



Scalable Vector Graphics (SVG) 1.0 Specification

W3C Working Draft *12 August 1999*

This version: <http://www.w3.org/1999/08/WD-SVG-19990812/>

Latest version: <http://www.w3.org/TR/SVG>

Previous version: <http://www.w3.org/1999/07/30/WD-SVG-19990730/>

Editor: Jon Ferraiolo <jferrai@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>

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 <jferrai@adobe.com>

Scott Furman, Netscape Communications Corporation <fur@netscape.com>

Peter Graffagnino, Apple <pgraff@apple.com>

Lofton Henderson, Inso Corporation <lofton@cgm.com>

Alan Hester, Xerox Corporation <Alan.Hester@usa.xerox.com>

Bob Hopgood, RAL (CCLRC) <frah@inf.rl.ac.uk>

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 <kmclusk@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 incorporates suggestions received in review comments since the first public working draft of SVG (05 February 1999) and further deliberations of the W3C SVG working group. Significant changes since the first public working draft are listed in [Appendix J: Change History](#). With the publication of this draft, the SVG specification enters "last call". The last call period will end on September 9, 1999. Last call comments should be sent to svg-comments@w3.org. Publication as a "last call" working draft does not imply endorsement by the W3C membership.

This is a draft document and may be updated, replaced or obsoleted by other documents at any time. 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](#) may 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/1999/08/WD-SVG-19990812/index.html>

a zip file:

<http://www.w3.org/1999/08/WD-SVG-19990812/WD-SVG-19990812.zip>.

and a PDF file:

<http://www.w3.org/1999/08/WD-SVG-19990812/WD-SVG-19990812.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

- [1 Introduction to SVG](#)
- [2 SVG Concepts](#)
- [3 SVG Document Structure](#)
- [4 SVG Rendering Model](#)
- [5 Clipping, Masking and Compositing](#)
- [6 Styling and CSS](#)
- [7 Coordinate Systems, Transformations and Units](#)
- [8 Painting \(Filling and Stroking\)](#)
- [9 Built-in Types of Paint](#)
- [10 Paths](#)
- [11 Basic Shapes](#)
- [12 Text](#)
- [13 Fonts](#)
- [14 Filter Effects](#)
- [15 Linking](#)
- [16 Scripting](#)
- [17 Interactivity](#)
- [18 Animation](#)
- [19 Metadata](#)
- [20 Backwards Compatibility](#)
- [21 Extensibility](#)
- [Appendix A: Document Type Definition](#)

- [Appendix B: SVG's Document Object Model \(DOM\)](#)
- [Appendix C: Implementation Notes](#)
- [Appendix D: Conformance Criteria](#)
- [Appendix E: Accessibility Support](#)
- [Appendix F: Internationalization Support](#)
- [Appendix G: Sample SVG files](#)
- [Appendix H: Minimizing SVG File Sizes](#)
- [Appendix I: References](#)
- [Appendix J: Change History](#)

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

- Appendix L: Element, attribute and property index
- Appendix M: Index

Full Table of Contents

- [1 Introduction to SVG](#)
 - [1.1 About SVG](#)
 - [1.2 SVG MIME Type](#)
 - [1.3 Compatibility with Other Standards Efforts](#)
 - [1.4 Terminology](#)
- [2 SVG Concepts](#)
- [3 SVG Document Structure](#)
 - [3.1 Defining an SVG document: the <svg> element](#)
 - [3.1.1 Overview](#)
 - [3.1.2 The <svg> element](#)
 - [3.2 Grouping and Naming Collections of Drawing Elements: the <g> element](#)
 - [3.2.1 Overview](#)
 - [3.2.2 The <g> element](#)
 - [3.3 References and the <defs> element](#)
 - [3.3.1 Overview](#)
 - [3.3.2 URI reference attributes](#)
 - [3.3.3 The <defs> element](#)
 - [3.4 The <desc> and <title> elements](#)
 - [3.5 The <symbol> element](#)

- [3.6 The <use> element](#)
- [3.7 The <image> element](#)
- [4 SVG Rendering Model](#)
 - [4.1 Introduction](#)
 - [4.2 The painters model](#)
 - [4.3 Rendering Order](#)
 - [4.4 Grouping](#)
 - [4.5 Types of graphics elements](#)
 - [4.5.1 Painting shapes and text](#)
 - [4.5.2 Painting raster images](#)
 - [4.6 Filtering painted regions](#)
 - [4.7 Clipping, masking and object opacity](#)
 - [4.8 Parent Compositing](#)
- [5 Clipping, Masking and Compositing](#)
 - [5.1 Introduction](#)
 - [5.2 Simple Alpha Blending/Compositing](#)
 - [5.3 Clipping paths](#)
 - [5.4 Masking](#)
 - [5.5 Object And Group Opacity: the 'opacity' Property](#)
- [6 Styling and CSS](#)
 - [6.1 SVG's Use of Cascading Style Sheets](#)
 - [6.2 Referencing External Style Sheets](#)
 - [6.3 The <style> element](#)
 - [6.4 The class attribute](#)
 - [6.5 The style attribute](#)
 - [6.6 Cascading and Inheritance of CSS Properties](#)
 - [6.7 The Scope/Range of CSS Styles](#)
 - [6.8 The 'display' property](#)
 - [6.9 'overflow' and 'clip' properties](#)
 - [6.10 Default styles sheet for SVG](#)
- [7 Coordinate Systems, Transformations and Units](#)
 - [7.1 Introduction](#)
 - [7.2 Establishing the initial viewport](#)
 - [7.3 Establishing A New User Coordinate System: Transformations](#)

- [7.4 Establishing an Initial User Coordinate System: the viewBox attribute](#)
- [7.5 Modifying the User Coordinate System: the transform attribute](#)
- [7.6 Establishing a New Viewport: the <svg> element within an SVG document](#)
- [7.7 Units](#)
- [7.8 Redefining the meaning of CSS unit specifiers](#)
- [7.9 Processing rules for CSS units and percentages](#)
- [7.10 Further Examples](#)
- [8 Painting \(Filling and Stroking\)](#)
 - [8.1 Introduction](#)
 - [8.2 Fill Properties](#)
 - [8.3 Stroke Properties](#)
 - [8.4 Markers](#)
 - [8.4.1 Introduction](#)
 - [8.4.2 The <marker> element](#)
 - [8.4.3 Marker properties](#)
 - [8.4.4 Details on How Markers are Rendered](#)
 - [8.5 Rendering Properties](#)
 - [8.6 Inheritance of Painting Properties](#)
- [9 Built-in Types of Paint](#)
 - [9.1 Introduction](#)
 - [9.2 Color](#)
 - [9.2.1 Introduction](#)
 - [9.2.2 Properties for specifying color profiles](#)
 - [9.3 Gradients](#)
 - [9.3.1 Introduction](#)
 - [9.3.2 Linear Gradients](#)
 - [9.3.3 Radial Gradients](#)
 - [9.3.4 Gradient Stops](#)
 - [9.4 Patterns](#)
- [10 Paths](#)
 - [10.1 Introduction](#)
 - [10.2 The <path> element](#)
 - [10.3 Path Data](#)
 - [10.3.1 General information about path data](#)

- [10.3.2 The "moveto" commands](#)
- [10.3.3 The "closepath" command](#)
- [10.3.4 The "lineto" commands](#)
- [10.3.5 The curve commands](#)
- [10.3.6 The grammar for path data](#)
- [11 Basic Shapes](#)
 - [11.1 Introduction](#)
 - [11.2 The <rect> element](#)
 - [11.3 The <circle> element](#)
 - [11.4 The <ellipse> element](#)
 - [11.5 The <line> element](#)
 - [11.6 The <polyline> element](#)
 - [11.7 The <polygon> element](#)
- [12 Text](#)
 - [12.1 Introduction](#)
 - [12.2 The <text> element](#)
 - [12.3 White space handling](#)
 - [12.4 Text selection](#)
 - [12.5 Text and font properties](#)
 - [12.5.1 Introduction](#)
 - [12.5.2 CSS font properties used by SVG](#)
 - [12.5.3 CSS text properties used by SVG](#)
 - [12.6 Ligatures and alternate glyphs](#)
 - [12.7 Text on a path](#)
 - [12.7.1 Vector graphics along a path](#)
- [13 Fonts](#)
 - [13.1 Introduction](#)
 - [13.2 SVG fonts](#)
 - [13.2.1 Overview of SVG fonts](#)
 - [13.2.2 The element](#)
 - [13.2.3 The <glyph> element](#)
 - [13.2.4 The <missingGlyph> element](#)
 - [13.2.5 The <kern> element](#)
- [14 Filter Effects](#)

