characters specified in the unicode attribute for the given 'glyph' element. The first successful match is used.

Note that any occurrences of 'altglyph' take precedence over the glyph selection rules within an SVG font.
Animatable: no.

glyph-name = "<name> [, <name>]* "

A name for the glyph. It is recommended that glyph names be unique across 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.

vert-text-orient = "default | h | v"

When drawing vertical text, indicates whether the given glyph is meant to be drawn with a vertical or horizontal orientation. The default value is vertOrient="default", which indicates that the Unicode character number determines the orientation of this glyph.
Animatable: no.

arabic = "initial | medial | terminal | isolated"

For Arabic glyphs, indicates which of the four possible forms this glyph represents.

Animatable: no.

han = "ja | zht | zhs | kor"

For glyphs in the Han range, indicates which of the four possible forms this glyph represents.
Animatable: no.

horiz-adv-x = "<number>"

The horizontal advance after rendering a glyph in horizontal orientation. The default value is the value of the font's horizAdvX attribute. Glyph widths are required to be positive, even if the glyph is typically rendered right-to-left, as in Hebrew and Arabic scripts.
Animatable: no.

vert-adv-y = "<number>"

The vertical advance after rendering a glyph in vertical orientation. The default value is the value of the font's vertAdvY attribute.
Animatable: no.

Attributes defined elsewhere:

id.

20.2.4 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 been defined. The attributes on the 'missing-glyph' element have the same meaning as the corresponding attributes on the 'glyph' element.

```
<!ENTITY % missing-glyphExt "" >
<!ELEMENT missing-glyph (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|g|switch
                   %missing-glyphExt;)*) >
<!ATTLIST missing-glyph
  id ID #IMPLIED
  horiz-adv-x CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED >
```

Attributes defined elsewhere:

id. horiz-adv-x, vert-adv-y.

20.2.5 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 bi-directional text in SVG is laid out in a two-step process, which is described in Relationship with bi-directionality. If SVG fonts are used, before kerning is aplied, characters are re-ordered into left-to-right (or top-to-bottom, for vertical text) visual rendering order. Kerning from SVG fonts is then applied on pairs of glyphs which are rendered contiguously. The first glyph in the kerning pair is the left (or top) glyph in visual rendering order. The second glyph in the kerning pair is the right (or bottom) glyph in the pair.

For convenience to font designers and to minimize file sizes, a single 'hkern' and 'vkern' can define a single kerning adjustment value between one set of glyphs (e.g., a range of Unicode characters) and another set of glyphs (e.g., another range of Unicode characters).

The 'hkern' element defines kerning pairs and adjustment values in the horizontal advance value when drawing pairs of glyphs which the two glyphs are contiguous and are both rendered horizontally (i.e., side-by-side). The spacing between characters is reduced by the kerning adjustment. (Negative kerning adjustments increase the spacing between characters.)

```
<!ELEMENT hkern EMPTY >
<!ATTLIST hkern
  id ID #IMPLIED
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >
```

Attribute definitions:

u1 = "[<character> | <urange>] [, [<character> | <urange>]]* "

A sequence (comma-separated) of Unicode characters (refer to the description of the unicode attribute to the 'glyph' element for a description of how to express individual Unicode characters) and/or unicode ranges (see description of unicode ranges in [CSS2]) which identify a set of possible first glyphs in the kerning pair. If a given Unicode character within the set has multiple corresponding 'glyph' elements (i.e., there are multiple 'glyph' elements with the same unicode attribute value, but different glyphName values), then all such glyphs are included in the set. Comma is the separator character; thus, to kern a comma, specify the comma as part of a Unicode range or as a glyph name using the g1 attribute. The total set of possible first glyphs in the kerning pair is the union of glyphs specified by the u1 and g1 attributes.
Animatable: no.

g1 = "<name> [, <name>]* "

A sequence (comma-separated) of glyph names (i.e., values that match glyphName attributes on 'glyph' elements) which identify a set of possible first glyphs in the kerning pair. All glyphs with the given glyph name are included in the set. The total set of possible first glyphs in the kerning pair is the union of glyphs specified by the u1 and g1 attributes.
Animatable: no.

u2 = "[<number> | <urange>] [, [<number> | <urange>]]* "

Same as the u1 attribute, except that u2 specifies possible second glyphs in the kerning pair.

Animatable: no.

g2 = "<name> [, <name>]* "

Same as the g1 attribute, except that g2 specifies possible second glyphs in the kerning pair.
Animatable: no.

k = "<number>"

The amount to decrease the spacing between the two glyphs in the kerning pair. The value is in the font coordinate system.
Animatable: no.

Attributes defined elsewhere:

id.

At least one each of u1 or g1 and at least one of u2 or g2 must be provided.

The 'vkern' element defines kerning pairs and adjustment values in the vertical advance value when drawing pairs of glyphs together when stacked vertically. The spacing between characters is reduced by the kerning adjustment.

```
<!ELEMENT vkern EMPTY >
<!ATTLIST vkern
  id ID #IMPLIED
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >
```

Attributes defined elsewhere:

id. u1. g1. u2. g2. k.

20.3 DOM interfaces

20.3.1 Interface SVGFontElement

The SVGFontElement interface corresponds to the 'font' element.

```
interface SVGFontElement : SVGElement {
};
```

20.3.2 Interface SVGGlyphBaseElement

The SVGGlyphBaseElement interface is the base interface for interfaces SVGGlyphElement and SVGMissingGlyphElement.

```
interface SVGBaseGlyphElement : SVGElement {
};
```

20.3.3 Interface SVGGlyphElement

The SVGGlyphElement interface corresponds to the 'glyph' element.

```
interface SVGGlyphElement : SVGBaseGlyphElement {
};
```

20.3.4 Interface SVGMissingGlyphElement

The SVGMissingGlyphElement interface corresponds to the 'missingGlyph' element.

```
interface SVGMissingGlyphElement : SVGBaseGlyphElement {
};
```

20.3.5 Interface SVGKernBaseElement

The SVGKernBaseElement interface is the base interface for interfaces SVGHKernElement and SVGVKernElement.

```
interface SVGBaseKernElement : SVGElement {
};
```

20.3.6 Interface SVGHKernElement

The SVGHKernElement interface corresponds to the 'hkern' element.

```
interface SVGHKernElement : SVGBaseKernElement {
};
```

20.3.7 Interface SVGVKernElement

The SVGVKernElement interface corresponds to the 'vkern' element.

```
interface SVGVKernElement : SVGBaseKernElement {
};
```

---

21 Metadata

Contents

21.1 Introduction

Metadata is information about a document.

RDF is the appropriate language for metadata. The specifications for RDF can be found at:

- Resource Description Framework Model and Syntax Specification
- Resource Description Framework (RDF) Schema Specification

It is recommended that metadata within an SVG document fragment be expressed in an appropriate RDF namespaces and placed within the **'metadata'** child element to the document's **'svg'** root element. (See Example below.)

Here are some suggestions for content creators regarding metadata:

- It is recommended that content creators refer to W3C Metadata Recommendations and activities when deciding which metadata schema to use in their documents.
- It is recommended that content creators refer to the Dublin Core, which is a set of generally applicable core metadata properties (e.g., Title, Creator/Author, Subject, Description, etc.).
- Additionally, SVG Metadata Schema (below) contains a set of additional metadata properties that are common across most uses of vector graphics.

Individual industries or individual content creators are free to define their own metadata schema, but everyone is 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 RDF which is used in combination with the given standard metadata schema than to totally avoid the standard schema.

21.2 The SVG Metadata Schema

(This schema has not yet been defined. Here are some candidate attributes for the schema: MeetsAccessibilityGuidelines, UsesDynamicElements, ListOfExtensionsUsed, ListOfICCProfilesUsed, LiistOfFontsUsed, ListOfImagesUsed, ListOfForeignObjectsUsed, ListOfExternalReferences.)

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 and the SVG metadata schema:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
    xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
    <desc xmlns:myfoo="http://bar.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 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <rdf:RDF
               xmlns:rdf = "http://www.w3.org/...-rdf-syntax-ns"
               xmlns:rdfs = "http://www.w3.org/TR/...-schema"
               xmlns:dc = "http://purl.org/dc/elements/1.1/"
               xmlns:svgmetadata = "http://www.w3.org/..." >
        <rdf:Description about="http://bar.org/myfoo"
                  dc:title="MyFoo Financial Report"
                  dc:description="$three $bar $thousands $dollars $from 1998
$through 2000"
                  dc:publisher="BarOrg Incorporated"
                  dc:date="1999-03-03"
                  dc:format="image/svg"
                  dc:language="en" >
          <dc:creator>
            <rdf:Bag>
                <rdf:li>Irving Bird</rdf:li>
                <rdf:li>Mary Lambert</rdf:li>
            </rdf:Bag>
          </dc:creator>
          <svgmetadata:General UsesControlledVocabulary="true"/>
        </rdf:Description>
      </rdf:RDF>
    </metadata>
</svg>
```

Download this example

---

22 Backwards Compatibility

A user agent (UA) might not have the ability to process and view SVG content. The following list outlines two of the backwards compatibility scenarios associated with SVG content:

- For XML grammars with the ability to embed SVG content, it is assumed that some sort of alternate representation capability such as the 'switch' element and some sort of feature-availability test facility (such as what is described in the SMIL 1.0 specification [SMIL1]) will be available.

  This 'switch' element and feature-availability test facility (or their equivalents) are the recommended way for XML authors to provide an alternate representation to SVG content, such as an image or a text string. The following example shows how to embed an SVG drawing within a SMIL 1.0 document such that an alternate image will display in the event the UA doesn't support SVG. (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",
         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" src="drawing.svg" />

      <!-- Else, render the alternate image. -->
      <img src="alternate_image.jpg" />
    </switch>
  </body>
</smil>
```

  Download this example

- For HTML 4.0, SVG drawings can be embedded using the 'object' element. An alternate representation such as an image can be included as the content of the 'object' element. In this case, the SVG content usually will be included via a URL reference. The following example shows how to use the 'object' element to include an SVG drawing via a URL reference with an image serving as the alternate representation in the absence of an SVG user agent:

```
<html>
  <body>
    <object type="image/svg" 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="short 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"
     xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <defs>
    <myapp:piechart xmlns:myapp="http://mycompany/mapapp"
                    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 graphics elements which
       draw the pie chart -->
</svg>
```

Download this example

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 the HTML/CSS processor 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 [MATHML] expression into an SVG drawing.

The 'foreignObject' element allows for inclusion of foreign namespaces which has graphical content drawn by a different user agent, where the graphical content that is drawn 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 is recursively embedded within the different namespace (e.g., an SVG document fragment contains an XHTML document fragement which in turn contains yet another SVG document fragment).

Usually, a 'foreignObject' will be used in conjunction with the 'switch' element and the system-required system-language attributes to provide proper checking for user agent support and provide an alternate rendering in case user agent support isn't available.

Here is an example:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
 xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'>
  <desc>This example uses the switch element to provide a
  fallback graphical representation of an equation, 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 HTML if the system-required attribute
         evaluates to true (i.e., the user agent supports XHTML
         embedded within SVG). -->
    <foreignObject system-required="SVGForeignObject:XHTML" width="100" height="50">
      <!-- XHTML content goes here -->
    </foreignObject>

    <!-- Else, process the following alternate SVG.
         Note that there are no testing attributes on the <g> element.
         If no testing attributes are provided, it is as if there
         were testing attributes and they evaluated to true.-->
    <g>
      <!-- Draw a red rectangle with a text string on top. -->
      <rect width="20" height="20" style="fill: red"/>
```

```
      <text>Formula goes here</text>
    </g>

  </switch>
</svg>
```

Download this example

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.

Appendix A: Document Type Definition

This appendix is normative.

The DTD is also available for download.

```
<!--
    This is the DTD for Scalable Vector Graphics (SVG) 1.0 (draft 19991203).
    The specification for SVG that corresponds to this DTD is available at:

        http://www.w3.org/1999/12/WD-SVG-19991203/

-->


<!--================== Generic Attributes ==================-->


<!-- This entity allows for at most one of desc and title,
     supplied in any order -->
<!ENTITY % descTitle
          "((desc,title?)|(title,desc?)?)" >

<!-- This entity allows for at most one of desc, title and defs,
     supplied in any order -->
<!ENTITY % descTitleDefs
          "(((desc,((title,defs?)|(defs,title?))?)|
              (title,((desc,defs?)|(defs,desc?))?)|
              (defs,((desc,title?)|(title,desc?))?))?)" >

<!-- Supplemental attributes to xlink:href for all elements
     which reference to other elements using XLink -->
<!ENTITY % xlinkRefAttrs
 "xmlns:xlink CDATA #FIXED 'http://www.w3.org/XML/XLink/0.9'
  xlink:type (simple|extended|locator|arc) #FIXED 'simple'
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|embed|replace) #FIXED 'embed'
  xlink:actuate (user|auto) #FIXED 'auto'" >

<!ENTITY % graphicsElementEvents
  "onfocusin CDATA #IMPLIED
   onfocusout CDATA #IMPLIED
   ongainselection CDATA #IMPLIED
   onloseselection CDATA #IMPLIED
   onactivate CDATA #IMPLIED
   onmousedown CDATA #IMPLIED
   onmouseup CDATA #IMPLIED
   onclick CDATA #IMPLIED
   ondblclick CDATA #IMPLIED
   onmouseover CDATA #IMPLIED
   onmousemove CDATA #IMPLIED
   onmouseout CDATA #IMPLIED
   onkeydown CDATA #IMPLIED
   onkeypress CDATA #IMPLIED
   onkeyup CDATA #IMPLIED
   onload CDATA #IMPLIED
   onselect CDATA #IMPLIED" >

<!ENTITY % documentEvents
  "onresize CDATA #IMPLIED
   onscroll CDATA #IMPLIED
   onunload CDATA #IMPLIED
   onzoom CDATA #IMPLIED
   onerror CDATA #IMPLIED
   onabort CDATA #IMPLIED " >
```

```
<!ENTITY % structured_text
  "content CDATA #FIXED 'structured text'" >

<!-- Allow for extending the DTD with internal subset for
     container and graphics elements -->
<!ENTITY % ceExt "" >
<!ENTITY % geExt "" >


<!--========== Document Structure and Grouping ==========-->

<!ENTITY % svgExt "" >
<!ELEMENT svg (%descTitleDefs;,metadata?,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%svgExt;)*) >
<!ATTLIST svg
  xmlns CDATA #FIXED 'http://www.w3.org/Graphics/SVG/SVG-19991203.dtd'
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  %documentEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  enableZoomAndPanControls (true | false) "true"
  contentScriptType CDATA #IMPLIED >

<!ENTITY % gExt "" >
<!ELEMENT g (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|
                   animate|set|animateMotion|animateColor|animateTransform
                   %ceExt;%gExt;)*) >
<!ATTLIST g
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED >

<!ENTITY % defsExt "" >
<!ELEMENT defs (script|style|symbol|marker|clipPath|mask|
                linearGradient|radialGradient|pattern|filter|cursor|font|
                animate|set|animateMotion|animateColor|animateTransform|
                path|text|rect|circle|ellipse|line|polyline|polygon|
                use|image|svg|g|view|switch|altGlyphDef
                %ceExt;%defsExt;)* >
<!ATTLIST defs
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED >


<!--==================== Shapes ====================-->
```

```
<!ENTITY % pathExt "" >
<!ELEMENT path (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%pathExt;)*) >
<!ATTLIST path
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  d CDATA #REQUIRED
  nominalLength CDATA #IMPLIED >

<!ENTITY % rectExt "" >
<!ELEMENT rect (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%rectExt;)*) >
<!ATTLIST rect
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  rx CDATA #IMPLIED
  ry CDATA #IMPLIED >

<!ENTITY % circleExt "" >
<!ELEMENT circle (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%circleExt;)*) >
<!ATTLIST circle
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
  r CDATA #REQUIRED >

<!ENTITY % ellipseExt "" >
<!ELEMENT ellipse (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%ellipseExt;)*) >
<!ATTLIST ellipse
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
```

```
  rx CDATA #REQUIRED
  ry CDATA #REQUIRED >

<!ENTITY % lineExt "" >
<!ELEMENT line (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%lineExt;)*) >
<!ATTLIST line
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x1 CDATA "0"
  y1 CDATA "0"
  x2 CDATA "0"
  y2 CDATA "0" >

<!ENTITY % polylineExt "" >
<!ELEMENT polyline (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polylineExt;)*) >
<!ATTLIST polyline
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  points CDATA #REQUIRED >

<!ENTITY % polygonExt "" >
<!ELEMENT polygon (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                %geExt;%polygonExt;)*) >
<!ATTLIST polygon
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  points CDATA #REQUIRED >


<!--==================== Text ====================-->

<!ENTITY % textExt "" >
<!ELEMENT text (#PCDATA|tspan|tref|textPath|altglyph|use|animate|set|animateMotion|animateColor|animateTransform
                %geExt;%textExt;)* >
<!ATTLIST text
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED >
```