- [14.1 Introduction](#)
- [14.2 Background](#)
- [14.3 Basic Model](#)
- [14.4 Defining and Invoking a Filter Effect](#)
- [14.5 Filter Effects Region](#)
- [14.6 Common Attributes](#)
- [14.7 Accessing the background image](#)
- [14.8 Filter Processing Nodes](#)
- [15 Interactivity](#)
 - [15.1 Introduction](#)
 - [15.2 Zoom and pan control: the allowZoomAndPan attribute on the <svg> element](#)
 - [15.3 Cursors](#)
 - [15.3.1 Introduction to cursors](#)
 - [15.3.2 The 'cursor' property](#)
 - [15.3.3 The <cursor> element](#)
- [16 Linking](#)
 - [16.1 Links out of SVG documents: the <a> element](#)
 - [16.2 Linking into SVG documents: URI fragments and SVG views](#)
 - [16.2.1 Introduction: URI fragments and SVG views](#)
 - [16.2.2 SVG fragment identifiers](#)
 - [16.2.3 Predefined views: the <view> element](#)
- [17 Scripting](#)
 - [17.1 Specifying the scripting language](#)
 - [17.1.1 Specifying the default scripting language](#)
 - [17.1.2 Local declaration of a scripting language](#)
 - [17.2 The <script> element](#)
 - [17.3 Event Handling](#)
- [18 Animation](#)
 - [18.1 Introduction](#)
 - [18.2 Animation elements](#)
 - [18.2.1 Introduction](#)
 - [18.2.2 The <animate> element](#)
 - [18.2.3 The <animateMotion> element](#)
 - [18.2.4 The <animateTransform> element](#)

- [18.2.5 The <animateColor> element](#)
 - [18.2.6 The <animateFlipbook> element](#)
 - [18.3 Animation example using the SVG DOM](#)
- [19 Metadata](#)
 - [19.1 Introduction](#)
 - [19.2 The SVG Metadata Schema](#)
 - [19.3 An Example](#)
- [20 Backwards Compatibility](#)
- [21 Extensibility](#)
 - [21.1 Foreign Namespaces and Private Data](#)
 - [21.2 Embedding Foreign Object Types](#)
 - [21.3 The system-required attribute](#)
- [Appendix A: Document Type Definition](#)
- [Appendix B: SVG's Document Object Model \(DOM\)](#)
 - [B.1 SVG DOM Overview](#)
 - [B.2 Naming Conventions](#)
 - [B.3 Objects related to SVG documents](#)
- [Appendix C: Implementation Notes](#)
 - [C.1 Introduction](#)
 - [C.2 Forward and undefined references](#)
 - [C.3 Referenced objects are "pinned" to their own coordinate systems](#)
 - [C.4 <path> element implementation notes](#)
- [Appendix D: Conformance Criteria](#)
 - [D.1 Introduction](#)
 - [D.2 Conforming SVG Documents](#)
 - [D.3 Conforming SVG Stand-Alone Files](#)
 - [D.4 Conforming SVG Included Documents](#)
 - [D.5 Conforming SVG Generators](#)
 - [D.6 Conforming SVG Interpreters](#)
 - [D.7 Conforming SVG Viewers](#)
- [Appendix E: Accessibility Support](#)
 - [E.1 Accessibility and SVG](#)
 - [E.2 SVG Accessibility Guidelines](#)
- [Appendix F: Internationalization Support](#)

- [F.1 Internationalization and SVG](#)
- [F.2 SVG Internationalization Guidelines](#)
- [Appendix G: Sample SVG files](#)
- [Appendix H: Minimizing SVG File Sizes](#)
- [Appendix I: References](#)
 - [I.1 Normative references](#)
 - [I.2 Informative references](#)
- [Appendix J: Change History](#)

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

- [Appendix L: Element, attribute and property index](#)
- [Appendix M: Index](#)

Copyright © 1999 [W3C](#) ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved.

[next](#) [contents](#) [properties](#) [index](#)



1 Introduction to SVG

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 the SVG document) 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.

1.2 SVG MIME Type

The MIME type for SVG will be "image/svg". The W3C will register this MIME type around the time which 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 supports many of the properties and approaches described in the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)]. (See [SVG's Use of Cascading Style Sheets](#).)
- External style sheets are referenced using the mechanism documented in "Associating Style Sheets with XML documents Version 1.0" [[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 is tracking and incorporating many of the facilities described in latest drafts of "Document Object Model (DOM) level 2" [[DOM2](#)], including the approach to 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 test attribute and the timing attributes that are within SVG's animation elements (see [Animation](#)). SVG's animation features were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [[SMIL1](#)], with an attempt at compatibility with SYMM work-in-progress efforts towards formulating SMIL 2.0 and general animation facilities for XML grammars such as XHTML [[XHTML10](#)].
- 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 compatibility with W3C work on web accessibility [[ACCESS](#)]. Also, see [Accessibility Support](#).

Because SVG conforms to DOM, it will be scriptable just like HTML version 4 (sometimes called DHTML). This will allow a single scripting approach to be used simultaneously for both XML documents and SVG graphics. Thus, interactive and dynamic effects will be possible on multiple XML namespaces using the same set of scripts.

1.4 Terminology

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 document canvas](#) in the chapter on [Coordinate Systems, Transformations](#)

[and Units](#).

clipping path

a combination of [<path>](#), [<text>](#) and [basic shapes](#) which serve as the outline of a (in the absence of 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 transformation matrix (CTM)

Transformation matrices define the mathematical mapping from one coordinate system into another using a 3x3 matrix using the equation $[x' \ y' \ 1] = [x \ y \ 1] * \text{matrix}$. The *current transformation matrix* (CTM) defines the mapping from the user coordinate system into the viewport coordinate system. See [Establishing A New User Coordinate System: Transformations](#)

fill

The operation of [painting](#) the interior of a [shape](#) or the interior of the character glyphs in a text string.

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](#).

number

A real number (??? provide exact BNF)

paint

A paint represents a way of putting color values onto the canvas. A paint might consist of both color values and associated alpha values which control the blending of colors against already

existing color values on the canvas. SVG supports three types of built-in paint: [color](#), [gradients](#) and [patterns](#).

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 document canvas

the [canvas](#) onto which the SVG document is rendered. See the discussion of the [SVG document canvas](#) in the chapter on [Coordinate Systems, Transformations and Units](#).

SVG document viewport

the [viewport](#) within the [SVG document canvas](#) which defines the rectangular region into which the SVG document is rendered. See the discussion of the [SVG document 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 [Establishing A New User Coordinate System: Transformations](#)

transformation matrix

Transformation matrices define the mathematical mapping from one coordinate system into another using a 3x3 matrix using the equation $[x' \ y' \ 1] = [x \ y \ 1] * \text{matrix}$. See [current transformation matrix \(CTM\)](#) and [Establishing A New User 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 [Establishing A New User 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 document 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 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 [Establishing the initial viewport](#) and [Establishing a New Viewport: the <svg> element within an SVG document](#).

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.

2 SVG Concepts

Explaining the name: SVG

SVG stands for [Scalable](#) [V](#)ector [G](#)raphics, 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 zoomed 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 `` 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 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 may 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 searchability and accessibility suffer. In response to feedback from designers, SVG includes font elements so that both text and graphical appearance are preserved.

Animation

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

3 SVG Document Structure

Contents

- [3.1 Defining an SVG document: the <svg> element](#)
 - [3.1.1 Overview](#)
 - [3.1.2 The <svg> element](#)
- [3.2 Grouping and Naming Collections of Drawing Elements: the <g> element](#)
 - [3.2.1 Overview](#)
 - [3.2.2 The <g> element](#)
- [3.3 References and the <defs> element](#)
 - [3.3.1 Overview](#)
 - [3.3.2 URI reference attributes](#)
 - [3.3.3 The <defs> element](#)
- [3.4 The <desc> and <title> elements](#)
- [3.5 The <symbol> element](#)
- [3.6 The <use> element](#)
- [3.7 The <image> element](#)

3.1 Defining an SVG document: the <svg> element

3.1.1 Overview

An SVG document is contained within an <svg> element, which is required to be the outermost element in an SVG document.

An SVG "document" can range from a single SVG [graphics element](#) such as a [<rect>](#) to a complex, deeply nested collection of [container elements](#) and [graphics elements](#). Also, an SVG document can be embedded inline as a **fragment** within a parent document (an expectedly common situation within XML Web pages) or it can stand by itself as a **self-contained** graphics file.

The following example shows a simple SVG document 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-19990812.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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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 documents. This is the mechanism by which SVG documents can be embedded within other SVG documents.

Another use for <svg> elements within the middle of SVG documents is to establish a new viewport and alter the meaning of CSS unit specifiers. See [Establishing a New Viewport: the <svg> element within an SVG document](#) and [Redefining the meaning of CSS unit specifiers](#).

3.1.2 The <svg> element

```
<!ELEMENT svg (defs?,desc?,title? ,
                (path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|switch|a)* )>
<!ATTLIST svg
  xmlns CDATA #FIXED 'http://www.w3.org/Graphics/SVG/SVG-19990812.dtd'
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  %documentEvents;
  system-required 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'
  allowZoomAndPan (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](#)].

`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](#)].

`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](#)].

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

Standard XML attribute to specify whether white space should be 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.

`x = "x-coordinate"`

(Ignored for outermost <svg> elements.) The *x-coordinate* of one corner of the rectangular region into which an embedded <svg> element should be placed. The default *x-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#).

`y = "y-coordinate"`

(Ignored for outermost <svg> elements.) The *y-coordinate* of one corner of the rectangular region into which an embedded <svg> element should be placed. The default *y-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#).

`width = "length"`

For outermost <svg> elements, the intrinsic width of the SVG document, 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 should be placed.

`height = "length"`

For outermost <svg> elements, the intrinsic height of the SVG document, 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 should be placed.

`refX = "x-coordinate"`

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

`refY = "y-coordinate"`

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

Attributes defined elsewhere:

[class](#), [style](#), [%graphicsElementEvents](#);, [%documentEvents](#);, [system-required](#), [viewBox](#), [preserveAspectRatio](#), [allowZoomAndPan](#), [contentScriptType](#).

3.2 Grouping and Naming Collections of Drawing Elements: the <g> element

3.2.1 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:


```

<!ELEMENT g (defs?,desc?,title?,
             (path|text|rect|circle|ellipse|line|polyline|polygon|
              use|image|svg|g|switch|a|
              animate|animateTransform|animateColor)*)>
<!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 CDATA #IMPLIED>

```

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

3.3 References and the <defs> element

3.3.1 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, among the facilities that allow URI references are:

- the [<use>](#) element
- the [<image>](#) element
- the 'clip-path' property
- the 'mask' property
- the 'fill' property
- the 'stroke' property
- the 'marker', 'marker-start', 'marker-mid' and 'marker-end' properties
- the [<textpath>](#) element
- the [<tspan>](#) element
- the [<feImage>](#) element
- the [<a>](#) element
- the [<animate>](#) element
- the [<animateMotion>](#) element
- the [<animateTransform>](#) element
- the [<animateColor>](#) element
- the [<animateFlipbookValue>](#) element

- the [<script>](#) 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 #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 should 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) should be treated as invalid references.
- URI references to elements that don't exist should be treated as invalid references.
- URI references to elements which are inappropriate targets for the given reference should 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>](#).

Invalid references should be treated as if no value were provided for the referencing attribute or property. For example, if there is no element with ID "BogusReference" in the current document, then fill="url(#BogusReference)" would represent an invalid reference. In cases like this, the element should be processed as if no 'fill' property were provided. In those cases where a list of property values are possible and one of the property values is an undefined reference, then the list will be processed as if the invalid reference were removed from the list.

3.3.2 URI reference attributes

```
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|parsed|replace) #FIXED 'parsed'
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](#)].

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](#)].

xlink:role = '<string>'

A generic string used to describe the function of the link's content. Refer to the "XML Linking Language (XLink)" [[XLink](#)].

xlink:title = '<string>'

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

xlink:show = 'parsed'

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

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](#)].

xlink:href = "<[URI-reference](#)>"

The location of the referenced object, expressed as a [URI-reference](#). Refer to the "XML Linking Language (XLink)" [[XLink](#)].

3.3.3 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.

```
<!ELEMENT defs (script|style|symbol|marker|clipPath|mask|
  linearGradient|radialGradient|pattern|filter|cursor|font|
  animate|animateMotion|animateTransform|animateColor|animateFlipbook|
  path|text|rect|circle|ellipse|line|polyline|polygon|
  use|image|svg|g|switch)* >
<!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 documents 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 August 1999//EN" "http://www.w3.org/Graphics/SVG/SVG-19990812.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.

3.4 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. These description strings provide information about the graphics to visually impaired users. User agents which can process SVG in most cases will ignore the description strings (except that the <title> might be used to provide a tooltip).

The following is an example. In typical operation, the SVG user agent would ignore (i.e., not display) the <desc> element and the <title> elements and would render the remaining contents of the <g> element.

If this file were processed by an older browser (i.e., a browser that doesn't have SVG support), then the browser would treat the file as HTML. All SVG elements would not be recognized and therefore ignored. The browser would render all character data (including any character data within <desc> and <title> elements) within the current text region using current text styling parameters.

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "http://www.w3.org/Graphics/SVG/SVG-19990812.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-19990812.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](#)

3.5 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 may be referenced by many different SVG documents
- 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.

```
<!ELEMENT symbol (defs?,desc?,title?,
                 (path|text|rect|circle|ellipse|line|polyline|polygon|
                  use|image|svg|g|switch|a|
                  animate|animateTransform|animateColor)*)>
<!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](#).

3.6 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 should be 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 should be 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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <defs style="stroke:green">
    <!-- Note that parent's stroke:green will be ignored 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 a container object which contents are the referenced element:

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <defs style="stroke:green">

```

```

    <!-- Note that parent's stroke:green will be ignored 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](#)

```

<!ELEMENT use (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

```

Attribute definitions:

x = "x-coordinate"

The *x-coordinate* of one corner of the rectangular region into which the referenced element should be placed. The default *x-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#). (??? Needs to be expanded to take care of referencing graphics elements.)

y = "y-coordinate"

The *y-coordinate* of one corner of the rectangular region into which the referenced element should be placed. The default *y-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#). (??? Needs to be expanded to take care of referencing graphics elements.)

width = "length"

The width of the rectangular region into which the referenced element should be placed. (??? Default value?) (??? Needs to be expanded to take care of referencing graphics elements.)

height = "length"

The height of the rectangular region into which the referenced element should be placed. (??? Default value?) (??? Needs to be expanded to take care of referencing graphics elements.)

href = "uri-reference"

A [URI reference](#) to an element/fragment within an SVG document.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#).

3.7 The <image> element

The <image> element indicates that the contents of a complete file should 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.

Unlike [<use>](#), the <image> element cannot reference elements within an SVG file.

```
<!ELEMENT image (desc?,title?,(animate|animateTransform)*) >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

x = "x-coordinate"

The *x-coordinate* of one corner of the rectangular region into which the referenced document should be placed. The default *x-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#). (??? Needs to be expanded to take care of referencing graphics elements.)

y = "y-coordinate"

The *y-coordinate* of one corner of the rectangular region into which the referenced document should be placed. The default *y-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#). (??? Needs to be expanded to take care of referencing graphics elements.)

width = "length"

The width of the rectangular region into which the referenced document should be placed.

height = "length"

The height of the rectangular region into which the referenced document should be placed.

href = "uri-reference"

A [URI reference](#).

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#).

A valid example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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-19990812.dtd'>
  <desc>This links to an external image
  </desc>
  <image x="200" y="200" width="100px" height="100px"
    xlink:type="simple" xlink:show="replace" xlink:actuate="user"
    xlink:href="myimage.png">
    <title>My image</title>
  </image>
</svg>
```

[Download this example](#)

4 SVG Rendering Model

Contents

- [4.1 Introduction](#)
- [4.2 The painters model](#)
- [4.3 Rendering Order](#)
- [4.4 Grouping](#)
- [4.5 Types of graphics elements](#)
 - [4.5.1 Painting shapes and text](#)
 - [4.5.2 Painting raster images](#)
- [4.6 Filtering painted regions](#)
- [4.7 Clipping, masking and object opacity](#)
- [4.8 Parent Compositing](#)

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 should 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 may 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 have an implicit drawing order, with the first elements in the SVG document 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. Marker symbols are not applicable to text.

The fill is painted first, then the stroke, and then the marker symbols.

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

5 Clipping, Masking and Compositing

Contents

- [5.1 Introduction](#)
- [5.2 Simple Alpha Blending/Compositing](#)
- [5.3 Clipping paths](#)
- [5.4 Masking](#)
- [5.5 Object And Group Opacity: the 'opacity' Property](#)

5.1 Introduction

SVG supports the following clipping/masking features:

- [clipping paths](#), which uses any combination of `<path>`, `<text>` and [basic shapes](#) to serve as the outline of a (in the absence of 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.)

5.2 Simple Alpha Blending/Compositing

[Insert discussion about color spaces and compositing techniques.]

It is likely that our default compositing colorspace will be linearized-sRGB, where it is linearized with respect to light energy. Any colors specified in sRGB would be composited after linearizing with the formula $(\alpha * \text{src}^{2.2} + (1 - \alpha) * \text{dst}^{2.2})^{(1/2.2)}$.

5.3 Clipping paths

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.

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 and are processed as if the `<use>` element were not present. 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.

If the 'clip-path' property references a non-existent object or if the referenced object is not a `<clipPath>` element, then the 'clip-path' property will be ignored.

For a given drawing element, the actual clipping path used will be the intersection of its specified *clip-path* with the clipping paths of all its ancestors.