```
<!ENTITY % tspanExt "" >
<!ELEMENT tspan (#PCDATA|tspan|tref|altglyph|animate|set|animateColor
                %tspanExt;)* >
<!ATTLIST tspan
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED
  rotate CDATA #IMPLIED >

<!ENTITY % trefExt "" >
<!ELEMENT tref (animate|set|animateColor
                %trefExt;)* >
<!ATTLIST tref
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED
  rotate CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >

<!ENTITY % textPathExt "" >
<!ELEMENT textPath (#PCDATA|tspan|tref|altglyph|animate|set|animateColor
                %textPathExt;)* >
<!ATTLIST textPath
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  startOffset CDATA "0"
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >

<!ENTITY % altGlyphExt "" >
<!ELEMENT altGlyph (#PCDATA %altGlyphExt;)* >
<!ATTLIST altGlyph
  id ID #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >

<!ENTITY % altGlyphDefExt "" >
<!ELEMENT altGlyphDef (glyphSub %altGlyphDefExt;)* >
<!ATTLIST altglyphDef
  id ID #IMPLIED >

<!ELEMENT glyphSub EMPTY >
<!ATTLIST glyphSub
  id ID #IMPLIED
  font CDATA #REQUIRED
```

```
  glyphRef CDATA #REQUIRED
  format CDATA #REQUIRED >

<!--=================== SVG Fonts ===================-->

<!ENTITY % fontExt "" >
<!ELEMENT font (%descTitle;,missing-glyph,(glyph|hkern|vkern
                  %fontExt;)*) >
<!ATTLIST font
  id ID #IMPLIED
  font-style CDATA #IMPLIED
  font-variant CDATA #IMPLIED
  font-weight CDATA #IMPLIED
  font-stretch CDATA #IMPLIED
  unicode-range CDATA #IMPLIED
  units-per-em CDATA #REQUIRED
  panose-1 CDATA #IMPLIED
  slope CDATA #IMPLIED
  cap-height CDATA #REQUIRED
  x-height CDATA #REQUIRED
  accent-height CDATA #IMPLIED
  ascent CDATA #REQUIRED
  descent CDATA #REQUIRED
  horiz-origin-x CDATA #IMPLIED
  horiz-origin-y CDATA #IMPLIED
  horiz-adv-x CDATA #REQUIRED
  vert-origin-x CDATA #IMPLIED
  vert-origin-y CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED
  text-bottom CDATA #REQUIRED
  baseline CDATA #REQUIRED
  centerline CDATA #REQUIRED
  mathline CDATA #REQUIRED
  ideographic CDATA #REQUIRED
  hanging CDATA #REQUIRED
  topline CDATA #REQUIRED
  text-top CDATA #REQUIRED
  font-face-name CDATA #IMPLIED
  underline-position CDATA #IMPLIED
  underline-thickness CDATA #IMPLIED
  strikethrough-position CDATA #IMPLIED
  strikethrough-thickness CDATA #IMPLIED
  overline-position CDATA #IMPLIED
  overline-thickness CDATA #IMPLIED >

<!ENTITY % glyphExt "" >
<!ELEMENT glyph (%descTitleDefs;,
                 (path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|g|switch
                  %glyphExt;)*) >
<!ATTLIST glyph
  id ID #IMPLIED
  unicode CDATA #REQUIRED
  glyph-name CDATA #IMPLIED
  vert-text-orient CDATA #IMPLIED
  arabic CDATA #IMPLIED
  han CDATA #IMPLIED
  horiz-adv-x CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED >

<!ENTITY % missing-glyphExt "" >
<!ELEMENT missing-glyph (%descTitleDefs;,
                 (path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|g|switch
                  %missing-glyphExt;)*) >
<!ATTLIST missing-glyph
  id ID #IMPLIED
  horiz-adv-x CDATA #IMPLIED
  vert-adv-y CDATA #IMPLIED >
```

```
<!ELEMENT hkern EMPTY >
<!ATTLIST hkern
  id ID #IMPLIED
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >

<!ELEMENT vkern EMPTY >
<!ATTLIST vkern
  id ID #IMPLIED
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >


<!--==================== Graphics Referencing Elements ====================-->

<!ENTITY % useExt "" >
<!ELEMENT use (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                    %geExt;%useExt;)*) >
<!ATTLIST use
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >

<!ENTITY % imageExt "" >
<!ELEMENT image (%descTitle;,(animate|set|animateMotion|animateColor|animateTransform
                    %geExt;%imageExt;)*) >
<!ATTLIST image
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >


<!--==================== Symbols and Markers ====================-->

<!ENTITY % symbolExt "" >
<!ELEMENT symbol (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%symbolExt;)*) >
```

```
<!ATTLIST symbol
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet' >

<!ENTITY % markerExt "" >
<!ELEMENT marker (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%markerExt;)*) >
<!ATTLIST marker
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  markerUnits (stroke-width | userSpace | userSpaceOnUse) "stroke-width"
  markerWidth  CDATA "3"
  markerHeight CDATA "3"
  orient CDATA "0" >


<!--========== Descriptions and Titles ==========-->

<!ELEMENT desc (#PCDATA)* >
<!ATTLIST desc
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %structured_text; >

<!ELEMENT title (#PCDATA)* >
<!ATTLIST title
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %structured_text; >


<!--================== Clipping and Masking ==================-->

<!ENTITY % clipPathExt "" >
<!ELEMENT clipPath (%descTitle;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%clipPathExt;)*) >
<!ATTLIST clipPath
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  clipPathUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace" >

<!ENTITY % maskExt "" >
<!ELEMENT mask (%descTitleDefs;,
```

```
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|image|svg|g|switch|a|
                     animate|set|animateMotion|animateColor|animateTransform
                     %ceExt;%maskExt;)*) >
<!ATTLIST mask
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  maskUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED >


<!--================== Gradients and Patterns ==================-->

<!ENTITY % linearGradientExt "" >
<!ELEMENT linearGradient (stop|animate|set|animateTransform
                    %linearGradientExt;)* >
<!ATTLIST linearGradient
  id ID #IMPLIED
  gradientUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  x1 CDATA #IMPLIED
  y1 CDATA #IMPLIED
  x2 CDATA #IMPLIED
  y2 CDATA #IMPLIED
  spreadMethod (pad | reflect | repeat) "pad"
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >


<!ENTITY % radialGradientExt "" >
<!ELEMENT radialGradient (stop|animate|set|animateTransform
                    %radialGradientExt;)* >
<!ATTLIST radialGradient
  id ID #IMPLIED
  gradientUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  cx CDATA #IMPLIED
  cy CDATA #IMPLIED
  r CDATA #IMPLIED
  fx CDATA #IMPLIED
  fy CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >


<!ENTITY % stopExt "" >
<!ELEMENT stop (animate|set|animateColor
                    %stopExt;)* >
<!ATTLIST stop
  id ID #IMPLIED
  style CDATA #IMPLIED
  offset CDATA #REQUIRED >

<!ENTITY % patternExt "" >
<!ELEMENT pattern (%descTitleDefs;,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|image|svg|g|switch|a
                     %ceExt;%patternExt;)*) >
<!ATTLIST pattern
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
```

```
  patternUnits (userSpace | userSpaceOnUse | objectBoundingBox) 'userSpace'
  patternTransform CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  refX CDATA #IMPLIED
  refY CDATA #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >


<!--==================== Linking ====================-->

<!ENTITY % aExt "" >
<!ELEMENT a (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a
                   %ceExt;%aExt;)*) >
<!ATTLIST a
  id ID #IMPLIED
  xmlns:xlink CDATA #FIXED "http://www.w3.org/XML/XLink/0.9"
  xlink:type (simple|extended|locator|arc) #FIXED "simple"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:show (new|embed|replace) 'replace'
  xlink:actuate (user|auto) #FIXED 'user'
  xlink:href CDATA #REQUIRED
  target CDATA #IMPLIED >

<!ENTITY % viewExt "" >
<!ELEMENT view (%descTitle;%viewExt;) >
<!ATTLIST view
  id ID #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  enableZoomAndPanControls (true | false) "true"
  viewTarget CDATA #IMPLIED >


<!--==================== Animation ====================-->

<!ENTITY % animTargetAttrs
 "%xlinkRefAttrs;
  xlink:href CDATA #IMPLIED
  attributeName  CDATA  #REQUIRED
  attributeType  CDATA  #IMPLIED" >

<!ENTITY % animTimingAttrs
 "begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  endActive CDATA #IMPLIED
  restart (always | never | whenNotActive) 'always'
  repeatCount CDATA #IMPLIED
  repeatDur CDATA #IMPLIED
  fill (remove | freeze) 'remove'" >

<!ENTITY % animValueAttrs
 "calcMode (discrete | linear | evenPace | spline) 'linear'
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  keyTimes CDATA #IMPLIED
  keySplines CDATA #IMPLIED" >
```

```
<!ENTITY % animAdditionAttrs
 "additive        (true | false) 'false'
  accumulate      (true | false) 'false'" >

<!ENTITY % animateExt "" >
<!ELEMENT animate (%descTitle;%animateExt;) >
<!ATTLIST animate
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >

<!ENTITY % setExt "" >
<!ELEMENT set (%descTitle;%setExt;) >
<!ATTLIST set
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  to CDATA #IMPLIED >

<!ENTITY % animateMotionExt "" >
<!ELEMENT animateMotion (%descTitle;%animateMotionExt;) >
<!ATTLIST animateMotion
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  path CDATA #IMPLIED
  rotate CDATA #IMPLIED
  origin CDATA #IMPLIED >

<!ENTITY % animateColorExt "" >
<!ELEMENT animateColor (%descTitle;%animateColorExt;) >
<!ATTLIST animateColor
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs; >

<!ENTITY % animateTransformExt "" >
<!ELEMENT animateTransform (%descTitle;%animateTransformExt;) >
<!ATTLIST animateTransform
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  %animTargetAttrs;
  %animTimingAttrs;
  %animValueAttrs;
  %animAdditionAttrs;
  type (translate | scale | rotate | skewX | skewY) "translate" >


<!--========== Defining Scripts and Declaring Styles ==========-->

<!ELEMENT script (#PCDATA)* >
<!ATTLIST script
  language CDATA #IMPLIED
  %xlinkRefAttrs;
```

```
  xlink:href CDATA #IMPLIED >

<!ELEMENT style (#PCDATA)* >
<!ATTLIST style type CDATA "text/css" >



<!--================== Custom cursors ====================-->

<!ELEMENT cursor (%descTitle;) >
<!ATTLIST cursor
  id ID #IMPLIED
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA "0"
  y CDATA "0"
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED >


<!--================== Extensibility ====================-->

<!ENTITY % switchExt "" >
<!ELEMENT switch (%descTitleDefs;,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|foreignObject|
                   animate|set|animateMotion|animateColor|animateTransform
                   %ceExt;%switchExt;)*) >
<!ATTLIST switch
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED >

<!ENTITY % foreignObjectExt "" >
<!ELEMENT foreignObject (#PCDATA %ceExt;%foreignObjectExt;)* >
<!ATTLIST foreignObject
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  transform CDATA #IMPLIED
  %graphicsElementEvents;
  system-required NMTOKEN #IMPLIED
  system-language CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  %structured_text; >



<!--================== Metadata ====================-->

<!ENTITY % metadataExt "" >
<!ELEMENT metadata (#PCDATA %metadataExt;)* >
<!ATTLIST metadata
  id ID #IMPLIED >


<!--============== Filter Effects ==============-->

<!ENTITY % filter_node_attributes
  "result CDATA #IMPLIED
   x CDATA #IMPLIED
```

```
    y CDATA #IMPLIED
    width CDATA #IMPLIED
    height CDATA #IMPLIED">

<!ENTITY % filter_node_attributes_with_in
  "%filter_node_attributes;
    in CDATA #IMPLIED">

<!ENTITY % component_transfer_function_attributes
  "type CDATA #REQUIRED
    tableValues CDATA #IMPLIED
    slope CDATA #IMPLIED
    intercept CDATA #IMPLIED
    amplitude CDATA #IMPLIED
    exponent CDATA #IMPLIED
    offset CDATA #IMPLIED" >

<!ENTITY % filterExt "" >
<!ELEMENT filter (feBlend|feFlood|
  feColorMatrix|feComponentTransfer|
  feComposite|feDiffuseLighting|feDisplacementMap|
  feGaussianBlur|feImage|feMerge|
  feMorphology|feOffset|feSpecularLighting|
  feTile|feTurbulence|
  animate|set
  %filterExt;)* >
<!ATTLIST filter
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  filterUnits (userSpace | userSpaceOnUse | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED
  filterRes CDATA #IMPLIED
  %xlinkRefAttrs;
  xlink:href CDATA #IMPLIED >


<!ELEMENT feBlend EMPTY >
<!ATTLIST feBlend
  %filter_node_attributes_with_in;
  mode (normal | multiple | screen | darken | lighten) "normal"
  in2 CDATA #REQUIRED >

<!ELEMENT feFlood (animate|set|animateColor)* >
<!ATTLIST feFlood
  %filter_node_attributes_with_in;
  style CDATA #IMPLIED >

<!ELEMENT feColorMatrix (animate|set)* >
<!ATTLIST feColorMatrix
  %filter_node_attributes_with_in;
  type CDATA #REQUIRED
  values CDATA #IMPLIED >

<!ELEMENT feComponentTransfer (feFuncR?,feFuncG?,feFuncB?,feFuncA?) >
<!ATTLIST feComponentTransfer
  %filter_node_attributes_with_in; >

<!ELEMENT feFuncR (animate|set)* >
<!ATTLIST feFuncR
  %component_transfer_function_attributes; >

<!ELEMENT feFuncG (animate|set)* >
<!ATTLIST feFuncG
  %component_transfer_function_attributes; >

<!ELEMENT feFuncB (animate|set)* >
<!ATTLIST feFuncB
  %component_transfer_function_attributes; >

<!ELEMENT feFuncA (animate|set)* >
<!ATTLIST feFuncA
  %component_transfer_function_attributes; >
```

```
<!ELEMENT feComposite EMPTY >
<!ATTLIST feComposite
  %filter_node_attributes_with_in;
  operator (over | in | out | atop | xor | arithmetic) "over"
  k1 CDATA #IMPLIED
  k2 CDATA #IMPLIED
  k3 CDATA #IMPLIED
  k4 CDATA #IMPLIED
  in2 CDATA #REQUIRED >

<!ELEMENT feDiffuseLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feDiffuseLighting
  %filter_node_attributes_with_in;
  resultScale CDATA #IMPLIED
  surfaceScale CDATA #IMPLIED
  diffuseConstant CDATA #IMPLIED
  lightColor CDATA #IMPLIED >

<!ELEMENT feDistantLight (animate|set)* >
<!ATTLIST feDistantLight
  azimuth CDATA #IMPLIED
  elevation CDATA #IMPLIED >

<!ELEMENT fePointLight (animate|set)* >
<!ATTLIST fePointLight
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  z CDATA #IMPLIED >

<!ELEMENT feSpotLight (animate|set)* >
<!ATTLIST feSpotLight
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  z CDATA #IMPLIED
  pointsAtX CDATA #IMPLIED
  pointsAtY CDATA #IMPLIED
  pointsAtZ CDATA #IMPLIED
  specularExponent CDATA #IMPLIED >

<!ELEMENT feDisplacementMap (animate|set)* >
<!ATTLIST feDisplacementMap
  %filter_node_attributes_with_in;
  scale CDATA #IMPLIED
  xChannelSelector (R | G | B | A) "A"
  yChannelSelector (R | G | B | A) "A"
  in2 CDATA #REQUIRED >

<!ELEMENT feGaussianBlur (animate|set)* >
<!ATTLIST feGaussianBlur
  %filter_node_attributes_with_in;
  stdDeviation CDATA #IMPLIED >

<!ELEMENT feImage (animate|set|animateTransform)* >
<!ATTLIST feImage
  %filter_node_attributes;
  %xlinkRefAttrs;
  xlink:href CDATA #REQUIRED
  transform CDATA #IMPLIED >

<!ELEMENT feMerge (feMergeNode)* >
<!ATTLIST feMerge
  %filter_node_attributes_with_in; >

<!ELEMENT feMergeNode EMPTY >
<!ATTLIST feMergeNode
  in CDATA #IMPLIED >

<!ELEMENT feMorphology (animate|set)* >
<!ATTLIST feMorphology
  %filter_node_attributes_with_in;
  operator (erode | dilate) "erode"
  radius CDATA #IMPLIED >

<!ELEMENT feOffset (animate|set)* >
<!ATTLIST feOffset
  %filter_node_attributes_with_in;
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED >
```

```
<!ELEMENT feSpecularLighting ((feDistantLight|fePointLight|feSpotLight),(animate|set|animateColor)*) >
<!ATTLIST feSpecularLighting
  %filter_node_attributes_with_in;
  surfaceScale CDATA #IMPLIED
  specularConstant CDATA #IMPLIED
  specularExponent CDATA #IMPLIED
  lightColor CDATA #IMPLIED >