(There will be a mechanism for ensuring that an initial clipping path is set to the bounds of the entire viewport into which the SVG is being drawn. The working group is also investigating ways to allow an SVG drawing to draw outside of the initial viewport [i.e., initial clipping path goes beyond the bounds of the initial viewport].)

```
<!ELEMENT clipPath (desc?,title?,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use)*) >
<!ATTLIST clipPath
  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](#).

'clip-path'

Value: <uri> | none
Initial: The bounds of the viewport (see ??? link to viewport).
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

<uri>

A [URI reference](#) to another graphical object within the same SVG document 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

evenodd

(??? Need detailed description plus drawings)

nonzero

(??? Need detailed description plus drawings)

5.4 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>.

If the 'mask' property references a non-existent object or if the referenced object is not a <mask> element, then the 'mask' property will be ignored.

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 the child elements of the <mask> define a one-channel image, then use that channel as the mask.
- If the child elements of the <mask> define a three-channel RGB image, then use the luminance from the image as the mask, where the luminance is calculated using the following formula:
$$1.0 - (.2126 * R^2.2 + .7152 * G^2.2 + .0722 * B^2.2)$$
- If the child elements of the <mask> define a four-channel RGBA image, then use the alpha

channel 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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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:

```
<!ELEMENT mask (defs?,desc?,title?,
                (path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|switch|a|
                 animate)*)>
<!ATTLIST mask
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  maskUnits (userSpace | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED >
```

Attribute definitions:

`maskUnits = "userSpace | objectBoundingBox"`

Defines the coordinate system for attributes `x`, `y`, `width`, `height`. If `maskUnits="userSpace"` (the default), `x`, `y`, `width`, `height` represent values in the current user coordinate system in place at the time when the `<mask>` element is defined. If `maskUnits="objectBoundingBox"`, then `x`, `y`, `width`, `height` 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 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.)

`x = "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 consist 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%.

`y = "y-coordinate"`

The *y-coordinate* of one corner of the rectangle for the largest possible offscreen buffer. The default value for `y` is 0%.

`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 consist 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%.

`height = "length"`

The height of the largest possible offscreen buffer. The default value for `height` is 100%.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#).

The following is a description of the 'mask' property.

'mask'

<i>Value:</i>	<uri> none
<i>Initial:</i>	none
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

<uri>

A [URI reference](#) to another graphical object which will be used as the mask.

5.5 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> | <pct>
Initial: 100%
Applies to: all elements
Inherited: no
Percentages: Yes, relative to the current viewport
Media: visual

<alphavalue>

The uniform opacity setting to be applied across an entire object expressed as an <number> between 0 and 1. If the object is 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>.

<pct>

The uniform opacity setting to be applied across an entire object expressed as a percentage (100% means fully opaque). If the object is 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 <pct> * 100.

6 Styling and CSS

Contents

- [6.1 SVG's Use of Cascading Style Sheets](#)
- [6.2 Referencing External Style Sheets](#)
- [6.3 The <style> element](#)
- [6.4 The class attribute](#)
- [6.5 The style attribute](#)
- [6.6 Cascading and Inheritance of CSS Properties](#)
- [6.7 The Scope/Range of CSS Styles](#)
- [6.8 The 'display' property](#)
- [6.9 'overflow' and 'clip' properties](#)
- [6.10 Default styles sheet for SVG](#)

6.1 SVG's Use of Cascading Style Sheets

As much as possible, SVG conforms to and leverages the "Cascading Style Sheets (CSS) level 2" Recommendation [[CSS2](#)].

SVG uses CSS properties to describe many of its document parameters. In particular, SVG uses CSS 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-align'](#)
 - ['vertical-align'](#)
 - ['text-decoration'](#)
 - ['letter-spacing'](#)
 - ['word-spacing'](#)
 - ['direction'](#)
 - ['unicode-bidi'](#)
- [Other properties for visual media:](#)
 - ['visibility'](#)
 - ['display'](#)
 - ['overflow'](#) (Only applicable to outermost <svg>)
 - ['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 an SVG document 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

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

6.2 Referencing External Style Sheets

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

6.3 The <style> element

```
<!ELEMENT style (#PCDATA)* >
<!ATTLIST style type CDATA #FIXED "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.

The <style> element allows authors to put style sheet rules embedded within an SVG document. <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 may 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 may 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:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

6.4 The class attribute

(??? Not yet written)

6.5 The style attribute

(?? Not yet written)

6.6 Cascading and Inheritance of CSS 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.

(?? Add discussion of properties that don't inherit syntactically but by usage, such as opacity, filters and clip-path.)

6.7 The Scope/Range of CSS Styles

The following define the scope/range of CSS styles:

Stand-alone SVG graphic

There is one parse tree, and all elements are in the SVG namespace. CSS styles defined anywhere within the SVG graphic (in style elements or style attributes, or in external style sheets linked with the stylesheet PI) apply across the entire SVG graphic.

Stand-alone SVG graphic embedded in an HTML document with the or <object> elements

There are two completely separate parse trees; one for the HTML document, and one for the SVG graphic. CSS styles defined anywhere within the HTML document (in style elements or style attributes, or in external style sheets linked with the stylesheet PI) apply across the entire HTML document. Since inheritance is down a parse tree, these styles do not affect the SVG graphic. CSS styles 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 document. To get the same styling across both HTML document and SVG graphic, link them both to the same stylesheet.

Stand-alone SVG graphic textually included in an XML document

There is a single parse tree, using multiple namespaces; one or more subtrees are in the SVG namespace. CSS styles 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 <style> element on the <svg> element. Alternatively, put an ID on the <svg> element and use contextual CSS selectors.

6.8 The 'display' property

To support SVG, the CSS2 [[CSS2](#)] 'display' property is enhanced to include an svg value as follows:

'display'

<i>Value:</i>	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 svg inherit
<i>Initial:</i>	inline
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	all

A value of display: svg indicates that the given element should be rendered by the SVG user agent into the viewport defined by the CSS box supplied as the principal container element for the SVG document.

6.9 'overflow' and 'clip' properties

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. Two properties from CSS2 determine the initial clipping path which the CSS user agent establishes for the SVG document:

'overflow'

<i>Value:</i>	visible hidden scroll auto inherit
<i>Initial:</i>	visible
<i>Applies to:</i>	block-level and replaced elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

'clip'

<i>Value:</i>	<shape> auto inherit
<i>Initial:</i>	auto
<i>Applies to:</i>	block-level and replaced elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

For descriptions of these properties, consult the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)].

6.10 Default styles sheet for SVG

The user agent's default style sheet for SVG should include the following entries:

```
svg { display: block; overflow: hidden }
```

```
svg * { display: svg }
```

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

7 Coordinate Systems, Transformations and Units

Contents

- [7.1 Introduction](#)
- [7.2 Establishing the initial viewport](#)
- [7.3 Establishing A New User Coordinate System: Transformations](#)
- [7.4 Establishing an Initial User Coordinate System: the viewBox attribute](#)
- [7.5 Modifying the User Coordinate System: the transform attribute](#)
- [7.6 Establishing a New Viewport: the <svg> element within an SVG document](#)
- [7.7 Units](#)
- [7.8 Redefining the meaning of CSS unit specifiers](#)
- [7.9 Processing rules for CSS units and percentages](#)
- [7.10 Further Examples](#)

7.1 Introduction

For all media, the SVG document canvas describes "the space where the SVG document 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 document viewport. For visual media, the SVG document viewport is the viewing area where the user sees the SVG document.

The size of the SVG document 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 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 coordinate systems are identical. Both coordinate 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. (The viewport coordinate system is also called viewport space and the user coordinate system is also called user space.)

Lengths in SVG can be specified as:

- (if no unit 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 in the SVG document 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 transformation operations are fundamental operations to 2D graphics and represent the typical way of controlling the size, position, rotation and skew of graphic objects.

New viewports also can be established, but are for more specialized uses. 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 Establishing the initial viewport

The SVG user agent negotiates with its parent user agent using any CSS positioning parameters on the <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. The size (i.e., width and height) of the viewport is determined as follows. If the outermost <svg> element contains CSS positioning properties which establish the width for the viewport, then the width of the viewport should be set to that size. If the CSS positioning properties on the outermost <svg> element do not provide sufficient information to determine the width of the viewport, then the XML attributes width= determines the width of the viewport. Similarly, if the outermost <svg> element contains CSS positioning properties which establish the height for the viewport, then the height of the viewport should be set to that size. If the CSS positioning properties on the <svg> element do not provide sufficient information to determine the height of the viewport, then the XML attributes height= determines the height of the viewport.

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-19990812.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 an SVG document is determined by the actual values of the 'clip' and 'overflow' properties that apply to the outermost <svg> element. (These concepts and properties are defined in the [Cascading Style Sheets, level 2 CSS2 Specification](#).)