<!ELEMENT feTile EMPTY >
<!ATTLIST feTile
  %filter_node_attributes_with_in; >

<!ELEMENT feTurbulence (animate|set)* >
<!ATTLIST feTurbulence
  %filter_node_attributes_with_in;
  baseFrequencyX CDATA #IMPLIED
  baseFrequencyY CDATA #IMPLIED
  numOctaves CDATA #IMPLIED
  stitchTiles (stitch | noStitch) "noStitch"
  type (fractalNoise | turbulence) "turbulence" >
```

Appendix B: SVG's Document Object Model (DOM)

Contents

This appendix is normative.

B.1 SVG DOM Overview

This appendix provides an introduction to the SVG DOM and discusses the relationship of the SVG DOM with the Document Object Model (DOM) Level 2 Specification [DOM2]. The specific SVG DOM interfaces that correspond to particular sections of the SVG specification are defined at the end of corresponding chapter in this specification.

The SVG DOM is compatible with the Document Object Model (DOM) Level 2 Specification [DOM2]. In particular:

- The SVG DOM includes complete support for the DOM2 core [DOM2-CORE]
- Wherever appropriate, the SVG DOM is modeled after and maintains consistency with the DOM for HTML as described in [DOM2-HTML].
- The SVG DOM includes complete support for the DOM2 views [DOM2-VIEWS]
- The SVG DOM includes complete support for the DOM2 stylesheets [DOM2-SHEETS]
- The SVG DOM incorporates relevant aspects of the DOM2 CSS object model [DOM2-CSS]. (For the specific [DOM2-CSS] features that are supported, see Relationship with DOM2 CSS

object model.)

- The SVG DOM incorporates relevant aspects of the DOM2 event model [DOM2-EVENTS]. (For the specific [DOM2-EVENTS] features that are supported, see Relationship with DOM2 event model.)
- The optional traversal features described in [DOM2-TRAV] are also optional features within the SVG DOM.
- The range features described in [DOM2-RANGE] are supported by the SVG DOM.

A DOM application can use the hasFeature method of the DOMImplementation interface to verify that the interfaces listed in this section are supported. The list of available interfaces is provided in section Feature strings for the **hasFeature** method call.

B.2 Naming Conventions

The SVG DOM follows similar naming conventions to the Document Object Model HTML [DOM2-HTML].

All names are defined as one or more English words concatenated together to form a single string. Property or method names start with the initial keyword in lowercase, and each subsequent word starts with a capital letter. For example, a property that returns document meta information such as the date the file was created might be named "fileDateCreated". In the ECMAScript binding, properties are exposed as properties of a given object. In Java, properties are exposed with get and set methods.

For attributes with the CDATA data type, the case of the return value is that given in the source document.

B.3 Interface SVGException

Exception *SVGException*

This exception is raised when a specific SVG operation is impossible to perform.

IDL Definition

```
exception SVGException {
  unsigned short   code;
};

// SVGExceptionCode
const unsigned short SYNTAX_ERR                   = 0;
const unsigned short SVG_INVALID_MODIFICATION_ERR = 1;
const unsigned short SVG_NO_GRAPHICS_ELEMENTS     = 2;
const unsigned short SVG_MATRIX_NOT_INVERTABLE    = 3;
```

B.4 Interface SVGDOMImplementation

Interface *SVGDOMImplementation*

The `SVGDOMImplementation` interface extends the `DOMImplementation` interface with a method for creating an SVG document instance.

IDL Definition

```
interface SVGDOMImplementation : DOMImplementation {
  SVGDocument  createSVGDocument(in DOMString title);
};
```

Methods

`createSVGDocument`

Creates an `SVGDocument` object with no content. **No Parameters**

Return Value

`SVGDocument` A new `SVGDocument` object.

No Exceptions

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 [DOM2-CORE] are the same features strings available for the system-required attribute that is available for many SVG elements.

The **version** number for the **hasFeature** method call is "1".

B.6 Relationship with DOM2 CSS object model

B.6.1 Introduction

This section describes the relationship between the SVG DOM and the Document Object Model CSS [DOM2-CSS] described in the [DOM2] specification.

B.6.2 Aural media

For the purposes of aural media, SVG represents a CSS-stylable XML grammar. For user agents that support aural styling [CSS2-AURAL], all of the interfaces defined in [DOM2-CSS] which apply to aural properties must be supported in the DOM.

B.6.3 Visual media

For visual media [CSS2-VISUAL], the SVG DOM extends [DOM2-CSS].

The SVG DOM supports all of the required interfaces defined in [DOM2-CSS]. All of the interfaces that are optional for [DOM2-CSS] are also optional for the SVG DOM.

The SVG DOM defines the following SVG-specific custom property interfaces, all of which are mandatory for SVG user agents:

- SVGColor
- SVGICCColor
- SVGPaint

[DOM2-CSS] defines a set of extended interfaces [DOM2-CSS-EI]. The following table specifies the type of CSSValue [DOM2-CSSVALUE] used to represent each SVG property that applies to visual media [CSS2-VISUAL]. The table indicates which extended interfaces are mandatory and which are not.

The expectation is that the CSSValue returned from the getPropertyCSSValue method on the CSSStyleDeclaration interface can be cast down, using binding-specific casting methods, to the specific derived interface.

For properties that are represented by a custom interface (the valueType of the CSSValue is CSS_CUSTOM), the name of the derived interface is specified in the table. For properties that consist of lists of values (the valueType of the CSSValue is CSS_VALUE_LIST), the derived interface is CSSValueList. For all other properties (the valueType of the CSSValue is CSS_PRIMITIVE_VALUE), the derived interface is CSSPrimitiveValue.

Property Name	Representation	Mandatory?
'baseline-shift'	`null`	
'clip'	rect, ident	
'clip-path'	uri, ident	
'clip-rule'	ident	
'color'	rgbcolor, ident	
'color-interpolation'	ident	
'color-rendering'	ident	
'cursor'	[DOM2-CSS2Cursor]	no
'direction'	ident	
'display'	ident	
'enable-background'	ident	
'fill'	SVGPaint	yes
'fill-opacity'	number	
'fill-rule'	ident	

'filter'	uri, ident	
'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-anchor'	ident	
'glyph-orientation-horizontal'	ident	
'glyph-orientation-vertical'	ident	
'image-rendering'	ident	
'letter-spacing'	ident, length	
'marker'	`null`	
'marker-end'	uri, ident	
'marker-mid'	uri, ident	
'marker-start'	uri, ident	
'mask'	uri, ident	
'opacity'	number	
'overflow'	ident	
'pointer-events'	ident	
'shape-rendering'	ident	
'stop-color'	SVGColor	yes
'stop-opacity'	number	
'stroke'	SVGPaint	yes
'stroke-dasharray'	ident or list of lengths	
'stroke-dashoffset'	length	
'stroke-linecap'	ident	
'stroke-linejoin'	ident	
'stroke-miterlimit'	length	
'stroke-opacity'	number	
'stroke-width'	length	
'text-anchor'	ident	

'text-decoration'	list of ident	
'text-rendering'	ident	
'unicode-bidi'	ident	
'visibility'	ident	
'word-spacing'	length, ident	
'writing-mode'	ident	

B.7 Relationship with DOM2 events

The SVG DOM supports the following interfaces and event types from [DOM2-EVENTS]:

- The SVG DOM supports all of the interfaces defined in [DOM2-EVENTS].
- The SVG DOM supports the following UI event types [DOM2-UIEVENTS]:
  - resize
  - scroll (triggered by either scroll or pan user actions)
  - focusin
  - focusout
  - gainselection (user agents are required to support this event for text selection actions)
  - loseselection (user agents are required to support this event for text selection actions)
  - activate
- The SVG DOM supports the following mouse event types [DOM2-MOUSEEVENTS]:
  - click
  - mousedown
  - mouseup
  - mouseover
  - mousemove
  - mouseout

  clientX and *clientY* parameters for mouse events represent viewport coordinates for the corresponding 'svg' element. *relatedNode* is the corresponding 'svg' element.
- The SVG DOM supports the following keyboard event types [DOM2-KEYEVENTS]:
  - keypress
  - keydown
  - keyup
- The SVG DOM supports the following mutation event types [DOM2-MUTEVENTS]:
  - DOMSubtreeModified
  - DOMNodeInserted

    - DOMNodeRemoved
    - DOMNodeRemovedFromDocument
    - DOMNodeInsertedIntoDocument
    - DOMAttrModified
    - DOMCharacterDataModified
- The SVG DOM defines the following SVG-specific custom event interfaces, which are compatible with the HTML event types [DOM2-HTMLEVENTS] defined in [DOM2-EVENTS]. These event interfaces are mandatory for SVG user agents:
    - load
    - unload
    - abort
    - error

  Additionally, the SVG DOM defines an additional custom event interface that is not available in the HTML DOM:
    - SVGZoomEvent
- SVG includes a set of animation events which allow scripts to get invoked when there is a state change in an animation.

Each SVG element which has at least one event attribute assigned to it in the SVG DTD supports the DOM2 event registration interfaces [DOM2-EVREG] and be registered as an event listener for the corresponding DOM2 event using the event registration interfaces. Thus, for example, if the SVG DTD indicates that a given element supports the "onclick" event attribute, then an event listener for the "click" event can be registered with the given element as the event target.

SVG's animation elements also support the DOM2 event registration interfaces [DOM2-EVREG]. Event listeners for any of the animation events can be registered on any of the animation elements.

Event listeners which are established by DOM2 Event registration interfaces [DOM2-EVREG] receive events before any event listeners that correspond to event attributes (see Event attributes) or animations.

Appendix C: Implementation Requirements

Contents

This appendix is normative.

C.1 Introduction

The following are notes about implementation requirements corresponding to various features in the SVG language.

C.2 Version control

The SVG user agent must verify the reference to the SVG DTD in the `<!DOCTYPE>` statement or the `xmlns` attribute to ensure that it identifies the DTD for a version of the SVG language which the SVG user agent supports. If the reference to the DTD is missing or it does not correspond to a version which the SVG user agent supports, then the SVG user agent must not attempt to process the given SVG document fragment. If the user environment allows it, the user agent must generate an appropriate error message with a suggested alternative processing option (e.g., installing an updated version of the user agent) if such an option exists.

In particular, SVG user agents must not attempt to process SVG document fragments whose `<!DOCTYPE>` statement or corresponding `xmlns` attribute references a DTD corresponding to a working draft version of the SVG specification. These document fragments need to be updated to the SVG Recommendation before they can be rendered by SVG user agents which support the SVG Recommendation. If the user environment allows it, the user agent must generate an appropriate error message indicating that the SVG document fragment is not a conforming SVG Document fragment.

C.3 Forward and undefined references

SVG makes extensive use of URI references to other objects. For example, to fill a rectangle with a linear gradient, you define a **'linearGradient'** element and give it an ID (e.g., **<linearGradient id="MyGradient"...>**, and then you can specify the rectangle as follows: **<rect style="fill:url(#MyGradient)"...>**.

In SVG, among the facilities that allow URI references are:

- the 'clip-path' property
- the 'mask' property
- the 'fill' property
- the 'stroke' property
- the 'marker','marker-start','marker-mid' and 'marker-end properties
- the 'use' element

Forward references are disallowed. All references must be to elements which are either defined in a separate document or defined earlier in same document. References to elements in the same document can only be to elements which are direct children of a 'defs' element. (See Defining referenced and undrawn elements: the **'defs'** element.).

Unless a given attribute or property has defined fallback behavior when a reference cannot be resolved (e.g., typically, when a list of alternative values is provided), invalid references are treated as errors (see Error Processing). For example, if there is no element with ID "BogusReference" in the current document, then **fill="url(#BogusReference)"** would represent an invalid reference and would be an error.

C.4 Referenced objects are "pinned" to their own coordinate systems

Except in the cases where value userSpaceOnUse or objectBoundingBox is assigned to:

- attribute gradientUnits on element 'linearGradient' or 'radialGradient'
- attribute patternUnits on element 'pattern'
- attribute clipPathUnits on element 'clipPath'
- attribute maskUnits on element 'mask'
- attribute filterUnits on element 'filter'

when a graphical object is referenced by another graphical object, the referenced object does not change location, size or orientation. Thus, referenced graphical objects are "pinned" to the user coordinate system that is in place within its own hierarchy of ancestors and is not affected by the user coordinate system of the

referencing object.

C.5 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, but 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).

C.6 '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/developer about where the error might be in the path data specification. This rule will greatly discourage generation of invalid SVG path data.
  - If a path data command contains an incorrect set of parameters, then the given path data command is rendered up to and including the last correctly defined path segment, even if that path segment is a sub-component of a compound path data command, such as a "lineto" with several pairs of coordinates. For example, for the path data string "M 10,10 L 20,20,30", there is an odd number of parameters for the "L" command, which requires an even number of parameters. The user agent is required to draw the line from (10,10) to (20,20) and then perform error reporting since "L 20 20" is the last correctly defined segment of the pat data specification.
  - Wherever possible, all SVG user agents shall report all errors to the user.
- Markers, directionality and zero-length path segments:
  - If markers are specified, then a marker is drawn on every applicable vertex, even if the given vertex is the end point of a zero-length path segment and even if "moveto" commands follow each other.
  - Certain line-capping and line-joining situations and markers require that a path segment have directionality at its start and end points. Zero-length path segments have no directionality. In these cases, the following algorithm is used to establish directionality: to determine the directionality of the start point of a zero-length path segment, go backwards in the path data

specification within the current subpath until you find a segment which has directionality at its end point (e.g., a path segment with non-zero length) and use its ending direction; otherwise, temporarily consider the start point to lack directionality. Similarly, to determine the directionality of the end point of a zero-length path segment, go forwards in the path data specification within the current subpath until you find a segment which has directionality at its start point (e.g., a path segment with non-zero length) and use its starting direction; otherwise, temporarily consider the end point to lack directionality. If the start point has directionality but the end point doesn't, then the end point uses the start point's directionality. If the end point has directionality but the start point doesn't, then the start point uses the end point's directionality. Otherwise, set the directionality for the path segment's start and end points to align with the positive X-axis in user space.

    - If **'stroke-linecap'** is set to **butt** and the given path segment has zero length, do not draw the linecap for that segment; however, do draw the linecap for zero-length path segments when **'stroke-linecap'** is set to either **round** or **square**. (This allows round and square dots to be drawn on the canvas.)

- The S/s commands indicate that the first control point of the given cubic bezier 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.

C.7 Elliptical arc implementation notes

C.7.1 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 formulae to help implementers deal with endpoint parameterization.

C.7.2 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.

C.7.3 Parameterization alternatives

An arbitrary point (x, y) on the elliptical arc can be described by the 2-dimensional matrix equation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos\theta \\ r_Y \sin\theta \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \quad \text{(C.7.3.1)}$$

(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).

φ 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_1 and $\Delta\theta$

are the start angle, end angle and sweep angle, respectively of the arc prior to the stretch and rotate operations. This leads to an alternate parameterization which is common among graphics APIs, which will be referred to as *center parameterization*. In the next sections, formulas are given for mapping in both directions between center parameterization and endpoint parameterization.

C.7.4 Conversion from center to endpoint parameterization

Given:

$$c_X \;\; c_X \;\; r_X \;\; r_Y \;\; \varphi \;\; \theta_1 \;\; \Delta\theta$$

the task is to find:

$$x_1 \;\; y_1 \;\; x_2 \;\; y_2 \;\; f_A \;\; f_S$$

Here are the formulas:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos\theta_1 \\ r_Y \sin\theta_1 \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \qquad \text{(C.7.4.1)}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} r_X \cos(\theta_1 + \Delta\theta) \\ r_Y \sin(\theta_1 + \Delta\theta) \end{pmatrix} + \begin{pmatrix} c_X \\ c_Y \end{pmatrix} \qquad \text{(C.7.4.2)}$$

$$f_A = \begin{cases} 1 & \text{if } |\Delta\theta| > 180° \\ 0 & \text{if } |\Delta\theta| \le 180° \end{cases} \tag{C.7.4.3}$$

$$f_S = \begin{cases} 1 & \text{if } \Delta\theta > 0° \\ 0 & \text{if } \Delta\theta < 0° \end{cases} \tag{C.7.4.4}$$

C.7.5 Conversion from endpoint to center parameterization

Given:

$$x_1 \quad y_1 \quad x_2 \quad y_2 \quad f_A \quad f_S$$

the task is to find:

$$c_X \quad c_X \quad r_X \quad r_Y \quad \varphi \quad \theta_1 \quad \Delta\theta$$

The equations simplify after a translation which places the origin at the midpoint of the line joining (x_1, y_1) to (x_2, y_2), followed by a rotation to line up the coordinate axes with the axes of the ellipse. All transformed coordinates will be written with primes. They are computed as intermediate values on the way toward finding the required center parameterization variables. This procedure consists of the following steps:

Step 1: Compute (x_1', y_1') *according to the formula*

$$\begin{pmatrix} x_1' \\ y_1' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} \frac{x_1 - x_2}{2} \\ \frac{y_1 - y_2}{2} \end{pmatrix} \tag{C.7.5.1}$$

Step 2: Compute (c_X', c_Y') *according to the formula*

$$\begin{pmatrix} c_X' \\ c_Y' \end{pmatrix} = \pm \sqrt{\frac{r_X^2 r_Y^2 - r_X^2 y_1'^2 - r_Y^2 x_1'^2}{r_X^2 y_1'^2 + r_Y^2 x_1'^2}} \begin{pmatrix} \frac{r_X y_1'}{r_Y} \\ -\frac{r_Y x_1'}{r_X} \end{pmatrix} \tag{C.7.5.2}$$

where the + sign is chosen if $f_A \neq f_S$

and the - sign is chosen if $f_A = f_S$

Step 3: Compute (c_X, c_Y) *from* (c_X', c_Y')

$$\begin{pmatrix} c_X \\ c_Y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} c_X' \\ c_Y' \end{pmatrix} + \begin{pmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{pmatrix} \qquad \text{(C.7.5.3)}$$

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

$$\angle(\vec{u}, \vec{v}) = \pm \arccos \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} \qquad \text{(C.7.5.4)}$$

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:

$$\theta_1 = \angle\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{x_1' - c_X'}{r_X} \\ \frac{y_1' - c_Y'}{r_Y} \end{pmatrix} \right) \qquad \text{(C.7.5.5)}$$

$$\Delta\theta \equiv \angle\left(\begin{pmatrix} \frac{x_1' - c_X'}{r_X} \\ \frac{y_1' - c_Y'}{r_Y} \end{pmatrix}, \begin{pmatrix} \frac{-x_1' - c_X'}{r_X} \\ \frac{-y_1' - c_Y'}{r_Y} \end{pmatrix} \right) \bmod 360° \qquad \text{(C.7.5.6)}$$

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 (C.7.5.6) is > 0, then subtract 360°, whereas if $f_S = 1$ and the right side of (C.7.5.6) is < 0, then add 360°. In all other cases leave it as is.

C.7.6 Correction of out-of-range radii

This section formalizes the adjustments to out-of-range r_X and r_Y mentioned in C.7.2. Algorithmically these adjustments consist of the following steps:

Step 1: Ensure radii are non-zero

If $r_X = 0$ or $r_Y = 0$, then treat this as a straight line from (x_1, y_1) to (x_2, y_2) and stop. Otherwise,

Step 2: Ensure radii are positive

Take the absolute value of r_X and r_Y:

$$r_X \rightarrow \left| r_X \right| \qquad r_Y \rightarrow \left| r_Y \right| \tag{C.7.6.1}$$

Step 3: Ensure radii are large enough

Using the primed coordinate values of equation (C.7.5.1), compute

$$\Lambda = \frac{x_1'^2}{r_X^2} + \frac{y_1'^2}{r_Y^2} \tag{C.7.6.2}$$

If the result of the above equation is less than or equal to 1, then no further change need be made to r_X and r_Y. If the result of the above equation is greater than 1, then make the replacements

$$r_X \rightarrow \sqrt{\Lambda}\ r_X \qquad r_Y \rightarrow \sqrt{\Lambda}\ r_Y \tag{C.7.6.3}$$

Step 4: Proceed with computations

Proceed with the remaining elliptical arc computations, such as those in section C.7.5. Note: As a consequence of the radii corrections in this section, equation (C.7.5.2) 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 (C.7.6.3), the radicand of (C.7.5.2) is zero and there is exactly one solution for the center of the ellipse.

C.8 Text selection implementation notes

The following implementation notes describe the algorithm for deciding which characters are selected during a text selection operation.

The text selection operation determines the *start selection character(s)* and *start selection subregion* and the *end selection character(s)* and *end selection subregion*. To determine the *start selection character(s)*, the

SVG user agent determines which rendered glyph received the initial select event (e.g., the initial mouse down event) and which *character(s)* corresponds to the given glyph (note that for ligatures, a single glyph represents multiple characters). For all of the glyphs used to render the given *character(s)*, determine *start selection subregion* depending on whether the selection/pointer event occurred in the top/left, top/right, bottom/left or bottom/right subregion of the character cell area that encompasses all of the glyphs used to render the given *character(s)*.

The *end character(s)* and the relevant *end character(s) subregion* are determined using a similar method, except use the pointer location at the end of the select operation (e.g., when the user releases the given mouse button).

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 initiate text selection operations but do not initiate event handlers or links.

For horizontal text (i.e., when the baseline of the glyph is parallel to the primary text advance direction):

- If the *start selection subregion* is either the top/left or bottom/left, then the selection starts between the *start character(s)* and the previous character in visual rendering order for the 'text' element. (If this is the first character in visual rendering order for the 'text' element, then the selection starts with this first character in visual rendering order. Note that the bi-directional algorithm might result in the selection being between two characters that are not contiguous in lexical order.)
- If the *start selection subregion* is either the top/right or bottom/right, then the selection starts between the *start character(s)* and the next character in visual rendering order for the 'text' element. (If this is the last character in visual rendering order for the 'text' element, then the selection starts with this last character in visual rendering order. Note that the bi-directional algorithm might result in the selection being between two characters that are not contiguous in lexical order.)

For vertical text (i.e., when the baseline of the glyph is perpendicular to the primary text advance direction):

- If the *start selection subregion* is either the top/left or top/right, then the selection starts between the *start character(s)* and the previous character in visual rendering order for the 'text' element. (If this is the first character in visual rendering order for the 'text' element, then the selection starts with this first character in visual rendering order. Note that the bi-directional algorithm might result in the selection being between two characters that are not contiguous in lexical order.)
- If the *start selection subregion* is either the bottom/left or bottom/right, then the selection starts between the *start character(s)* and the next character in visual rendering order for the 'text' element. (If this is the last character in visual rendering order for the 'text' element, then the selection starts with this last character in visual rendering order. Note that the bi-directional algorithm might result in the selection being between two characters that are not contiguous in lexical order.)

When the user agent is implementing selection of bi-directional text in lexical order and the selection starts (or ends) between characters which are not contiguous in lexical 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.

Appendix D: Conformance Criteria

Contents

This appendix is informative, not normative.

D.1 Introduction

Different sets of SVG conformance criteria exist for:

- Conforming SVG Document Fragments
- Conforming SVG Stand-Alone Files
- Conforming SVG Included Documents
- Conforming SVG Generators
- Conforming SVG Interpreters
- Conforming SVG Viewers

D.2 Conforming SVG Document Fragments

An SVG document fragment is a *Conforming SVG Document Fragment* if it adheres to the specification described in this document (Scalable Vector Graphics (SVG) Specification) including SVG's DTD (see Document Type Definition) and also:

- (relative to XML) is well-formed
- if all non-SVG namespace elements and attributes and all xmlns attributes which refer to non-SVG namespace elements are removed from the given document, and if an appropriate `<?xml...?>` statement is included at the top of the document, and if an appropriate `<!DOCTYPE svg ... >` statement which points to the SVG DTD is included immediately

thereafter, the result is a valid XML document

- conforms to the following W3C Recommendations:
  - the XML 1.0 specification (Extensible Markup Language (XML) 1.0)
  - (if any namespaces other than SVG are used in the document) Namespaces in XML
  - any use of CSS styles and properties needs to conform to Cascading Style Sheets, level 2 CSS2 Specification
  - any references to external style sheets shall conform to Associating stylesheets with XML documents

D.3 Conforming SVG Stand-Alone Files

A file is a *Conforming SVG Stand-Alone File* if:

- it is an XML document
- its root element is an 'svg' element
- it conforms to the criteria for Conforming SVG Document Fragment

D.4 Conforming SVG Included Document Fragments

SVG document fragments can be included within parent XML documents using the XML namespace facilities described in Namespaces in XML.

An SVG document fragment that is included within a parent XML document is a *Conforming Included SVG Document Fragment* if the SVG document fragment, when taken out of the parent XML document, conforms to the SVG Document Type Definitions (DTD).

In particular, note that individual elements from the SVG namespace *cannot* be used by themselves. Thus, the SVG part of the following document is *not* conforming:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE SomeParentXMLGrammar PUBLIC "-//SomeParent"
  "http://SomeParentXMLGrammar.dtd">
<ParentXML>
  <!-- Elements from ParentXML go here -->

  <!-- The following is not conforming -->
  <z:rect xmlns:z="http://www.w3.org/Graphics/SVG/SVG-19991203.dtd"
            x="0" y="0" width="10" height="10" />

  <!-- More elements from ParentXML go here -->
</ParentXML>
```

Instead, for the SVG part to become a Conforming Included SVG Document Fragment, the file could be modified as follows:

```
<?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/Graphics/SVG/SVG-19991203.dtd"
         width="100px" height="100px" >
    <z:rect x="0" y="0" width="10" height="10" />
  </z:svg>

  <!-- More elements from ParentXML go here -->
</ParentXML>
```

D.5 Conforming SVG Generators

A *Conforming SVG Generator* is a program which:

- always creates at least one of Conforming SVG Document Fragments, Conforming SVG Stand-Alone Files or Conforming SVG Included Documents
- does not create non-conforming SVG document fragments of any of the above types
- conforms to all of the Priority 1 accessibility guidelines from the document "Authoring Tool Accessibility Guidelines 1.0" [ACCESS-AUTHOR] 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.

D.6 Conforming SVG Interpreters

An SVG interpreter is a program which can parse and process SVG document fragments. Examples of SVG interpreters are server-side transcoding tools (e.g., a tool which converts SVG content into a raster image) 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. An SVG Viewer does indeed parse and process SVG document fragments, where processing consists of rendering the SVG content to the target medium.

A *Conforming SVG Interpreter* is defined as follows:

- The interpreter must be able to successfully parse and process any Conforming SVG Document Fragment. (It is not required, however, that the semantics of every possible SVG feature be understood and supported beyond parsing. Thus, for example, a Conforming SVG Interpreter might only parse the defined syntax but not process the semantics of all features in the language.)
- The XML parser must be able to parse and process and XML constructs defined within [XML10] and [XML-NS].
- The XML parser must be able to parse and process arbitrarily long data streams.
- If the program allows scripts to run against Document Object Model, then a Conforming SVG Interpreter must support the entire DOM model for SVG defined in this specification

D.7 Conforming SVG Viewers

An SVG viewer is a program which can parse and process an SVG document fragment and render the contents of the document onto some sort of output medium such as a display or printer. Usually, an *SVG Viewer* is also an *SVG Interpreter*.

There are two sub-categories of *Conforming SVG Viewers*:

- *Conforming Static SVG Viewers* apply to platforms and environments which only render static documents, such as printers
- *Conforming Dynamic SVG Viewers* apply to platforms and environments such as common web browsers which support user interaction and content whose representation can change over time

Specific criteria that apply to both *Conforming Static SVG Viewers* and *Conforming Dynamic SVG Viewers*:

- In the typical case where the SVG Viewer is also an SVG Interpreter, then the program must also be a Conforming SVG Interpreter,
- All of SVG static rendering features corresponding to the feature name "SVGStatic" (see Feature strings) must be supported and rendered according to this specification.
- For interactive user environments, facilities must exist for zooming and panning of standalone SVG documents or SVG document fragments embedded within parent XML documents.
- 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., color must print correctly 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.
- The viewer must support alpha channel blending of the image of the SVG content onto the target canvas.
- The viewer must support the following W3C Recommendations with regard to SVG contents:
    - complete support for the XML 1.0 specification (Extensible Markup Language (XML) 1.0)
    - complete support for inclusion of non-SVG namespaces within SVG content Namespaces in XML (Note that data from non-SVG namespaces are included in the DOM but are otherwise ignored.)
    - complete support for all features from CSS2 (Cascading Style Sheets, level 2 CSS2 Specification) that are described in this specification as applying to SVG
    - complete support for external style sheets as described in Associating stylesheets with XML documents
- All visual rendering must be accurate to within one device pixel to the mathematically correct result.
- On systems which support accurate sRGB [SRGB] color, all sRGB color computations and all resulting color values must be accurate to within one sRGB color component value, where sRGB

color component values range from 0 to 255.

Although anti-aliasing support isn't a strict requirement for a Conforming SVG Viewer, it is highly recommended for display devices. Lack of anti-aliasing support will generally result in poor results on display devices.

Specific criteria that apply to only *Conforming Dynamic SVG Viewers*:

- In web browser environments, the viewer must have the ability to search and select text strings within SVG content.
- If display devices are supported, the viewer must have the ability to select and copy text from SVG content to the system clipboard
- The viewer must have complete support for an ECMAScript binding of the SVG Document Object Model.

The Web Accessibility Initiative [WAI] is defining "User Agent Accessibility Guidelines 1.0" [ACCESS-USERAGENTS]. 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 class 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:

- Generally, 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 using algorithms at least as good as bicubic resampling methods
- 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

---

Appendix E: Accessibility Support

Contents

This appendix is informative, not normative.

E.1 Accessibility and SVG

The degree to which SVG content can be considered accessible depends on what SVG is compared against and the manner in which the SVG content is constructed.

When comparing SVG content to images, SVG will be much more accessible for the following reasons:

- Text strings in SVG are represented as regular XML character data rather than bits in an image. (See Text.)
- At any place in the SVG hierarchy, a drawing can include a long set of descriptive text (i.e., the 'desc' element) and/or a short description in the form of a title (i.e., the 'title' element). Both of these features can be used to help the visually impaired interpret both the intent and specific content of a drawing. The drawing can be architected such that there is a single description for the drawing as a whole or there are multiple descriptions which are distributed within the drawing and describe each separate component within the drawing.
- SVG has a notion of structure from its grouping constructs such as the 'g' element. This grouping constructs, when used in conjunction with the 'desc' and 'title' elements, provide information about document structure and semantics.
- Personal style sheets can be used to adjust the color contrast of graphic elements.
- Because SVG content is scalable, people with partial visual impairment will be able to magnify the content or zoom in on graphics for easier viewing.
- This specification includes a set of SVG accessibility guidelines, which will help to promote the creation of accessible SVG content.
- SVG's Conformance Guidelines:
    - requires support of Priority 1 authoring tool accessibility guidelines in Conforming SVG Generators
    - encourages support of Priority 1 user agent accessibility guidelines in Conforming SVG Viewers

thereby promoting increased accessibility for both generation and viewing of SVG content.

On the other hand, when comparing SVG to other markup languages, SVG can be less accessible. For example, in most cases, an XHTML [XHTML10] document will be more accessible as XHTML than it would be if converted into SVG since the higher-level structure and semantics will be lost in the translation.

The degree to which SVG content is accessible depends on the degree to which the document's author follows the SVG accessibility guidelines.

E.2 Aural style sheets

For the purposes of aural media, SVG represents a stylable XML grammar. In user agents that support aural style sheets, aural style properties [CSS2-AURAL] can be applied as defined in [CSS2].

Aural style properties can be applied to any SVG element that can contain character data content, including 'desc', 'title', 'tspan'. 'tref' and 'textPath'. On user agents that support aural style sheets, the following [CSS2] properties can be applied:

'azimuth'	[CSS2-azimuth]
'cue'	[CSS2-cue]
'cue-after'	[CSS2-cue-after]
'cue-before'	[CSS2-cue-before]
'elevation'	[CSS2-elevation]
'pause'	[CSS2-pause]
'pause-after'	[CSS2-pause-after]
'pause-before'	[CSS2-pause-before]
'pitch'	[CSS2-pitch]
'pitch-range'	[CSS2-pitch-range]
'play-during'	[CSS2-play-during]
'richness'	[CSS2-richness]
'speak'	[CSS2-speak]
'speak-header'	[CSS2-speak-header]
'speak-numeral'	[CSS2-speak-numeral]
'speak-punctuation'	[CSS2-speak-punctuation]
'speech-rate'	[CSS2-speech-rate]
'stress'	[CSS2-stress]
'voice-family'	[CSS2-voice-family]
'volume'	[CSS2-volume]

For user agents that support aural style sheets and also support [DOM2], the user agent is required to support the DOM interfaces defined in [DOM2-CSS] that correspond to aural properties [CSS2-AURAL]. (See Relationship with DOM2 CSS object model.)

E.3 SVG accessibility guidelines

The definition of a Conforming SVG Generator requires that it adhere to "Authoring Tool Accessibility Guidelines 1.0" [ACCESS-AUTHOR].

The following are additional SVG-specific accessibility guidelines:

- 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 that higher-level information be made available somehow. The latter information is necessary for purposes of accessibility and intelligent textual processing. Alternatives include:
  - Provide access to the original source document which convey's the document's structure and semantics.
  - Express the document in another markup language which conveys the document's structure and semantics.
  - Express the document using a combination SVG with other XML namespaces where elements and attributes from other namespaces convey the document's structure and semantics and the SVG namespace is used to represent the information graphically.
  - Express the document solely using SVG, but utilize SVG's grouping constructs (e.g., the 'g' element) to represent document structure and use the 'desc' and 'title' elements to describe the document semantics.
- Wherever supplemental descriptive information is available about part of the SVG content, it is strongly recommended that the descriptive information be included within a 'desc' or 'title' element. This guideline also applies to 'text' elements, as the actual rendered text string might not convey the higher-level structure and semantics of the character data content.
- Drawing programs often provide some sort of document structuring capabilities, such as the ability to create and name layers of graphics, and authors often assign meaningful names to these structural components for their purposes of managing their own data. For example, an author might create a drawing which has three layers: a lake (the background layer), an island (drawn on top of the lake), and a tree (drawn on top of the island), with the names of the layers being "lake", "island" and "tree". It is recommended that SVG content convey as best as possible both the inherent structure of the original drawing and any names of the structural components. In particular, it is recommended that drawing tools which support the concept of named layers create SVG content which create a distinct 'g' element for each layer and that a 'title' element be provided for each layer's 'g' element, where the 'title' element contains the name of the given layer.
- For SVG viewers, it is recommended that 'title' elements be rendered as popup 'tooltips' when the pointing device is positioned over the given graphics or container element. For visual rendering environments, the tooltip might appear within a transient popup window. For aural rendering environments, the tooltip might be rendered aurally.