7.3 Establishing A New User Coordinate System: Transformations

To change the current user space coordinate system, you define a transformation which defines how to transform coordinates from the new user coordinate system into the previous user coordinate system. Mathematically, the transformation is represented by a transformation matrix which maps coordinates in the new user coordinate system into the previous user coordinate system. To illustrate:

(Insert an image which shows this concept.)

Transformation matrices define the mathematical mapping from one coordinate space into another. Of particular interest is the current transformation matrix (CTM) which defines the mapping from user space into viewport space.

(Insert an image showing the CTM mapping user space into device space.)

Transformation matrices are specified as 3x3 matrices of the following form:

(Insert an image showing $[a\ b\ 0\ c\ d\ 0\ e\ f\ 1]$, but as a rectangular matrix.)

Because SVG's transformation matrices only have six entries that can be changed, these matrices will be called 2x3 transformation matrices, which for convenience are often written as an array of six numbers: $[a\ b\ c\ d\ e\ f]$.

All coordinates in user space are expressed as (x,y) values. To calculate the transformation from the current user space coordinate system into viewport space, you multiply the vector (x,y,1) by the current transformation matrix (CTM) to yield (x',y',1):

(Insert an image showing $[x',y',1]=[x,y,1][a\ b\ 0\ c\ d\ 0\ e\ f\ 1]$)

Whenever a new transformation is provided, a new CTM is calculated by the following formula. Note that the new transformation is pre-multiplied to the CTM:

(Insert an image which shows $\text{newCTM}[a\ b\ c\ d\ e\ f]=[\text{transformmatrix}]*\text{oldCTM}[a\ b\ c\ d\ e\ f]$.)

It is important to understand the following key points regarding transformations in SVG:

- Transformations alter coordinate systems, not objects. All objects defined outside the scope of a transformation are unchanged by the transformation. All objects defined within the scope of a transformation will be drawn in the transformed coordinate system.
- Transformations specify how to map the transformed (new) coordinate system to the untransformed (old) coordinate system. All coordinates used within the scope the transformation are specified in the transformed coordinate system.
- Transformations are always premultiplied to the CTM.

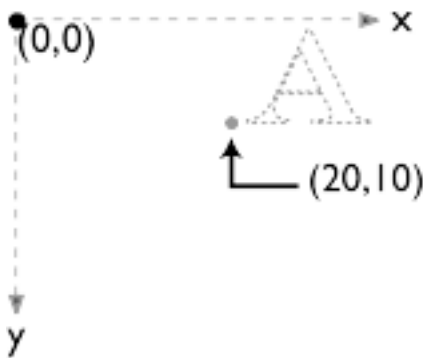
- Matrix operations are not commutative - the order in which transformations are specified is significant. For example, a translation followed by a rotation will yield different results than a rotation followed by a translation:

(Insert an image illustrates the above concept.)

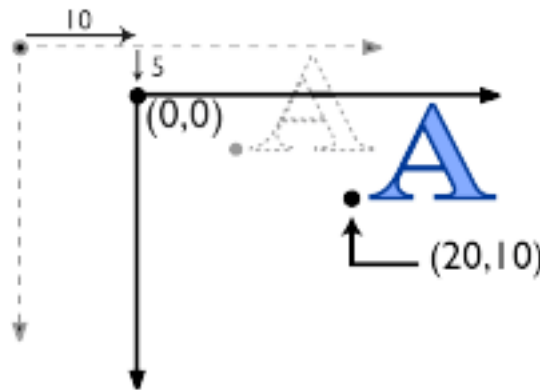
Mathematically, all transformations can be expressed as matrices. To illustrate:

- Translations are represented as $[1 \ 0 \ 0 \ 1 \ tx \ ty]$, where tx and ty are the distances to translate the origin of the coordinate system in x and y , respectively.
- Scaling is obtained by $[sx \ 0 \ 0 \ sy \ 0 \ 0]$. This scales the coordinates so that one unit in the x and y directions of the new coordinate system is the same as sx and sy units in the previous coordinate system, respectively.
- Rotations are carried out by $[\cos(\text{angle}) \ \sin(\text{angle}) \ -\sin(\text{angle}) \ \cos(\text{angle}) \ 0 \ 0]$, which has the effect of rotating the coordinate system axes by angle degrees counterclockwise.
- (Similar examples can be given for skew, reflect and other simple transformations. At this time, the SVG working group is still investigating these other simple transformations.)

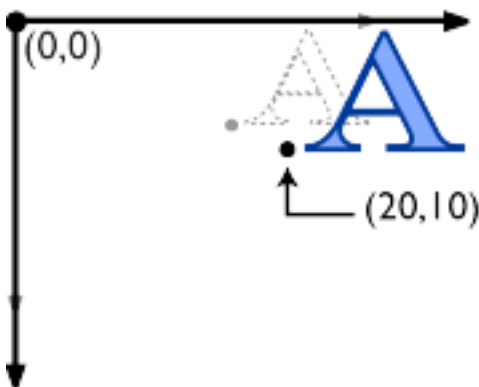
A: Original



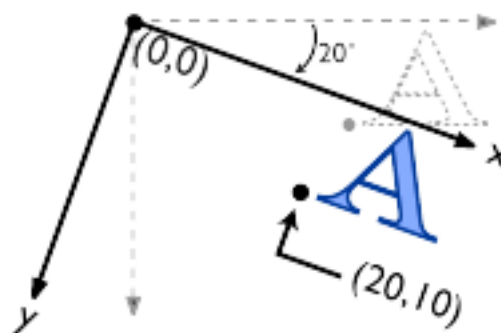
B: transform="translate(10,5)"



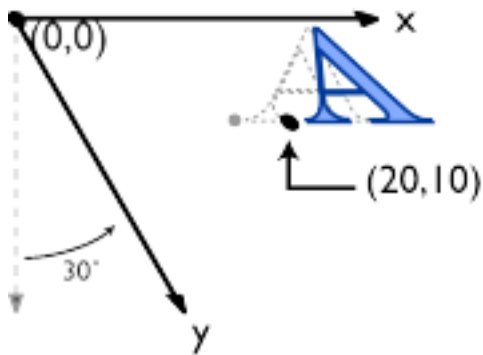
C: transform="scale(1.25)"



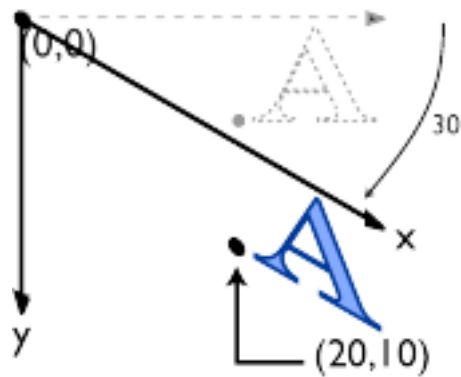
D: transform="rotate(20)"



E: transform="skewX(30)"



F: transform="skewY(30)"



7.4 Establishing an Initial User Coordinate System: the viewBox attribute

Various SVG elements have the effect of establishing a new viewport:

- A [<svg>](#) element establishes 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.

It is very common to have the requirement that a given SVG document, `<symbol>`, `<marker>` or `<pattern>` should be scaled automatically to fit within a target rectangle. The `viewBox="<min-x> <min-y> <width> <height>"` attribute on the `<svg>` and `<symbol>` elements indicates that an initial coordinate system should be established such that the given rectangle in user space (specified by the four numbers `<min-x>` `<min-y>` `<width>` `<height>`) maps exactly to the bounds of the viewport. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in" viewBox="0 0 40 30">
  <desc>This SVG drawing 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 (40,30) in user space.
  Because of the viewBox attribute above,
```