Appendix F: Internationalization Support

Contents

This appendix is informative, not normative.

F.1 Internationalization and SVG

SVG is an application of XML [XML10] and thus supports Unicode [UNICODE] [UNICODE21], which provides universal 16-bit encoding for the scripts of the world's principal languages.

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:

- access to glyphs which are not defined in standard Unicode [UNICODE]
- the ability to follow the guidelines for normalizing character data for the purposes of enhanced interoperability (see [CHARMOD]), while still having precise control over the glyphs that are drawn.

SVG supports:

- Horizontal, left-to-right text found in Roman scripts (see the 'writing-mode' property)
- Vertical and vertical-ideographic text (see the 'writing-mode' property)
- Arabic bi-directional text (see the 'direction' and 'unicode-bidi' properties)

SVG fonts support alternate glyphs for Arabic and Han text.

Multi-language SVG documents are possible by utilizing the system-language attribute to have different text strings appear based on the user's language setting.

F.2 SVG Internationalization Guidelines

SVG generators should follow W3C guidelines for normalizing character data [CHARMOD] and should use the facilities for Alternate glyphs to override the standard glyphs used to represent normalized character data with specified glyphs.

Appendix G: 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 don't have to be supplied when tokens can be identified implicitly; smooth curve formulations are available (cubic beziers, quadratic beziers and elliptical arcs) to prevent the need to tesselate into polylines; and shortcut formulations exist for common forms of cubic bezier segments, quadratic bezier 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
- SVG supports CSS selectors and property inheritance, which allows commonly used sets of attributes to be defined once as named styles.
- Filter effects allow for compelling visual results and effects typically found only in image-authoring tools using small amounts of vector and/or raster data

Additionally, HTTP 1.1 allows for compressed data to be passed from server to client, which can result in significant file size reduction. Here are some sample compression results using gzip compression on SVG documents:

Uncompressed SVG	With gzip compression	Compression ratio
30,203	8,680	71%
12,563	8,048	36%
7,106	2,395	66%
6,216	2,310	63%
4,381	2,198	50%

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 Web fonts 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 bezier segments whenever possible; eliminate extraneous white space and separators.
- Utilize symbols if the same graphic appears multiple times in the document
- 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 H. References

Contents

H.1 Normative references

[COLORIMETRY]

"Colorimetry, Second Edition", CIE Publication 15.2-1986, ISBN 3-900-734-00-3. Available at http://www.hike.te.chiba-u.ac.jp/ikeda/CIE/publ/abst/15-2-86.html.

[CSS2]

"Cascading Style Sheets, level 2", B. Bos, H. W. Lie, C. Lilley, I. Jacobs, 12 May 1998. Available at http://www.w3.org/TR/REC-CSS2.

Specific topics:

- **[CSS2-ATRULES]** CSS2 At-rules
- **[CSS2-POSN]** CSS2 positioning properties
- **[CSS2-LAYOUT]** CSS2 positioning properties
- **[CSS2-DYNPSEUDO]** CSS2 dynamic pseudo-classes
- **[CSS2-AURAL]** aural media
- **[CSS2-VISUAL]** visual media
- **[]** CSS2 dynamic pseudo-classes
- **[CSS2-azimuth]** CSS2 'azimuth' property definition
- **[CSS2-clip]** CSS2 'clip' property definition
- **[CSS2-cue]** CSS2 'cue' property definition
- **[CSS2-cue-after]** CSS2 'cue-after' property definition
- **[CSS2-cue-before]** CSS2 'cue-before' property definition
- **[CSS2-display]** CSS2 'display' property definition
- **[CSS2-elevation]** CSS2 'elevation' property definition
- **[CSS2-overflow]** CSS2 'overflow' property definition
- **[CSS2-pause]** CSS2 'pause' property definition
- **[CSS2-pause-after]** CSS2 'pause-after' property definition

- **[CSS2-pause-before]** CSS2 'pause-before' property definition
- **[CSS2-pitch]** CSS2 'pitch' property definition
- **[CSS2-pitch-range]** CSS2 'pitch-range' property definition
- **[CSS2-play-during]** CSS2 'play-during' property definition
- **[CSS2-richness]** CSS2 'richness' property definition
- **[CSS2-speak]** CSS2 'speak' property definition
- **[CSS2-speak-header]** CSS2 'speak-header' property definition
- **[CSS2-speak-numeral]** CSS2 'speak-numeral' property definition
- **[CSS2-speak-punctuation]** CSS2 'speak-punctuation' property definition
- **[CSS2-speech-rate]** CSS2 'speech-rate' property definition
- **[CSS2-stress]** CSS2 'stress' property definition
- **[CSS2-voice-family]** CSS2 'voice-family' property definition
- **[CSS2-volume]** CSS2 'volume' property definition

[DOM1]

"Document Object Model (DOM) Level 1 Specification", V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, L. Wood, editors, 1 October 1998.
Available at http://www.w3.org/TR/REC-DOM-Level-1/

[DOM2]

"Document Object Model (DOM) Level 2 Specification", V. Apparao, M. Champion, A. Le Hors, T. Pixley, J. Robie, P. Sharpe, C. Wilson, L. Wood, editors, 4 March 1999.
Available at http://www.w3.org/TR/WD-DOM-Level-2-19991025
Specific topics:

- **[DOM2-CORE]** Document Object Model Core
- **[DOM2-HTML]** Document Object Model HTML
- **[DOM2-VIEWS]** Document Object Model Views
- **[DOM2-SHEETS]** Document Object Model StyleSheets
- **[DOM2-CSS]** Document Object Model CSS
  - **[DOM2-CSSVALUE]** Document Object Model CSS - Interface CSSValue
  - **[DOM2-CSS-RGBCOLOR]** Document Object Model CSS - Interface RGBColor
  - **[DOM2-CSS-EI]** Document Object Model CSS - Extended Interfaces
  - **[DOM2-CSS2Azimuth]** Interface CSS2Azimuth
  - **[DOM2-CSS2Cursor]** Interface CSS2Cursor
  - **[DOM2-CSS2PlayDuring]** Interface CSS2PlayDuring
- **[DOM2-EVENTS]** Document Object Model Events
  - **[DOM2-EVREG]** Event registration interfaces

        - **[DOM2-EVTARGET]** Interface EventTarget
        - **[DOM2-EVLISTEN]** Interface EventListener
        - **[DOM2-EVCAPTURE]** Event capture
        - **[DOM2-EVBUBBLE]** Event bubbling
        - **[DOM2-UIEVENTS]** Interface UIEvent
        - **[DOM2-MOUSEEVENTS]** Interface MouseEvent
        - **[DOM2-KEYEVENTS]** Interface KeyEvent
        - **[DOM2-MUTEVENTS]** Interface MutationEvent
        - **[DOM2-HTMLEVENTS]** HTML event types
    - **[DOM2-TRAV]** Document Object Model Traversal
    - **[DOM2-RANGE]** Document Object Model Range

[ESS]

"Associating Style Sheets with XML documents Version 1.0", James Clark, editor, 29 June 1999.
Available at http://www.w3.org/TR/xml-stylesheet/.

[IEEE-754]

"?????", 19??.
Available at ???.

[ICC32]

"ICC Profile Format Specification, version 3.2", 1995.
Available at ftp://sgigate.sgi.com/pub/icc/ICC32.pdf.

[PNG10]

"PNG (Portable Network Graphics) Specification, Version 1.0 specification", T. Boutell ed., 1 October 1996.
Available at http://www.w3.org/pub/WWW/TR/REC-png-multi.html.

[RFC1766]

"Tags for the Identification of Languages", H. Alvestrand, March 1995.
Available at ftp://ftp.isi.edu/in-notes/rfc1766.txt.

[RFC2045]

"Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996.
Available at ftp://ftp.internic.net/rfc/rfc2045.txt. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

[RFC2119]

"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Available at http://www.ietf.org/rfc/rfc2119.txt.

[SMIL1]

"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", P. Hoschka, editor, 15 June 1998.

Available at http://www.w3.org/TR/REC-smil/
Specific topics:

- **[SMIL10-SYSLANG]** 'system-language' attribute

[SMILAnim]

"SMIL Animation", P. Schmitz, K. Day, A. Cohen, P. Hoschka, editors, 02 September 1999.
Available at http://www.w3.org/TR/smil-animation/
Specific topics:

- **[SMILANIM-TARGET]** Specifying the animation target
- **[SMILANIM-ANIMFUNC]** Specifying the animation function
- **[SMILANIM-AD]** Computing the Active Duration
- **[SMILANIM-UNIFY]** Unifying Event-based and Scheduled Timing
- **[SMILANIM-ADD]** Additive Animation
- **[SMILANIM-ACCUM]** Controlling behavior of repeating animation - Cumulative Animation
- **[SMILANIM-FROMTOBY-ADD]** How from, to and by attributes affect additive behavior
- **[SMILANIM-LINKS]** Hyperlinks and Timing
- **[SMILANIM-TRANSITIONS]** State Transition Model
- **[SMILANIM-ATTR-BEGIN]** 'begin' attribute
- **[SMILANIM-ATTR-DUR]** 'dur' attribute
- **[SMILANIM-ATTR-END]** 'end' attribute
- **[SMILANIM-ATTR-ENDACTIVE]** 'endActive' attribute
- **[SMILANIM-ATTR-RESTART]** 'restart' attribute
- **[SMILANIM-ATTR-REPEATCOUNT]** 'repeatCount' attribute
- **[SMILANIM-ATTR-REPEATDUR]** 'repeatDur' attribute
- **[SMILANIM-ATTR-FILL]** 'fill' attribute
- **[SMILANIM-ATTR-VALUES]** Specifying function values
- **[SMILANIM-ATTR-ORIGIN]** 'origin' attribute
- **[SMILANIM-DOM-METHODS]** Supported methods

[SRGB]

"A Standard Default Color Space for the Internet - sRGB", M. Stokes, M. Anderson, S. Chandrasekar, R. Motta.
Available at http://www.w3.org/Graphics/Color/sRGB.

[UNICODE]

"The Unicode Standard: Version 2.0", The Unicode Consortium, Addison-Wesley Developers Press, 1996.

[UNICODE21]

"Unicode Technical Report # 8, The Unicode Standard, Version 2.1", September 1998. Available at: http://www.unicode.org/unicode/reports/tr8.html.

The latest version of Unicode. For more information, consult the Unicode Consortium's home page at http://www.unicode.org/. For bidirectionality, see also the corrigenda at http://www.unicode.org/unicode/uni2errata/bidi.htm.

[URI]

"Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998.
Available at http://www.ics.uci.edu/pub/ietf/uri/rfc2396.txt. (The term "URI-reference" is defined in Section 4: URI References.)

[XLINK]

"XML Linking Language (XLink)", S. DeRose, D. Orchard, B. Trafford, editors, 26 July 1999.
Available at http://www.w3.org/1999/07/WD-xlink-19990726

[XML10]

"Extensible Markup Language (XML) 1.0" T. Bray, J. Paoli, C.M. Sperberg-McQueen, editors, 10 February 1998.
Available at http://www.w3.org/TR/REC-xml/.

[XML-NS]

"Namespaces in XML" T. Bray, D. Hollander, A. Layman, editors, 14 January 1999.
Available at http://www.w3.org/TR/REC-xml-names/.

[XPTR]

"XML Pointer Language (XPointer)", S. DeRose, R. Daniel Jr., editors, 09 July 1999.
Available at http://www.w3.org/1999/07/WD-xptr-19990709

H.2 Informative references

[ACCESS-AUTHOR]

"Authoring Tool Accessibility Guidelines 1.0", J. Treviranus, J. Richards, I. Jacobs, C. McCathieNevile, editors, 26 October 1999.
Available at http://www.w3.org/TR/WAI-AUTOOLS/

[ACCESS-USERAGENTS]

"User Agent Accessibility Guidelines 1.0", J. Gunderson, I. Jacobs, editors, 11 August 1999.
Available at http://www.w3.org/TR/WAI-USERAGENT/

[CHARMOD]

"Character Model for the World Wide Web (working draft)", M. Dürst, editor, 25 February 1999.
Available at http://www.w3.org/TR/1999/WD-charmod-19990225

[FOLEY-VANDAM]

"Computer Graphics : Principles and Practice, Second Edition", James D. , Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, Addison-Wesley, pp. 488-491.

[HTML40]

"HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs, 8 July 1997.

Available at http://www.w3.org/TR/REC-html40/. The Recommendation defines three document type definitions: Strict, Transitional, and Frameset, all reachable from the Recommendation.

[RFC2068]

"HTTP Version 1.1 ", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, and T. Berners-Lee, January 1997.
Available at ftp://ftp.internic.net/rfc/rfc2068.txt.

[MATHML]

"Mathematical Markup Language (MathML) 1.01 Specification", P. Ion, R. Miner, 7 July 1999.
Available at http://www.w3.org/1999/07/REC-MathML-19990707/.

[WAI]

Home page for Web Accessibility Initiative:
http://www.w3.org/wai/.

[WAI-PAGEAUTH]

"WAI Accesibility Guidelines: Page Authoring" for designing accessible documents are available at:
http://www.w3.org/TR/WD-WAI-PAGEAUTH.

[XHTML10]

"XHTML(tm) 1.0: The Extensible HyperText Markup Language",
Available at http://www.w3.org/TR/xhtml1/.

[XSL]

"Extensible Stylesheet Language (XSL) Specification", S. Deach, editor, 21 Apr 1999.
Available at http://www.w3.org/TR/WD-xsl/

[XSLT]

"XSL Transformations (XSLT) Version 1.0", J. Clark, editor, 08 October 1999.
Available at http://www.w3.org/TR/1999/PR-xslt-19991008

Appendix I: Change History

Changes with the 03-December-1999 SVG Draft Specification

- Global/miscellaneous changes
  - Considerable miscellaneous editorial cleanup throughout, including significant rearrangement of content and reordering of chapters.
  - Provided considerable more detail about SVG's DOM interfaces. The DOM interfaces are not provided at the end of the chapters to which they most relate.
  - Added a new chapter called Basic Data Types and Interfaces which provides a single definition of things such as <number> and <length>.
  - Changed all references to element start tags such as "the <svg> element" with single-quoted references such as "the 'svg' element".
  - Added an indication for each attribute and property about whether it can be animated.
  - Added a navigation bar at the bottom of all pages.
- Changes to Introduction
  - Added a new section 1.6 Error processing which describes user agents error handling.
  - Modified the wording about relationship to SMIL to emphasize that SVG static or animated content is designed to be used as a component media for future versions of SMIL.
  - Changed the name of section 1.4 from Terminology to Definitions to match the approach taken in the CSS spec. Removed definition of "number" as this is now defined under Basic Data Types.
  - Added a new section 1.5 called Terminology to Terminology to define the terms "must", "required", etc.
- Changes to Basic Data Types and Interfaces
  - Provided more explicit range information on integers and numbers.
  - Modified all DOM interfaces to use "float" instead of "double" per decision that conforming implementations need only support single-precision floating point operations.
- Changes to SVG Rendering Model
  - Added a note about markers drawing in order according to the directionality of the shape, and added a note that marker symbols are rendered as if their graphics content were expanded into the document tree..

- Changes to Document Structure
  - Modified the wording in the section on URI references to say that invalid references are errors and handled by standard user agent error processing behavior, per decision by the working group to have consistent error handling.
  - Removed methods open, close, write, writeln from SVGSVGElement as these will be obsoleted for HTML down the road and equivalent capability is possible by manipulating the DOM tree.
  - For interface SVGDocument, added a forceRedraw() method.
  - Modified the description of SVGDocument to clarify that an SVGDocument object only exists when the 'svg' element is the root element of the document hierarchy,
  - Moved various utility functions that had been part of interface SVGDocument to interface SVGSVGElement since SVGDocument will not exist in all cases.
  - Clarified that an 'image' creates its own document tree and doesn't inherit properties.
  - Moved 'switch', 'system-required' and the new 'system-language' to Document Structure.
  - Added attribute system-language, from SMIL 1.0, for all elements that have attribute system-required to provide the mechanism for multiple localized text strings for the same document, as noted in previous draft specifications. Documented in The system-language attribute
  - Defined SVG's available feature strings for "hasFeature" method call and "system-required" attribute.
  - Modify system-required to be NMTOKEN instead of CDATA to match SMIL 1.0. Modify example to use an NMTOKEN instead of a URL.
- Changes to Styling
  - Changed the title of the chapter from "Styling and CSS" to "Styling" to re-inforce that SVG content can be styled by either CSS or XSL.
  - Added an introduction talking about styling with both XSL and CSS.
  - Modified many descriptions so that they apply to both CSS and XSL.
  - Added descriptions for the 'style' and 'class' attributes.
  - Modified the description of the 'display' property to eliminate the SVG-specific value "svg" and to say that anything other than 'none' indicates that the given element can be rendered.
  - Indicate that CSS dynamic pseudo classes :hover, :active, :focus are supported.
  - Added a bullet about aural style sheet support.
  - The 'overflow' and 'clip' properties have been moved to the Clipping and Masking chapter. (See Masking notes below.)
  - Modified the default style sheet so that the overflow property has an initial value of 'hidden' for all elements in the SVG language. This is necessary to achieve the default behavior that new viewports also establish new clipping paths.
  - Modified the default style sheet to eliminate the reference to 'display:svg'. Until CSS3, which is expected to provide general purpose methods for associating user agents with particular XML namespaces, user agents will have to have special detection for SVG documents and fragments.

    - In the default style sheet, included all methods which support the viewBox attribute to have a default value of 'hidden' for the 'overflow' property.
- Changes to Coordinate Systems, Transformations and Units
    - Major editorial cleanup. Switch to column vectors instead of row vectors. Change wording per suggestions from various members of the working group. Added many examples and supportive pictures.
    - Fix errors in Pythagorean formula examples where a comma needed to be replaced by a plus-sign.
    - Fixed a long-standing error in the section describing how to set up a new viewport where the example and some of the write-up referred to using CSS properties left/right/top/bottom/etc instead of attributes x/y/width/height. Now this chapter is correct and consistent with the descriptions of the x/y/width/height attributes for the 'svg' element.
- Changes to Paths
    - Removed the erroneous reference to the "B" and "b" commands, which don't exist in the current specification.
    - Fixed BNF to remove possibility of trailing comma at the end of a command, such as "M 100,100, L 200,200,". This is no longer valid, and instead you have to remove the extraneous commas, as in "M 100,100 L 200,200".
    - Changed the path data BNF to eliminate ambiguous productions (i.e., when two productions could satisfy a given path specification).
    - Added a note indicating that the path data parser needs to be greedy (i.e., consume all characters that continue to satisfy a production).
    - Added a comment to the distance-along-a-path chapter stating that "moveto" operations do not contribute to the distance calculations.
    - Removed "flatness" attribute due to statement in conformance criteria that rendering needs to be within a device pixel.
    - Defined animation behavior for booleans that are part of the arc command.
    - Enhanced the write-up of attribute nominalLength and clarified "distance-along-a-path."
    - Fixed errors in SVG DOM where it was inconsistent with our newer arc command.
    - For interface SVGPathElement, added distance-along-a-path utiltiy functions getTotalLength(), getPointAtLength() and getPathSegAtLength().
- Changes to Basic shapes
    - Thorough editorial cleanup of this chapter, providing more precise language and illustrating with pictures.
    - Fixed BNF to remove possibility of trailing comma at the end of the points list, such as "100,100,200,200,". This is no longer valid, and instead you have to remove the extraneous ending comma, as in "100,100,200,200".
- Changes to Text
    - Major cleanup, reorganization and rewriting of sections in preparation of Proposed Recommendation. Sections have been renumbered and content has been shuffled between sections. Very detailed descriptions have been provided for all layout rules, including text on a path and text selection. Several drawings have been added.

- Incorporation of Last Call feedback from the I18N interest group, Last Call internationalization feedback from the XSL working group, and Last Call feedback from the CSS and FP working groups. Detailed changes are listed as follows.
- Various editorial clean-up actions, including more careful use of the terms "character" and "glyph".
- Added a section entitled Characters and their corresponding glyphs to clearly describe the differences and relationships between XML characters and glyphs.
- Broke up the 'tspan' element into two separate elements, 'tspan' and 'tref', per Last Call feedback from the I18N interest group. The 'href' attribute has been removed from the 'tspan' element. All referenced to non-embedded character data are done with the 'tref' element. The 'tref' element is not allowed to have embedded character data.
- Changed the descriptions of the x,y,dx,dy and rotate attributes to the 'tspan' element to correspond to characters rather than glyphs, per Last Call feedback from the I18N interest group.
- Wrote up how x,y,dx,dy and rotate attributes on the 'tspan' element work with the bi-directionality algorithm, reflecting Last Call feedback from the XSL and I18N working groups.
- Substituted SVG-specific property 'text-anchor' for 'text-align' and 'glyph-anchor' for 'vertical-align', per Last Call feedback comments from the I18N interest group and the XSL working group. Both 'text-align' and 'vertical-align' are meant for block text and did not fit with SVG's single-line approach to text-at-a-point and text-on-a-path.
- Added property 'baseline-shift' per Last Call feedback comments from the I18N interest group and the XSL working group. The XSL working group is still developing the definition of this property. The current definition in the SVG spec is a placeholder which might require modifications as the XSL definition of the property gets further formulated.
- Allow for nested 'tspan' elements so that 'baseline-shift' properties can be nested, to allow for (as an example) exponents of exponents.
- Replaced properties textPath-transform and orient-to-path properties with attribute 'rotate', which directly addresses the desired feature (i.e., precisely specified text-on-a-path when the SVG text-on-a-path algorithm doesn't provide sufficient precision), doesn't introduce additional unnecessary and hard-to-use capability, and which has a parallel construction with the dx and dy attributes.
- Changed 'altglyph' from a property to an element, 'altGlyph', per Last Call feedback comments from the I18N interest group and the XSL working group. To complete the definition of the altglyph capability, other new elements include 'altGlyphDef' and 'glyphSub'. 'altGlyph' can appear within a 'text' element and refers to either a glyph defined in an SVG font or an 'altGlyphDef' element. 'altGlyphDef' is a child of 'defs'.
- Removed the ability to include a 'use' element as a child of a 'text', 'tspan' or 'textPath' element, per discussions derivative of Last Call feedback on internationized text issues. Instead, it was felt that SVG-on-a-path was better accomplished by including the SVG-on-a-path as a glyph defined in the private section of Unicode.
- Added a sentence in Relationship with bi-directionality indicating that the default orientation for Arabic text when rendered in a vertical text string is rotated 90 degrees counter-clockwise, which is the same as for Roman text.
- Rephrased the description of the W3C character model in the section on Alternate glyphs

to avoid the impression that normalization eliminates compatibility equivalents, per Last Call feedback comments from the I18N interest group.

- Made clear that the x,y,dx and dy attributes on the 'tspan' element can be used for both minor and major adjustments of the current text position, per Last Call feedback comments from the I18N interest group.
- Made clear that text selection in bi-directional situations selects text that is contiguous in lexical order, with user agents given the option of providing an alternative text selection facility in visual rendering order. This addresses Last Call feedback comments from the I18N interest group.
- For White space, modified the rule for xml:space='default' to throw out all line breaks. This accommodates Last Call feedback comments from the I18N interest group that many languages would not want automatic generation of white space if character data were spread across multiple lines. By making this change, then Roman text will need to include explicit white space wherever word separators are needed (possibly achieved by simply indenting the text data).
- Removed the paragraph about the "uu" suffix with the font shorthand property. This paragraph was in the previous spec only because of an editorial error.
- Added a paragraph to Introduction recommending the use of appropriate semantic markup along with 'text' elements to make SVG documents more accessible, per Last Call feedback comments from Daniel Dardailler of the Web Accessibility Initiative (WAI).

- Changes to Painting: Filling, Stroking and Marker Symbols
  - Added definitions of properties 'color-interpolation' and 'color-rendering' to control whether color computations are performed in the sRGB or linearRGB color spaces and to provide speed/quality rendering hints to the user agent.
  - Added 'inherit' as a value for 'fill-opacity'. (It already existed for 'stroke-opacity'.)
  - Added a note to stroke properties that all stroke operations must begin at the start of the graphics element and must employ the user agent's distance-along-a-path algorithms.
  - Added a note that complex paint servers such as gradients and patterns must produce the same result as if the stroke were converted to a 'path' and then filled with the given paint server.
  - Added 'userSpaceOnUse' to markerUnits per working group decision to enhance re-usability of referenced elements used in 'defs' and to achieve consistency across all types of referenced elements.
  - Added short descriptions of the available values for 'color-interpolation', which were missing in previous drafts.
  - Added a note that ICC color values cannot be expressed as percentages and indicates that ICC color values are <number>s.
  - Reworded descriptions of 'stroke-opacity' and 'fill-opacity' about value clamping.
  - Copied explicit formulas for converting sRGB to linearRGB from http://www.w3.org/Graphics/Color/sRGB.
  - For Rendering Properties, changed attribute value 'default' to 'auto' per Last Call comments from the CSS and FP working groups.
  - When a URI is provided for 'fill' or 'stroke', there used to be an 'inherit' value available as

a back-up option in case the URI were invalid. Because the 'inherit' backup option causes serious complications with regard to the DOM, this little used option has been removed.

- Changes to Color
  - Merged properties 'color-profile' and 'rendering-intent' into a "color profile description" described by an @color-profile construct. As a result, ICC color definitions now take an additional initial parameter, which is the name of the profile to use.
  - Fixed a couple of errors with the 'color' property. First, removed the reference to the 'color' property being used for gradient stops. (Actually, it is the 'stop-color' property that applies to gradient stops.) Second, removed the 'icc-color' option, as this was present only to address the needs of gradient stops. (This removal also serves to sidestep the issue of why SVG is extending this very widely used existing CSS2 property.)
- Changes to Gradients and Patterns
  - Added xlink:href attribute to gradient and pattern elements, which allows one gradient/pattern to inherit attributes and (in the case of gradients) a gradient ramp from a previously defined element of the same type. This addition promotes re-usability of gradients and patterns and thus promotes more compact files.
  - Added 'userSpaceOnUse' to gradientUnits and patternUnits per working group decision to enhance re-usability of referenced elements used in 'defs' and to achieve consistency across all types of referenced elements.
  - Fixed examples of gradient stops which erroneously used 'stop-color' instead of 'color'.
- Changes to Clipping, Masking and Compositing
  - Added 'inherit' values to 'opacity', 'mask' and 'clip-path' to correct an error in the spec.
  - Removed percentage values from the 'opacity' property per Last Call comments from the CSS and FP working groups.
  - Added a drawing which illustrates various 'opacity' settings on objects and groups. The drawing illustrates the accumulative effects of opacity, which addresses one of the Last Call comments from the CSS and FP working groups.
  - Added explicit alpha blending formulas to Simple alpha blending/compositing.
  - Removed wording about approximating sRGB with 2.2 exponent since precise formulas involve little extra computation.
  - Changed wording for the 'mask' element to clarify exactly when one-channel masking happens and when luminance-to-alpha processing happens, and changed the formulas for luminance-to-alpha to use the same formulas as the feColorMatrix filter effect.
  - The 'overflow' and 'clip' properties have been moved to this section.
  - Elaborated on the 'overflow' and 'clip' properties. In particular, the specification now states explicitly state in words that the effect of the default style sheet having svg { overflow:hidden } is that the default behavior has the SVG user agent clipping to the bounds of the initial viewport. Also, discuss the effect of these properties on embedded 'svg' elements and the difference between clipping to the viewBox versus clipping to the viewport.
  - Added clipPathUnits="userSpace|userSpaceOnUse|objectBoundingBox" and added 'userSpaceOnUse' to maskUnits per working group decision to enhance re-usability of referenced elements used in 'defs' and to achieve consistency across all types of referenced elements.

    - For 'clipPath', changed wording on invalid references to say that standard error processing would apply.
    - Added clarification for the 'clipPath' element and clipping path inheritance rules (i.e., a 'clipPath' does not inherit clipping paths from its ancestors, but it can use the 'cli-path' property to explicitly indicate that the clipping path itself must be clipped.).
    - Changed the wording for the 'mask' and 'clip-path' properties to say that invalid references are errors and handled by standard user agent error processing behavior, per decision by the working group to have consistent error handling.
    - Reworded descriptions of 'stroke-opacity' and 'fill-opacity' about value clamping.
- Changes to Filter Effects
    - Added xlink:href attribute to the 'filter' element, which allows one filter to inherit attributes and the filter effect definition from a previously defined 'filter' element. This addition promotes re-usability of filters and thus promotes more compact files.
    - Added 'userSpaceOnUse' to filterUnits per working group decision to enhance re-usability of referenced elements used in 'defs' and to achieve consistency across all types of referenced elements.
    - Replaced the old initial example with one that actually works.
    - For feColorMatrix, changed hue-rotate and luminance-to-alpha to hueRotate and luminateToAlpha to conform to camel notation conventions found in the rest of the spec.
    - Added 'inherit' values to 'enable-background' to correct an error in the spec.
    - Fixed typos in description of hueRotate in feColorMatrix to say [a00 a01 a02] [a10 a11 a12] [a20 a21 a22].
    - Added x, y, width, height attributes to all filter effects, per resolution by SVG working group to provide sufficient information for feImage and feTile to know what to do.
    - Added stitchTiles attribute and changed baseFrequency to baseFrequencyX and baseFrequencyY in feTurbulence to allow for small tiles of generated noise which can be stitched together.
    - Fixed error in example from 110% to 120%.
- Changes to Interactivity
    - Expanded the Introduction.
    - Included detailed rules for handling pointer events.
    - Added property 'pointer-events', which the working group decided to add at the Ottawa face-to-face meeting but which did not make it into the Last Call specification due to an editorial error.
    - Modified the description for 'cursor' element to explain that its primary purpose is to provide for a platform-independent cursor adding a hot spot to a PNG used as the source image per Last Call comments from the CSS and FP working groups.
    - Defined terms zoom, pan and magnify. Emphasized that zoom and pan are required for user agents in interactive environments, and that magnification is recommended. Documented that zoom does not change meaning of CSS units, per working group decision.
    - Renamed "allowZoomAndPan" to "enableZoomAndPanControls" per working group decision.

- Changes to Scripting
  - Added event attributes onfocusin, onfocusout, ongainselection, onloseselection, onactivate, onresize and onscroll to match events in DOM2.
  - In the description of Document events, modified the description of "onload" to mean that the element and its descendants are ready to be rendered but that external resources are not necessarily available yet, per working group decision.
- Changes to Animation
  - Major update to track new versions of the SMIL-Animation specification.
  - Added the 'set' element from SMIL animation
  - Dropped the 'animateFlipbook' element per request from the SYMM working group, since the same functionality will be available using other facilities and concerns about 'animateFlipbook' being too close to the SMIL 'par' element.
  - Added a type attribute to 'animateTransform' to explicitly disallow mixing of different types of transformations within the same animation element.
  - Added an "auto-reflect" option to the rotate attribute to make it easy to pick with which side of the motion path the target element will rotate.
  - Add keyPoints as an additional SVG extension to compensate for SMIL Animation's changes in semantics to 'animateMotion' since the SVG Last Call draft of 12Aug1999.
  - Added some pictures.
  - Indicated that the 'color-interpolation' property applies to color interpolations that result from 'animateColor'.
- Changes to Linking
  - Removed the #FIXED setting on 'xlink:show' on the 'a' element to allow for "new" vs. "replace" per Last Call comments from the CSS and FP working groups.
  - Changed 'parsed' to 'embed' to match upcoming expected revisions to XLink.
  - Added 'target' attribute to 'a' element.
- Changes to Fonts
  - Renamed all attributes and elements that used to use camelCaseNotation to use lowercase-separated-by-hyphens notation so that names match the corresponding CSS properties, per Last Call comments from the CSS, FP and I18N groups.
  - Renamed full-font-name to font-face-name to match CSS2. (The previous name was chosen by error.)
  - Replaced the 'kern' element with 'hkern' and 'vkern' elements to support vertical kerning, per feedback comments from the I18N and XSL working groups.
  - The value of the 'unicode' attribute on the 'glyph' element is now just a regular old Unicode character, possibly expressed as a character reference, per feedback comments from the I18N and XSL working groups. Modified the descriptions of the 'hkern' and 'vkern' elements' u1 and u2 attributes, accordingly. In the process, cleaned up the wording for the u1, g1, u2 and g2 attributes and also the introductory wording about the kerning elements.
  - Added a note that Arabic glyph widths are required to be positive, per feedback comments from the I18N and XSL working groups.

- Added required attributes text-top, hanging, ideographic and text-bottom to the 'font' element, per feedback comments from the I18N and XSL working groups.
- Replaced the term "standard" with "isolated" for attribute 'arabic', per feedback comments from the I18N interest group.
- Modified the rule for prioritizing which 'glyph' is chosen to match current font practice, per feedback comments from the I18N and XSL working groups. Now, the first 'glyph' in lexical order which matches the sequence of characters to be rendered gets chosen.
- Included a recommendation that the glyphName attribute be unique across a given SVG font.
- Changed the glyphName attribute such that a single string rather than a list of strings is provided to better match the needs of the newly revised kerning elements.
- Removed attribute 'bbox' per font discussion in the SVG working group with representation from the Unicode Corsortium and Apple's font group. The conclusion was that it was better for the viewing to calculate the bbox from the available graphical information. Changed horiz-adv-x to be #REQUIRED on the 'font' element, since removal of bbox removed its fallback value. Changed the default for vert-origin-x to be half of horiz-adv-x, since removal of bbox removed its fallback value. Changed the default for vert-adv-y to be the sum of attributes ascent and descent, as removal of bbox removed its fallback value.