```
the rectangle will end up filling the entire area
reserved for the SVG document. -->
<rect x="0" y="0" width="40" height="30" style="fill: blue" />
</svg>
```

[Download this example](#)

In some cases, it is necessary that the aspect ratio of the graphic be retained when utilizing `viewBox`. A supplemental attribute `preserveAspectRatio="<align> [<meetOrSlice>]"` indicates whether or not to preserve the aspect ratio of the original graphic. The `<align>` parameter indicates whether to preserve aspect ratio and what alignment method should be used if aspect ratio is preserved. The `<align>` parameter must be one of the following strings:

- none (the default) - Do not attempt to preserve aspect ratio. Scale the graphic non-uniformly if necessary such that the graphic's bounding box exactly matches the viewport rectangle.
- xMinYMin - Attempt to preserve aspect ratio. Align the smallest X value of the graphic's bounding box with the smallest X value of the viewport. Align the smallest Y value of the graphic's bounding box with the smallest Y value of the viewport.
- xMidYMin - Attempt to preserve aspect ratio. Align the midpoint X value of the graphic's bounding box with the midpoint X value of the viewport. Align the smallest Y value of the graphic's bounding box with the smallest Y value of the viewport.
- xMaxYMin - Attempt to preserve aspect ratio. Align the maximum X value of the graphic's bounding box with the maximum X value of the viewport. Align the smallest Y value of the graphic's bounding box with the smallest Y value of the viewport.
- xMinYMid - Attempt to preserve aspect ratio. Align the smallest X value of the graphic's bounding box with the smallest X value of the viewport. Align the midpoint Y value of the graphic's bounding box with the midpoint Y value of the viewport.
- xMidYMid - Attempt to preserve aspect ratio. Align the midpoint X value of the graphic's bounding box with the midpoint X value of the viewport. Align the midpoint Y value of the graphic's bounding box with the midpoint Y value of the viewport.
- xMaxYMid - Attempt to preserve aspect ratio. Align the maximum X value of the graphic's bounding box with the maximum X value of the viewport. Align the midpoint Y value of the graphic's bounding box with the midpoint Y value of the viewport.
- xMinYMax - Attempt to preserve aspect ratio. Align the smallest X value of the graphic's bounding box with the smallest X value of the viewport. Align the maximum Y value of the graphic's bounding box with the maximum Y value of the viewport.
- xMidYMax - Attempt to preserve aspect ratio. Align the midpoint X value of the graphic's bounding box with the midpoint X value of the viewport. Align the maximum Y value of the graphic's bounding box with the maximum Y value of the viewport.
- xMaxYMax - Attempt to preserve aspect ratio. Align the maximum X value of the graphic's bounding box with the maximum X value of the viewport. Align the maximum Y value of the graphic's bounding box 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 graphic (as defined by its bounding box) is visible within the viewport
 - the graphic 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 graphic (i.e., the area into which the graphic 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 graphic (as defined by its bounding box)
 - the graphic is scaled down 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 graphic will extend beyond the bounds of the viewport (i.e., the area into which the graphic will draw is larger than the viewport).

7.5 Modifying the User Coordinate System: the transform attribute

All modifications to the user coordinate system are specified with the transform attribute: The transform attribute defines a new coordinate system transformation and thus implicitly a new user space and a new CTM. A transform attribute takes a list of transformations, which are applied in the order provided. The available transformations include:

- `matrix(<a> <c> <d> <e> <f>)`, which specifies that the given transformation matrix should be premultiplied to the old CTM to yield a new CTM.
- `translate(<tx> [<ty>])`, which indicates that the origin of the current user coordinate system should be translated by tx and ty . If $<ty>$ is not provided, it is assumed to be zero.
[A translate is equivalent to `matrix(1 0 0 1 tx ty)`].
- `scale(<sx> [<sy>])`, which indicates that the user coordinate system should be scaled by sx and sy . If $<sy>$ is not provided, it is assumed to be equal to $<sx>$.
[A scale is equivalent to `matrix(sx 0 0 sy 0 0)`].
- `rotate(<rotate-angle>)`, which indicates that the current user coordinate system should be rotated relative to its origin by $<rotate-angle>$, which is expressed in degrees.
[A rotation is equivalent to `matrix(cos(angle) sin(angle) -sin(angle) cos(angle) 0 0)`].
- `skewX(<skew-angle>)`, which indicates that the current user coordinate system should be transformed such that, for positive values of $<skew-angle>$, increasingly positive Y values will be tilted by increasing amounts in the direction of the positive X-axis. (??? Need picture).
 $<skew-angle>$ is expressed in degrees.
[A skewX is equivalent to `matrix(1 0 tan(angle) 1 0 0)`].
- `skewY(<skew-angle>)`, which indicates that the current user coordinate system should be transformed such that, for positive values of $<skew-angle>$, increasingly positive X values will be tilted by increasing amounts in the direction of the positive Y-axis. (??? Need picture).
 $<skew-angle>$ is expressed in degrees.
[A skewY is equivalent to `matrix(1 tan(angle) 0 1 0 0)`].

All values are real numbers.

If a list of transforms is provided, then the net effect is as if each transform had been applied separately in the order provided. For example, transform="translate(-10,-20) scale(2) rotate(45) translate(5,10)" indicates that:

- the origin of the user coordinate system should be translated -10 units in X and -20 units in Y (equivalent to transformation matrix [1 0 0 1 -10 -20]),
- then the user coordinate system should be scaled uniformly by a factor of 2 (equivalent to transformation matrix [2 0 0 2 0 0]),
- then the user coordinate system should be rotated by 45 degrees (equivalent to transformation matrix [cos(45) sin(45) -sin(45) cos(45)]),
- then origin of the user coordinate system should be translated by 5 units in X and 10 units in Y (equivalent to transformation matrix [1 0 0 1 5 10]).

The result is equivalent to pre-multiplying all of the matrices together: [1 0 0 1 5 10] * [cos(45) sin(45) -sin(45) cos(45)] * [2 0 0 2 0 0] * [1 0 0 1 -10 -20], which would be roughly equivalent to the following transform definition: matrix(1.414 1.414 -1.414 1.414 -17.07 1.213).

The transform attribute is applied to an element before processing any other coordinate or length values supplied for that element. Thus, 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. Thus, 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.

7.6 Establishing a New Viewport: the <svg> element within an SVG document

At any point in an SVG drawing, you can establish a new viewport into which all contained graphics should be drawn by including an <svg> element inside an SVG document. 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.

To establish a new viewport, you use the positioning properties from CSS such as 'left', 'top', 'right', 'bottom', 'width', 'height', margin properties and padding properties. Here is an example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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 style="left: 25%; top: 25% width="50%" height="50%">
    <!-- drawing B goes here -->
  </svg>
</svg>
```

[Download this example](#)

You can also specify values for the 'clip' and 'overflow' properties for <svg> elements within an SVG document. If specified on an <svg> element, these properties will change the current clipping path.

(Note that these properties will be ignored if used on any other type of element.)

7.7 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 unit designator is provided on a coordinate or length value, then the given value is assumed to be in CSS units. Available unit designators are the absolute and relative unit designators from CSS (*em*, *ex*, *px*, *pt*, *cm*, *mm*, *in* and percentages). As in CSS, the *em* and *ex* unit designators 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*, *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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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 should 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 should assume a *px* is defined to be exactly .28mm.

7.8 Redefining the meaning of CSS unit specifiers

The process of [establishing a new viewport](#) via an <svg> element inside of an SVG document changes the meaning of the following CSS unit specifiers: px, pt, 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, 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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

7.9 Processing rules for CSS units and percentages

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

- For any x-coordinate value or width value ($xValueInVP\text{Space}$) expressed using CSS units (other than percentages), first convert $xValueInVP\text{Space}$ 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 width value for the given operation.
- For any y-coordinate value or height value ($yValueInVP\text{Space}$) 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 ($xValueInVP\text{Space}$) 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 width value for the given operation.
- For any y-coordinate value or height value ($yValueInVP\text{Space}$) 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 ($lengthVP\text{Space}$), the following approach is used to give appropriate weighting to the contribution of the two dimensions of the viewport. First, convert $lengthVP\text{Space}$ into viewport pixel units using the SVG user agent's standard conversion factor from pixels 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 $lengthVPPixels=percentageValue*\sqrt{vpWidthPixels^2+vpHeightPixels^2}/\sqrt{2}$.

7.10 Further Examples

(Include an example which shows multiple viewports, multiple user spaces and multiple use of different units.)

8 Painting (Filling and Stroking)

Contents

- [8.1 Introduction](#)
- [8.2 Fill Properties](#)
- [8.3 Stroke Properties](#)
- [8.4 Markers](#)
 - [8.4.1 Introduction](#)
 - [8.4.2 The <marker> element](#)
 - [8.4.3 Marker properties](#)
 - [8.4.4 Details on How Markers are Rendered](#)
- [8.5 Rendering Properties](#)
- [8.6 Inheritance of Painting Properties](#)

8.1 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 define a gradient by including a [<gradient>](#) element within a [<defs>](#), assign an ID to that [<gradient>](#) object, and then reference that ID in a 'fill' or 'stroke' property:

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

8.2 Fill Properties

'fill'

Value: none |
currentColor |
<color> [icc-color(<colorvalue>*)] |
inherit |
<uri> [none | currentColor | <color> [icc-color(<colorvalue>*)] | inherit]

Initial: currentColor

Applies to: all elements

Inherited: see [Inheritance of Painting Properties](#) below

Percentages: N/A

Media: visual

none

Indicates that the object should not be filled.

currentColor

Indicates that the object should be 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(<colorvalue>*)]

<color> is the explicit color (in the sRGB color space) to be used to fill the current object. SVG supports all of CSS2's <color> specifications. If an optional **[icc-color(<colorvalue>*)]** is provided, then the list of <colorvalue>'s is a set of ICC-profile-specific color values. On platforms which support ICC-based color management, the **icc-color** gets precedence over the <color> (which is in the sRGB color space). For more on ICC-based colors, refer to [Color](#).