- Changes to Extensibility
  - Updated the example. References Dublin Core 1.1 instead of 1.0.
- Changes to Extensibility
  - Moved 'switch', 'system-required' and the new 'system-language' to Document Structure.
- Changes to SVG DTD
  - Various changes to correspond to changes described above.
  - Added attribute system-language, from SMIL 1.0, for all elements that have attribute system-required to provide the mechanism for multiple localized text strings for the same document, as noted in previous draft specifications.
  - Added entities %descTitle and %descTitleDefs to remove the restriction that child elements 'desc', 'title' and 'defs' had to appear in a particular order. Now, at most one can appear, and any order is allowed.
  - Scattered null entities xxxExt (e.g., svgExt, gExt, pathExt) throughout the DTD to allow extensions to the SVG language via the internal DTD subset.
  - Removed the #FIXED attribute to xlink:show to allow for "new" vs. "replace".
  - Changed 'parsed' to 'embed' for XLink.
  - Added 'target' attribute to 'a' element.
  - Removed #FIXED from the specification for the 'type' attribute on the 'style' element. (The #FIXED was just an editorial error.)
  - Various text-related and font-related changes, described above under "Text" and "Fonts".
  - Removed graphic element events and system-required from 'desc' as it must have been an error to have these attributes in the first place.
  - Add a 'metadata' child element to the 'svg' element and defined the 'metadata' element. (This element was defined in the spec but left out of the DTD inadvertantly.)

- Added the 'view' element, which by oversight had been left out of the previous DTD. Modified 'defs' to allow 'view' as a child element.
- Added standard attributes id, lang, class, style, etc. to <textPath>. (Previous error of omission.)
- Make an entity for all standard XLink attributes other than xlink:href.
- Lots of changes to which elements have which animation elements as children to match the tables in the animation chapter that show which Elements, attributes and properties that can be animated

- Changes to SVG DOM
  - Major editorial modifications, including a good deal of reorganization. Detailed descriptions of relationship to DOM2, including CSS OM and DOM2 Events.
  - Moved detailed interface definitions into chapters which describe corresponding elements and features.
  - Defined SVG's available feature strings for "hasFeature" method call and "system-required" attribute.
  - Renamed suspend_redraw, unsuspend_redraw and unsuspend_redraw_all to suspendRedraw, unsuspendRedraw and unsuspendRedrawAll per suggestion from the DOM working group.
- Changes to Implementation Requirements
  - Changed the title of the appendix to "Implementation Requirements" instead of "Implementation Notes" to emphasize the fact that this appendix is normative.
  - Added a section Version control which says that SVG user agents should only render documents which have a reference to the SVG DTD and for which the reference points to a DTD which the user agent supports.
  - Added a bullet to the implementation notes on path data on error handling stating that the rendering should continue up to and including the last correctly defined path segment, even if it is in the middle of a compound path command such as "L 100 200 100 400".
  - Modified the wording in Forward and undefined references to say that invalid references are errors and handled by standard user agent error processing behavior, per decision by the working group to have consistent error handling.
  - Added a section Clamping values which are restricted to a particular range to indicate that out-of-range values get clamped at the latest possible moment.
  - Added a section Elliptical arc implementation notes which provides the details on how to implement elliptical arcs in SVG path data.
  - Under Text selection implementation notes, added a paragraph that talks about user agents providing an ability to select text strings which might have an associated event handler or link.
  - Modified the description of the text selection algorithm for user agents to be consistent with the two-stage text layout processing model (i.e., first re-order characters into visual rendering order, and then position the characters, do ligatures, do kerning, etc.).
- Changes to Conformance Criteria
  - Now there are four types of Conforming SVG Viewers: Conforming Static SVG Viewer, Conforming Dynamic SVG Viewer, Conforming High-Quality Static SVG Viewer,

Conforming High-Quality Dynamic SVG Viewer, per feedback from people within in SVG working group.

- Reworded the reference to accessibility authoring guidelines, per Last Call feedback from WAI.
- Added a conformance requirement for viewers that encourages conformance to the WAI user agent guidelines, per Last Call feedback from WAI.
- Expanded the bullet point about a Conforming SVG Document needing to be validatable after removing non-SVG elements and attributes.
- Added a comment about a conforming SVG interpreter requiring an XML Parser that supports XML 1.0 and XML Namespaces.
- Added a bullet stating that High-Quality SVG Viewers need to support double-precision floating point operations on coordinate system transformation numerical operations
- For Conforming SVG Viewers, added bullets stating that rendering must be accurate to within one device pixel and sRGB colors must be accurate to within one color value.

- Changes to Accessibility Support
  - General editorial cleanup.
  - Made additions and changes to Accessibility and SVG about the potential for harming the accessibility of information if representing information in pure visual, final-form SVG, per Last Call feedback comments from Daniel Dardailler of the Web Accessibility Initiative (WAI).
  - Listed the aural style sheet properties from CSS2 as the set of properties which will be available in user agents that support aural properties.
  - Transformed the various notes and comments about accessibility from the Last Call spec into a consolidated bulleted list of SVG-specific accessibility guidelines.
- Changes to References
  - Shifted some references from being informative to normative and vice versa as part of general editorial cleanup.
  - Added a normative reference to [RFC2119] for definitions of some of the terms used in this specification.
  - Added a normative reference to "SMIL Animation", which SVG will reference normatively.
  - Added an informative reference to [XSL].
  - Added many references into specific sections of referenced specs, such as references to various property definitions from the CSS2, DOM2 and SMIL-Animation spec.
  - Added a normative reference to the PNG specification.
  - Modified ACCESS to refer to ACCESS-AUTHOR.
  - Added ACCESS-USERAGENTS.
  - Added a normative reference to RFC1766 for language identification.

Changes with the 12-August-1999 (Last Call) SVG Draft Specification

- Global/miscellaneous changes
  - Created a new chapter on Fonts which contains the contents of the old appendix "Implementation and Performance Notes for Fonts" plus the specification for SVG fonts (i.e., fonts defined in SVG).
  - Created a new chapter exclusively on Linking.
  - Changed the order of some of the chapters (Interactivity, Scripting, etc) as a result of creating a chapter on Linking and moving various sections to different chapters.
- Changes to Document Structure
  - Included additional details on property inheritance with the 'use' element.
  - Changed one of the alternative syntaxes for URI reference from #id(foo) to #xptr(id(foo)) to be compatible with latest draft of XPointer. Inserted language stating that the only part of XPointer that SVG 1.0 user agents are required to support are the #elementID and #xptr(id(elementID)) syntaxes.
- Changes to Styling and CSS
  - Modified the discussion of the 'display' property so that svg { display: block } and svg * { display: svg}. Removed all of the detailed display values, such as svg-g, svg-rect, etc.
  - Added quick documentation of the 'overflow' and 'clip' properties.
  - Added a quick write-up on SVG's default CSS style sheet.
- Changes to Painting: Filling, Stroking and Marker Symbols
  - Renamed chapter to "Painting: Filling, Stroking and Marker Symbols".
  - Removed 'fill-params' and 'stroke-params'. Instead, use private data via foreign namespaces.
- Changes to Gradients and Patterns
  - Changed 'stick' to 'pad' for spreadMethod attribute.
  - For gradient stops, color and opacity are now set by properties 'stop-color' and 'stop-opacity' rather than 'color' and 'opacity'.
- Changes to Text
  - For the 'tspan' element, modified the x,y,dx,dy attributes to accept a list of values for a compact way to provide individual kerning and tracking between glyphs. Also, added new attribute dCoordUnits="userSpace|em|fontSpace" which permits dx,dy to be provided in three different coordinate systems.
  - Added property 'text-advance' to allow for horizontal, vertical or vertical-ideographic text.
- Changes to Filters
  - Minor change to description of saturate value on feColorMatrix to indicate clearly that it can take on either a real number between 0 and 1 or a percentage value such as "50%".
  - Renamed feColor to feFlood. Changed color value for feFlood from a color attribute to 'flood-color' and 'flood-opacity' properties.

- Changes to Interactivity
    - Added an Introduction to provide an overview of the various interactivity options that are available (e.g., links, scripting event handling).
    - Added a section on Cursors which describe new element 'cursor' and new property 'cursor' which allow a built-in or custom cursor to be used when the pointing device is over a specific element.
- Changes to Linking
    - New chapter. Contains description of 'a' element and discussion of linking into an SVG document.
- Changes to Scripting
    - Added events onresize, onscroll, onerror, onabort to expose to script writers these events which are a standard part of DOM level 2.
- Changes to Animation
    - Inserted a new section 15.2.1 Introduction which describes the collaborative effort between the SYMM and SVG working groups to define SVG's animation elements.
    - Removed the "dom" option to the attributeType attribute.
    - Made all of the xlink:href attributes be #IMPLIED rather than #REQUIRED and indicated that you can only reference elements within the same SVG document.
    - Modified the wording on vtimes attribute.
    - Replaced the "repeat" attribute with "repeatCount" to track latest changes to the SYMM timing and animation drafts.
    - Removed interpColorModel attribute. SVG 1.0 will only support rgb color animations.
- Changes to Extensibility
    - Added language indicating that attributes from foreign namespaces are OK. They will be included in the DOM but otherwise ignored.
- Changes to SVG DTD
    - Fixed omission in previous DTD where the various animation elements were not children of any other elements. Now, many elements have various animation elements as optional children.
    - Changed href attribute on 'animate', 'animateMotion', 'animateTransform' and 'animateColor' from #REQUIRED to #IMPLIED to match SYMM animation formulation where an animation element can be a child of the object being animated and thus the default href is the animation element's parent.
- Changes to SVG DOM
    - Fixed error where it used to say that 'title' is a subelement to 'defs'.
- Changes to Conformance Criteria
    - Changed the title from "Conformance Requirements and Recommendations" to "Conformance Criteria" since the W3C will not be policing adherence to the specification and will not be sanctioning another body to do so either. Instead, industry and the media will have to do its own policing. The Conformance Criteria are the W3C's statements about quality and completeness of implementations. These criteria will help developers create complete implementations and will help industry and the media to judge the qualith and completeness of SVG support in industry.

- Added a note about removing foreign namespace attributes (in addition to foreign namespace elements) before attempting to validate.

Changes with the 30-July-1999 SVG Draft Specification

- Global/miscellaneous changes
  - Major editorial cleanup touching almost everything in a major push toward readying the specification for formal review by other working groups.
  - Lots of renaming of element names, attributes and identifiers to use "camel notation". For example, 'fit-box-to-viewport' is now 'fitBoxToViewport'. Exact list of changes is found in camel.sed.19990722.txt
  - Created new appendices: Implementation Notes (whose content used to be scattered about the spec) and Conformance Requirements and Recommendations (whose content used to be found in Chapter 3: Conformance Requirements and Recommendations).
  - Various consolidation and rearrangement of chapters and sections within chapters, resulting in lots of chapter and appendix renumbering.
  - Updated all href attributes to conform to latest XLink draft..
  - Moved 'desc' and 'title' elements into Document Structure.
  - Consolidated Private Data, Extensibility and Foreign Object sections into a single chapter Extensibility.
  - Removed SVG Requirements and Change History from document.
  - Renumbered appendices.
- Changes to Introduction to SVG
  - Updated the section describing SVG's relationship to other web standards.
  - Included a list of standard terms in Definitions.
- Changes to Document Structure
  - Near total rewrite of the section on references and the 'defs' element. (See References and the 'defs' element.) Included a more precise definition of the exact formats allowed in a reference (i.e., #foo and #id(foo)). (Nearly everything is described more precisely.)
- Changes to CSS and Styling
  - Reformulated the chapter to represented all of the introductory and high-level discussion of how CSS relates to SVG.
  - Moved the main discussion of the 'script' element from struct.html into this chapter.
  - Added stub sections to discuss the style and class attributes.
- Changes to Coordinate Systems, Transformations and Units
  - Renamed Implementation Notes to Processing rules for CSS units and percentages
  - General cleanup of the discussion in Processing rules for CSS units and percentages. Included an explicit description of what to do if percentages are used for coordinate

values. Reformulated the discussion of x and y coordinates exprssed in viewport-relative units because the previous methods could result in attempting to find the intersection of parallel lines.

- Changes to SVG Rendering Model
  - Lots of cleanup to remove ambiguities and to fix omissions. Included discussion of: marker symbols, the order of fill vs. stroke vs markers, distinction of shapes vs. text vs. raster images, centering of the paint on the stroke, three different types of built-in paint that can be applied to fill and stroke operations (i.e., solid color, gradients and patterns), unambiguously defined the order in which operations apply (e.g., filters before clipping, masking and object opacity). Incorporated standard terminology and added several hyperlinks.
  - Fixed bug in image-rendering which used to say that the property applied to text elements. (Then moved the rendering properties into other chapters.)
- Changes to Clipping, Masking and Compositing
  - Changed the range on the 'opacity' property from 0-255 to 0-1 to match common usage and to make consistent with properties 'fill-opacity' and 'stroke-opacity'.
- Transformed the old chapter "CSS Properties, XML Attributes, Cascading and Inheritance" into Styling and CSS. Specific changes:
  - First crack at defining explicitly which CSS features would be supported.
  - Moved 'style' element and class/style attributes into this chapter.
- Changes to Filling, Stroking and Paint Servers
  - Reorganization. Moved markers into this chapter. Moved colors, gradients and patterns into Gradients and Patterns.
  - Fixed initial values for fill-opacity and stroke-opacity from "evenodd" (obviously a bug) to "100%".
  - Removed sentence saying a null value for stroke-dasharray was equivalent to 'none'. (Instead, for all properties, a null value is invalid and shall result in the property setting getting ignored.) Added a sentence indicating that if an odd number of values is provided, then the list of values is repeated to yield an even number of values (i.e., twice the values).
  - Removed comments about paint server extensibility
- New chapter Gradients and Patterns
  - From reorganization. Contains discussions of colors, gradients and patterns.
  - Under Properties for specifying color profiles, replaced property 'icc-profile' with latest proposals from CSS working group: 'color-profile' and 'rendering-intent'.
- Changes to Paths
  - Removed the 1023 character limitation on path data and eliminated the 'data' child element to the 'path' element. Was going to add newline and tab characters to the BNF for path data, but discovered they were already there. Added a guideline recommending that SVG generators insert newline characters into long path data strings to keep line lengths less than 255 characters.
  - Fixed error in In Path data where the previous spec showed the parameters to moveto, lineto, etc. as (x y)*, which means zero or more. It is now (x y)+, which means one or more.
  - In the table for Close path command, we now show both uppercase and lowercase "Z" to

match the BNF.

- Changes to Basic Shapes
  - Removed the 1023 character limitation on vertices for polylines and polygons. Added a guideline recommending that SVG generators insert newline characters into long path data strings to keep line lengths less than 255 characters.
  - Fixed examples to use width/height attributes instead of width/height properties.
- Changes to Text
  - Explicitly listed which CSS2 properties SVG supports.
  - Removed text-direction from text-on-a-path section awaiting decisions on vertical text support. (The old formulation was clearly wrong.)
  - In text-on-a-path section, renamed text-transform to textPath-transform because CSS already has a property named 'text-transform'.
- Added a new chapter on Scripting
  - Defined a new contentScriptType attribute on the 'svg' element to allow specification of a default scripting language.
- Changes to Filters
  - Added a new section Accessing the background image which describes property enable-background, which can be used to enable the ability to access the currently accumulated background image on the current canvas. Possible values are 'accumulate' and 'new [(x y width height)]'.
  - Cleanup of write-up on feBlend. Simplified the equation for computing result opacity and expressed all formulas using premultiplied colors.
- Changes to Animation
  - Included the declarative animation syntax that has been developed in close collaboration with the SYMM working group.
- Changes to DTD
  - Major cleanup. Changed names, conventions and comments throughout.
  - Added <desc> and <title> child elements to basic shapes, <path>, <text>, <use> and <image>.
  - Removed 'data' element as a child to <path>.
  - Removed x,y,width,height attributes from the 'symbol' element. (Assumed to have been a mistake that it had these attributes.).
  - Removed the transform attribute from the 'svg' element to make it parallel with 'symbol' element. (Assumed to have been a mistake that it had a transform attribute.).
- Changes to References
  - Updated the reference to the definition of URIs from the proposed draft dated 1997 (to which is what the HTML4 and CSS2 documents point) to the to an updated document dated 1998. The updated document includes a discussion of fragment identifiers, which are used throughout SVG.
- Changes to Accessibility Support
  - Included a reference to the latest draft of SVG authoring guidelines for accessibility.
  - Included a parenthetical comment about the WG's current investigation about providing for vocalization of tooltips along with an authoring guideline so that SVG generators

automatically convert object names (e.g., layer names) to 'title' or 'tooltip' elements.

- New appendix on Internationalization Support
  - Discussion of XML and Unicode support.
  - Discussion of W3C Character Model and altglyph.
  - Describe vertical text as an open issue.

Changes with the 06-July-1999 SVG Draft Specification

- Changes to Conformance Requirements and Recommendations:
  - In Conforming SVG Viewers, dropped GIF from the list of required formats. Now, only JPEG and PNG are listed.
  - In Forward and undefined references, indicated that forward references are disallowed and included a link to the description of the 'defs' element.
- Changes to Document Structure:
  - Modified the description of the 'defs' element to discuss how all referenced elements must be direct children of a 'defs' element.
  - Modified the description of the 'use' element to indicate that 'use' can only refer to elements within an SVG file (not entire files).
  - Added a section on the 'image' element. The 'image' element is very comparable to 'use' except that it can only refer to whole file (not elements within a file).
- Changes to Rendering Model:
  - Moved the recently modified/renamed properties shape-rendering, text-rendering and image-rendering into this chapter. (There used to be properties 'stroke-antialiasing' and 'text-antialiasing'.)
- Changes to Clipping, Masking and Compositing:
  - For Clipping paths, reformulated how clipping paths are specified. Now, there is a 'clipPath' element whose children can include 'path' elements, 'text' elements and other vector graphic shapes such as 'circle'. 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. Also, fixed a bug in the spec by replacing the 'inherit' value on 'clip-path' with a 'none' value and fixed the spec to say that 'clip-path' does *not* inherit the 'clip-path' property from its parent.
  - For Masking, reformulated how clipping paths are specified. Now, there is a 'mask' element whose children can include any graphical object. The 'mask' element can have attributes maskUnits, x, y, width and height to indicate a sub-region of the canvas for the masking operation. These changes obsolete the following old properties: 'mask-method', 'mask-width', 'mask-height', 'mask-bbox'.
- Changes to Filling, Stroking and Paint Servers:
  - Renamed stroke-antialiasing to shape-rendering, with possible values of default, crispEdges, optimizeSpeed and geometricPrecision. The revised property is now just a

hint to the implementation. Moved to Rendering chapter.

    - Revised the wording on gradient stops to indicate that out-of-order gradient stops shall be resolved by adjusting offset values until the offset values become valid. (Previously, the spec said that gradient stops would be sorted.)

- Changes to Paths:
    - Removed the old elliptical arc commands A|a and B|b and inserted a new elliptical arc command called A|a, which has a different set of parameters than the previous two formulations. The new arc command matches the formulation of the other path data commands in that it starts with the current point and ends at an explicit (x,y) value.
- Changes to Other Vector Graphic Shapes:
    - In the sentence, "Mathematically, these shape elements are equivalent to the cubic bezier path objects that would construct the same shape", removed the words "cubic bezier".
- Changes to Text:
    - Replaced the old 'textflow', 'textblock', 'text' and 'textsrc' with the new 'text' and 'tspan', which is a subelement to 'text' and has optional attributes x=, y=, dx=, dy=, style= and href= (which allows it to take the place of 'textsrc'). The only lost functionality from this simplification is the ability to select text across discontiguous blocks of text elements.
    - Made 'textPath' a container element which can contain 'tspan' elements or character data. This reformulation was necessary given the changes in the previous bullet.
    - Renamed text-antialiasing to text-rendering, with possible values of default, optimizeLegibility, optimizeSpeed and geometricPrecision. The revised property is now just a hint to the implementation. Moved to Rendering chapter.
- Changes to Images:
    - Added new property image-rendering, with possible values of default, optimizeSpeed and optimizeQuality. The new property is just a hint to the implementation. The new property is documented in the Rendering chapter.
- Changes to Filter Effects:
    - Removed vector effects, including VEAdjustGraphics and VEPathTurbulence -- the working group decided that we hadn't found a critical mass of vector graphics effects functionality sufficient to warrant the additional complexity
    - Modified the names of all of the filter effects processing nodes to have the prefix "fe". The prefix is meant to prevent name clashes (e.g., 'feImage' won't clash with 'image').
    - Removed the section on parameter substitution -- the WG didn't see why filter effects deserved macro expansion over other features.
- Changes to Animation chapter to indicate that SVG will include declarative animation. (Syntax still under development.)
- One line change in the SVG DOM chapter to change getStyle() to style property, per feedback from the DOM working group.
- Minor changes to the example in the Metadata chapter to fix incorrect references to Dublin Core elements.
- Changes to DTD
    - Changes to DTD to reflect all of the changes described earlier in this section.
    - Flattened some double-indirect entity referencing into only single-indirect referencing. Fixed bug where pattern used x,y,width,height twice.

- Changed rx,ry on 'rect' to be #IMPLIED so that if one of them is missing the other one will be assigned the same value (for circular fillets).

Changes with the 25-June-1999 SVG Draft Specification

- General editorial activities:
  - Modified the titles and content of chapters 1 and 2. Chapter 1 is now a Introduction to SVG and chapter 2 is now SVG Concepts.
  - Included a first pass of information about conformance requirements, including a discussion of what makes a conforming document, generator, interpreter and viewer.
  - Included updated wording on the Rendering Model.
  - Reorganized the appendices. Added the beginnings of Appendix D. SVG's Document Object Model (DOM), Appendix E. Sample SVG files, Appendix F. Accessibility Support, Appendix G. Minimizing SVG File Sizes, Appendix H. Implementation and performance notes for fonts and Appendix I. References.
  - Included an example of DOM-based animation>.
  - Removed some of the wording that indicated tentativeness about certain features as the specification of various features is firming up.
- Coordinate Systems, Transformations and Units modifications:
  - Changed the 'transform' property into the transform attribute. The **transform** attribute can now accept a list of transformations such as **transform="translate(-10,-20) scale(2) rotate(45) translate(5,10)"**. Added skewX and skewY convenience transformations. Removed the fit() options from the old transform property and created new attributes **fitBoxToViewport=** and **preserveAspectRatio**, described in new section Establishing an Initial User Coordinate System: the fitBoxToViewport attribute.
  - Added an Implementation Notes section to the chapter on Coordinate Systems, Transformations and Units.
  - Added a note to the description of the transform attribute to indicate that the transform attribute is applied before other attributes or properties are processed.
- Paths modifications:
  - The J|j commands (elliptical quadrant) have been dropped from the list of path data commands because the working group felt the J|j commands would not receive wide usage.
  - The path data commands for switching between absolute and relative coordinates in the middle of a command (the former A and r commands) have been dropped because of their high complexity relative to their limited space-saving value.
  - The various arc commands in path data have been consolidated, renamed, and then expanded. The new commands are: A|a (an arc whose sweep is described by a start angle and end angle) and B|b (an arc whose sweep is described by two vectors whose intersections with the ellipse define the start point and end points of the arc).
  - Reformulated the T/t path data commands to be consistent with the rest of the path data commands (i.e., vertices provided, control points automatically calculated as in S/s).

- Broke up the path data commands into separate tables to improve understandability.
- Modified the write-up on markers so that the 'marker' element no longer is a subelement to 'path'. 'marker' is now defined to be just like 'symbol', but with marker-specific attributes **markerUnits**, **markerWidth**, **markerHeight** and **orient**. To use a marker on a given 'path' or vector graphic shape, we have new properties **'marker-start'**, **'marker-end'**, **'marker-mid'** and **'marker'**. See Markers.
- Indicated that each **d=** attribute in a 'path' element is restricted to 1023 characters. See Path Data.
- Added an Implementation Notes section to the document that describes various details about expected processing and rendering behavior when drawing paths.
- Added The grammar for path data, a BNF for path data.

- Filling, Stroking and Paint Servers modifications:
  - Included a note under 'fill' property that indicates that open paths and polylines still can be filled.
  - Provided a more detailed write-up on patterns to make the 'pattern' element consistent in various ways with 'symbol', 'marker', 'linearGradient' and 'radialGradient'.
  - Modified gradients in various ways. Replaced attribute target-type with gradientUnits. Replaced 'linearGradient' attributes vector-start-x, vector-start-y, vector-length, vector-angle with x1, y1, x2, y2. Replaced 'radialGradient' attributes outermost-origin-x, outermost-origin-y, outermost-radius, innermost-x, innermost-y with cx, cy, r, fx, fy. Removed attributes target-left, target-top, target-right, target-bottom, which were deemed superfluous. Renamed attribute matrix to gradientTransform. Added gradientTransform back to linear gradients (they were in an earlier draft). Renamed 'gradientstop' to 'stop' to save space since the working group decided it didn't want to offer non-linear gradient ramps. Removed attribute color from 'stop' and included new paragraphs indicating that color and opacity are set via the 'color' and 'opacity' properties.
  - Added a value of **none** to property 'stroke-dasharray'.
- Text modifications:
  - Broke the 'textflow' element into two elements 'textblock' and 'textflow' to greatly simplify the feature, to remove the need to maintain consistent doubly linked lists, and to remove the possibility of cyclic references. Removed 'tf' and renamed 't' to 'tref'
  - Renamed the 'src' subelement to 'text' to 'textsrc' for more consistency in nomenclature and to avoid use of such a generic element name for such a specific purpose.
- General/Miscellaneous:
  - Added a syntax and various processing details for Filter Effects
  - Altered the description of the 'symbol' element to reflect the changes in transform-related attributes and properties.
  - In the chapter on Other Vector Graphic Shapes, changed the attributes on 'ellipse' from major/minor to rx/ry for consistency with other parts of the spec, removed the angle attribute on ellipse, reformulated polygon to be exactly line polyline except that it automatically closes, changed "verts" to "points", and added rounding radii rx and ry to rectangle. Also, included a note about the 1023 character limit on the "points" attribute for 'polyline' and 'polygon'.
  - Removed property 'z-index'. The working group decided that a z-index effect can be achieved either by having CSS manage multiple SVG drawings or by rearranging graphical elements via the DOM. A z-index option would complicate implementation and

streaming for little gain.

- Add a chapter on Metadata, with an initial description of how metadata would work with SVG.
- Removed the 'private' element after concluding it is unnecessary given XML namespaces and the new W3C approach to validating namespaces. As a consequence, modified the write-up under Private Data.
- Updated the descriptions under Embedding Foreign Object Types to reflect increased certainty about the direction SVG is headed in this area.
- Added a General Implementation Notes section to the chapter on Conformance Requirements and Recommendations which discusses implementation issues that apply across the entire SVG language. In particular, added sections Forward and Undefined References (which explains implementation rules involving references that aren't valid at initial processing time) and Referenced objects are "pinned" to their own coordinate systems.
- Changed all occurrences of "SVG processor" to "SVG user agent".
- Fixed all incorrect references to 'description' and replaced them with 'desc'.
- Renamed attribute 'nodeId' to 'result' due to feedback that having a name with the term 'id' in it that wasn't an ID was potentially confusing.

- Summary of changes to the DTD:
  - Gave the 'a' element have the same content model as the 'g' element.
  - Add transform attribute to most graphic objects.
  - Added attributes fitBoxToViewport and preserveAspectRatio to 'svg' and 'symbol' elements
  - Added attributes x and y to the 'svg' element.
  - For symbol_descriptor_attributes, renamed attributes x-min, y-min, x-max, y-max to x, y, width, height, respectively.
  - Modified the 'marker' element to reflect the revised formulation for markers.
  - Added a 'pattern' element which reflects the modified write-up on patterns. (The 'pattern' element was missing from the previous DTD.).
  - Modified the definitions of 'linearGradient' and 'radialGradient' to reflect the modified write-up on gradients.
  - Renamed 'gradientstop' to 'stop'.
  - Removed attribute color from 'stop'.
  - Changed the attributes on 'ellipse' from major/minor to rx/ry for consistency with other parts of the spec, removed the angle attribute on ellipse, reformulated polygon to be exactly line polyline except that it automatically closes, changed "verts" to "points", and added rounding radii rx and ry to rectangle.
  - Removed the 'private' element after concluding it is unnecessary given XML namespaces and the new W3C approach to validating namespaces.
  - Added xml:space to every element that might have character data content somewhere inside of it. This will allow content developers to control whether white space is preserved on 'text' elements.
  - Text-related: renamed 'src' to 'textsrc' for more consistency in nomenclature and to avoid use of such a generic element name for such a specific purpose. Because of modifications

in the area of defining textflows, added 'textblock', renamed 't' to 'tref' and changed 'textflow' so that if can only contain 'tref' subelements.

- Added a syntax for Filter Effects
- Modified 'foreignObject' such that it can only be the child of a 'switch' element.
- Added an href attribute to the 'script' element. (Oversight that it wasn't there before.)
- General clean-up in the area of anything using attributes x, y, width or height. Defined standard entities xy_attributes, bbox_attributes_optional and bbox_attributes_wh_required. In particular, the following elements now require width and height attributes: 'image', 'rect', 'foreignObject', 'pattern'.

Changes with the 12-April-1999 SVG Draft Specification

- Included a DTD in Appendix C.
- There is now an 'svg' element which is the root for all stand-alone SVG documents and for any SVG fragments that are embedded inline within a parent XML grammar. (See SVG Document Structure>.)
- Added initial descriptions of how text-on-a-path and SVG-along-a-path might work. (See Text on a Path.)
- Added 'symbol' and 'marker' elements to provide packaging for the following:
  - Necessary additional attributes on template objects
  - A clean way of defining standard drawing symbol libraries
  - The definition of a graphic to use as a custom glyph within a 'text' element (e.g., generalize "text-on-a-path" to "SVG-on-a-path")
  - Necessary additional attributes for pattern definitions (for pattern fill)
  - Definition of a sprite for an animation
  - Marker symbols
  - Arrowheads

  Also added a new optional 'data' subelement to the 'path' element to provide the necessary hook to provide for custom arrowheads.
- Many changes to Coordinate Systems, Transformations and Units to make the section more complete and more readable. The specific changes to this chapter include:
  - Relatively minor changes in terminology to better match the terminology used in the CSS2 specification. For example, the definitions of the terms *canvas* and *viewport* were modified to be as close as possible to the corresponding definitions in the CSS2 specification.
  - The initial coordinate system is now based on the parent document's notion of pixels rather than points.
  - When embedded inline within a parent XML grammar, the outermost 'svg' element in an SVG document acts like a block-level formatting object in the CSS layout model and thus supports CSS positioning properties such as **'left'** and **'width'** and the CSS properties

'clip' and **'overflow'**.

- Nested 'svg' elements are the mechanism for recursively including nested SVG drawings, but also provide the one and only means of establishing a new viewport and thus changing the meaning of the various CSS unit specifiers such as px, pt, cm and % (percentages). Nested 'svg' elements support the same CSS positioning properties as an outermost 'svg' element,

- Removed 'pieslice', which was considered to be of lesser general utility than the other predefined vector graphic shapes, and added 'line', which allows a one-segment line to be drawn. See Other Vector Graphic Shapes.
- Replaced the 'althtml' element with a description for how to use the 'switch' (or equivalent) elements in XML grammars or the 'object' element in HTML 4.0 as the recommended way to provide for alternate representations in the event the user agent cannot process an SVG drawing. (See Backwards Compatibility.)
- Removed the comment in the discussion under 'description' and 'title' which said that the given text string could be specified as an attribute. The text string now can only be supplied as character data. (See The 'description' and 'title' elements.
- Changed the wording about text strings to say that the current point is advanced by the metrics of the glyph(s) used rather than the character used. (See text positioning.)
- Added some details to the description of the 'textflow' element to indicate that 'text' elements can be directly embedded within 'textflow' and that the current text position is remembered within a 'textflow' from one 'text' element to the next 'text' element. (See Text Flows.)
- Added a new property **'text-antialiasing'** to provide a hint to the user agent about whether or not text shall be antialiased. The lack of such a property was an inadvertant omission from previous versions of the spec and was called for in the SVG Requirements document.
- Removed the 'matrix' property from linear gradients because it was unnecessary (overspecification) and the 'spreadMethod' property from radial gradients because it was difficult to specify and implement, it didn't match current common usage and is of little apparent utility. (See Gradients.)
- Included a new section 2.1 with a brief discussion about the "image/svg" MIME type. Subsequent sections in chapter 2 have been renumbered accordingly. (See SVG MIME Type.)
- Added another bullet to the Accessibility section to indicate that SVG's zooming feature aids those with partial visual impairment. (See Accessibility.)
- Elaborated to a small level on how Embedded Foreign Object Types might work to reflect progess within the working group on the issue.
- Changed altglyph from a subelement to 'text' to a CSS property in response to discussion on the W3C Character Model. See Alternate Glyphs.
- In the discussion about the 'use' element, made clear that template objects could come from either the same document or an external document.
- Minor changes to description under Event Handling to indicate that any element can have an onload or onunload event handler to provide additional control via scripting as parts of the drawing download progressively.

Changes with the 05Feb1999 SVG Draft Specification

This was the first public working draft.

Property Index

This will contain the property index

Index

This will contain the index