<uri> [none | currentColor | <color> | inherit]

The <uri> is how you identify a fancy paint style such as a gradient, a pattern or a custom paint from extensibility. The <uri> should provide the ID of the paint server (e.g., a gradient [??? see link] or a pattern [??? see link]) 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> | **inherit**) is used if provided; otherwise, no gradient will occur.

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

evenodd

(??? Need detailed description plus drawings)

nonzero

(??? Need detailed description plus drawings)

'fill-opacity'

Value: <opacity-value>
Initial: 100%
Applies to: all elements
Inherited: yes
Percentages: Allowed
Media: visual

'fill-opacity' specifies the opacity of the painting operation used to fill the current object. It is important to note that any given object can have three different opacity properties: '[fill-opacity](#)', '[stroke-opacity](#)' and '[opacity](#)'. The 'fill' painting operation is done and blended into the current background (or temporary offscreen buffer, if 'opacity' is not 1.0) using the value of 'fill-opacity'. Next, The 'stroke' painting operation is done and blended into the current background (or temporary offscreen buffer, if 'opacity' is not 1.0) using the value of 'stroke-opacity'. Finally, if 'opacity' is not 1.0, the offscreen holding the object as a whole is blended into the current background.

(The above paragraph needs to be moved someplace else, such as SVG Rendering Model.)

<opacity-value>

The opacity of the painting operation used to fill the current object. If a <number> is provided, then it must be in the range of 0.0 (fully transparent) to 1.0 (fully opaque). If a percentage is provided, then it must be in the range of 0% to 100%. Any values outside of the acceptable range are rounded to the nearest acceptable value.

8.3 Stroke Properties

'stroke'

Value: none |
currentColor |
<color> [icc-color(<number>*)] |
inherit |
<uri> [none | currentColor | <color> | inherit]

Initial: none

Applies to: all elements

Inherited: see [Inheritance of Painting Properties](#) below

Percentages: N/A

Media: visual

none

(Same meaning as with ['fill'](#).)

currentColor

(Same meaning as with ['fill'](#).)

<color>

(Same meaning as with ['fill'](#).)

<uri> [none | currentColor | <color> [icc-color(<colorvalue>*)] | inherit]

(Same meaning as with ['fill'](#).)

'stroke-width'

Value: <width> | inherit

Initial: 1

Applies to: all elements

Inherited: yes

Percentages: Yes

Media: visual

<width>

The width of the stroke on the current object, either expressed as a <length> in user units, a <length> in Transformed CSS units (??? add link) or as a percentage. If a percentage is used, the <width> is expressed as a percentage of the current viewport (??? add link).

'stroke-linecap'

Value: butt | round | square | inherit

Initial: butt

Applies to: all elements

Inherited: yes

Percentages: N/A

Media: visual

'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.

(??? insert drawing here)

'stroke-linejoin'

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

'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.

(??? insert drawing here)

'stroke-miterlimit'

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

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.

(??? insert drawing here)

'stroke-dasharray'

Value: none | <dasharray> | inherit
Initial: none
Applies to: all elements
Inherited: yes
Percentages: Yes. See below.
Media: visual

'stroke-dasharray' controls the pattern of dashes and gaps used to stroke paths. **<dasharray>** should contain 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 should be used. If stroked, the line should be 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.](#))

(??? insert drawing here)

'stroke-dashoffset'

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

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

<dashoffset>

A <length> 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 (??? Add link here).

(??? insert drawing here)

'stroke-opacity'

Value: <opacity-value> | inherit
Initial: 100%
Applies to: all elements
Inherited: yes
Percentages: Allowed
Media: visual

'stroke-opacity' specifies the opacity of the painting operation used to stroke the current object. (??? Add link about how different opacity parameters interact.)

<opacity-value>

The opacity of the painting operation used to stroke the current object. If a <number> is provided, then it must be in the range of 0.0 (fully transparent) to 1.0 (fully opaque). If a percentage is provided, then it must be in the range of 0% to 100%. Any values outside of the acceptable range are rounded to the nearest acceptable value.

8.4 Markers

8.4.1 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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

8.4.2 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](#).

```
<!ELEMENT marker ( (defs?, desc?, title?,
  (path|text|rect|circle|ellipse|line|polyline|polygon|
  use|image|svg|g|switch|a|
  animate|animateTransform|animateColor)*) >

<!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) "stroke-width"
  markerWidth CDATA "3"
  markerHeight CDATA "3"
  orient CDATA "0">
```

Attribute definitions:

markerUnits = "*strokeWidth* | *userSpace*"

markerUnits indicates how to interpret the values of **markerWidth** and **markerHeight** (described as follows). 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="stroke-width"**, then **markerWidth** and **markerHeight** represent

scale factors relative to the stroke width in place for graphic object referencing the marker.

`markerWidth = "length"`

Represents the width of the temporary viewport that is to be created when drawing the marker. Default value is "3".

`markerHeight = "length"`

Represents the height of the temporary viewport that is to be created when drawing the marker. Default value is "3".

`orient = "auto | <angle>"`

Indicates how the marker should be rotated. A value of *auto* indicates that the marker should be 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 degrees.

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.

8.4.3 Marker properties

'**marker-start**' defines the arrowhead or polymarker that should be drawn at the first vertex of the given `<path>` element or [vector graphic shape](#). '**marker-end**' defines the arrowhead or polymarker that should be drawn at the final vertex. '**marker-mid**' defines the arrowhead or polymarker that should 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

none

Indicates that no marker symbol should be drawn at the given vertex (vertices).

<uri>

The `<uri>` is a [URI reference](#) to the ID of a `<marker>` element which should 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) should not be drawn.

The '**marker**' property specifies the marker symbol that should 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

8.4.4 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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

8.5 Rendering Properties

The creator of an SVG document 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: default | optimizeSpeed | crispEdges |
geometricPrecision | inherit

Initial: false

Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

default

Indicates that the user agent should 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 should 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 should 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 should emphasize geometric precision over speed and crisp edges.

The creator of an SVG document 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: default | optimizeSpeed | optimizeLegibility |
geometricPrecision | inherit
Initial: default
Applies to: <text> elements
Inherited: yes
Percentages: yes
Media: visual

default

Indicates that the user agent should 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 should 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 should 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 should 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 an SVG document 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: default | optimizeSpeed | optimizeQuality | inherit
Initial: default
Applies to: images
Inherited: yes
Percentages: N/A
Media: visual

default

Indicates that the user agent should make appropriate tradeoffs to balance speed and quality, but quality should be given more importance than speed.

optimizeSpeed

Indicates that the user agent should 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 should 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 should be rendered at all.

'visibility'

Value: visible | hidden | inherit
Initial: visible
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

visible

The current object should be drawn.

hidden

The current object should not be drawn.

8.6 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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

9 Built-in Types of Paint

Contents

- [9.1 Introduction](#)
- [9.2 Color](#)
 - [9.2.1 Introduction](#)
 - [9.2.2 Properties for specifying color profiles](#)
- [9.3 Gradients](#)
 - [9.3.1 Introduction](#)
 - [9.3.2 Linear Gradients](#)
 - [9.3.3 Radial Gradients](#)
 - [9.3.4 Gradient Stops](#)
- [9.4 Patterns](#)

9.1 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**. Color, gradients and patterns are just specific types of paint servers. For example, first you define a gradient by including a `<gradient>` element within a `<defs>`, assign an ID to that `<gradient>` object, and then reference that ID in a 'fill' or 'stroke' property:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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](#)

9.2 Color

9.2.1 Introduction

All SVG colors are specified in the sRGB color space (see [\[SRGB\]](#)). At a minimum, SVG user agents should conform to the color behavior requirements specified in the Colors chapter of the CSS2 specification (see [\[CSS2\]](#)).

Additionally, SVG documents 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 and specifies the color values for [gradient stops](#). The color property is extended from CSS2 to accommodate color definitions in arbitrary color spaces defined by the ['color-space'](#) property.

'color'

Value: <color> [icc-color(<colorvalue>*)] | inherit
Initial: depends on user agent
Applies to: ['fill'](#) and ['stroke'](#) properties and [gradient stops](#).
Inherited: see [Inheritance of Painting Properties](#)
Percentages: N/A
Media: visual

For a description of the parameters, refer to the ['fill'](#) property.

9.2.2 Properties for specifying color profiles

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.

The ['color-profile'](#) property identifies the ICC profile which should be used to process all <icc-color> definitions within the current object.

'color-profile'

Values: auto | sRGB | <uri> | inherit
Initial: auto

Applies to: all elements
Inherited: yes
*Percentages:*n/a
Media: visual

This property permits the specification of a source color profile other than the default.

auto

This is the default behavior. All colors are presumed to be defined in the sRGB color space unless a more precise embedded profile is specified within content data. For images that do have a profile built into their data, that profile is used. For images that do not have a profile, the sRGB profile is used so that the colors in these images can be kept "in synch" with the colors specified in CSS and HTML.

sRGB

The source profile is assumed to be 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. Just like specifying **sRGB**, it overrides an embedded profile. 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 should be identical.

'rendering-intent'

Values: auto | perceptual | relative-colorimetric |
saturation | absolute-colorimetric | inherit
Initial: auto
Applies to: all elements
Inherited: yes
*Percentages:*n/a
Media: visual

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 should be assumed that the intent specified within the profile is the desired intent. Otherwise, the user agent should use the current profile (based on the *color-profile* style) and force the intent, overriding any intent that may be stored in the profile itself.

9.3 Gradients

9.3.1 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](#).(??? Include drawing)

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 should be filled or stroked with the referenced gradient.

9.3.2 Linear Gradients

Linear gradients are defined by a <linearGradient> element.

```
<!ELEMENT linearGradient (stop|animate|animateTransform)* >
<!ATTLIST linearGradient
  id ID #IMPLIED
  gradientUnits (userSpace | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  x1 CDATA #IMPLIED
  y1 CDATA #IMPLIED
  x2 CDATA #IMPLIED
  y2 CDATA #IMPLIED
  spreadMethod (pad | reflect | repeat) "pad">
```

Attribute definitions:

`gradientUnits = "userSpace | 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="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.)

`gradientTransform = "transform"`

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. **gradientTransform** can take on the same values as the [transform](#) attribute.

`x1 = "x-coordinate"`

x1, **y1**, **x2**, **y2** define a *gradient vector* for the linear gradient. This *gradient vector* provides starting and ending points onto which the <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%".

y1 = "y-coordinate"

See [x1](#). Default value is "0%".

x2 = "x-coordinate"

See [x1](#). Default value is "100%".

y2 = "y-coordinate"

See [x1](#). Default value is "0%".

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.

Attributes defined elsewhere:

[id](#).

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.

(??? Need to include some drawings here showing these attributes)

9.3.3 Radial Gradients

Radial gradients are defined by a `<radialGradient>` element.

```
<!ELEMENT radialGradient (stop|animate|animateTransform)* >
<!ATTLIST radialGradient
  id ID #IMPLIED
  gradientUnits (userSpace | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  cx CDATA #IMPLIED
  cy CDATA #IMPLIED
  r CDATA #IMPLIED
  fx CDATA #IMPLIED
  fy CDATA #IMPLIED>
```

Attribute definitions:

`gradientUnits` = "*userSpace* | *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="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.)

`gradientTransform` = "*transform*"

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. `gradientTransform` can take on the same values as the `transform` attribute.

`cx` = "*x-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%".

`cy` = "*y-coordinate*"

See `cx`. Default value is "50%".

`r` = "*length*"

See `cx`. Default value is "50%".

`fx` = "*x-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%.

`fy` = "*y-coordinate*"

See `fx`. Default value is "50%".

Attributes defined elsewhere:

[id](#).

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.

(??? Need to include some drawings here showing these attributes)

9.3.4 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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <desc>Radial gradient example with three gradient stops
  </desc>
  <g>
    <defs>
      <radialGradient id="MyGradient">
        <stop offset="0%" style="color:white"/>
        <stop offset="50%" style="color:red"/>
        <stop offset="100%" style="color:black"/>
      </radialGradient>
    </defs>
    <circle style="fill: url(#MyGradient)" r="42"/>
  </g>
</svg>
```

[Download this example](#)

```
<!ELEMENT stop (animate|animateColor)* >
<!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 should be 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.

Attributes defined elsewhere:

[id](#), [style](#).

The 'stop-color' property (non-inherited) indicates what color to use at that gradient stop. All valid CSS2 color property specifications are available.

An 'stop-opacity' property (non-inherited) can be used to define the opacity of a given gradient stop.

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 should 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 should 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.

9.4 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:**.

```
<!ELEMENT pattern ( (defs?,desc?,title?,
                    (path|text|rect|circle|ellipse|line|polyline|polygon|
                     use|image|svg|g|switch|a|
                     animate|animateTransform) *) )>
<!ATTLIST pattern
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  patternUnits (userSpace | 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' >
```

Attribute definitions:

`patternUnits = "userSpace | objectBoundingBox"`

Defines the coordinate system for attributes **x**, **y**, **width**, **height**. If `patternUnits="userSpace"` (the default), **x**, **y**, **width**, **height** represent values in the current user coordinate system when the

<pattern> element is defined. If **patternUnits="objectBoundingBox"**, then **x, y, width, height** 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 pattern, 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.)

`patternTransform = "transform"`

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. **patternTransform** can take on the same values as the [transform](#) attribute.

`x = "x-coordinate"`

x, y, width, height indicate how the pattern tiles should be placed and spaced and represent coordinates and values in the coordinate space specified by **patternUnits**. Default value is "0%".

`y = "y-coordinate"`

See [x](#). Default value is "0%".

`width = "width"`

See [x](#). Default value is "100%".

`height = "height"`

See [x](#). Default value is "100%".

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [refX](#), [refY](#), [viewBox](#), [preserveAspectRatio](#).

An example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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" />
    </defs>
    <!-- Fill this ellipse with the above pattern -->
    <ellipse style="fill: url(#TrianglePattern)" rx="40" ry="27" />
</svg>
```

[Download this example](#)

10 Paths

Contents

- [10.1 Introduction](#)
- [10.2 The <path> element](#)
- [10.3 Path Data](#)
 - [10.3.1 General information about path data](#)
 - [10.3.2 The "moveto" commands](#)
 - [10.3.3 The "closepath" command](#)
 - [10.3.4 The "lineto" commands](#)
 - [10.3.5 The curve commands](#)
 - [10.3.6 The grammar for path data](#)

10.1 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.

10.2 The <path> element

```
<!ELEMENT path (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  d CDATA #REQUIRED
  flatness CDATA #IMPLIED
  nominalLength CDATA #IMPLIED >
```

Attribute definitions:

`d` = "path data"

The definition of the outline of a shape. See [Path data](#).

`flatness` = "value"

(??? Will be provided later) Used in calculating distance along a path.

`nominalLength` = "length"

The expected length for the distance calculation along the path. Use to calibrate text-on-a-path and animation. (??? More details are forthcoming.)

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

10.3 Path data

10.3.1 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-19990812.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, you should say: "13000.56".)

(??? Need to clean this up) 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.

10.3.2 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.
--	--------	--------	---

10.3.3 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).

10.3.4 The "lineto" commands

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

Command	Name	Parameters	Description
L (absolute) I (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; I (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.

10.3.5 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**, **a**, **B** and **b**). An elliptical arc segment draws a segment of an ellipse defined by the formulas:

$$x = cx + rx * \cos(\theta)$$

$$y = cy + ry * \sin(\theta)$$

where the elliptical arc is drawn as a sweep for every possible *theta* between a given start angle and end angle.

The cubic bezier commands are as follows:

Command	Name	Parameters	Description

<p>C (absolute) c (relative)</p>	<p>curveto</p>	<p>(x1 y1 x2 y2 x y)+</p>	<p>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.</p>
<p>S (absolute) s (relative)</p>	<p>shorthand/smooth curveto</p>	<p>(x2 y2 x y)+</p>	<p>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.</p>

The quadratic bezier commands are as follows:

Command	Name	Parameters	Description

<p>Q (absolute) q (relative)</p>	<p>quadratic bezier curveto</p>	<p>(x1 y1 x y)+</p>	<p>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.</p>
<p>T (absolute) t (relative)</p>	<p>Shorthand/smooth quadratic bezier curveto</p>	<p>(x y)+</p>	<p>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.</p>

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 should be 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).)

10.3.6 The grammar for path data

(??? This requires clean-up and a more formal write-up on the terminology.) The following is the BNF for SVG paths. The following notation is used:

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

```
svg-path:  
  wsp* subpaths?
```

```
subpaths:  
  subpath  
  | subpath subpaths
```

```
subpath:  
  moveto subpath-elements?
```

```
subpath-elements:  
  subpath-element-wsp  
  | subpath-element-wsp subpath-elements
```

```
subpath-element-wsp:  
  subpath-element wsp*
```

```
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 lineto-argument-sequence
```

```
closepath:  
  ( "Z" | "z" ) wsp*
```

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

```
lineto-argument-sequence:  
  coordinate-pair  
  | coordinate-pair lineto-argument-sequence
```

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

```
horizontal-lineto-argument-sequence:  
  horizontal-lineto-argument
```

```

    | horizontal-lineto-argument horizontal-lineto-argument-sequence
horizontal-lineto-argument:
    coordinate

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

vertical-lineto-argument-sequence:
    vertical-lineto-argument
    | vertical-lineto-argument vertical-lineto-argument-sequence

vertical-lineto-argument:
    coordinate

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

curveto-argument-sequence:
    curveto-argument
    | curveto-argument curveto-argument-sequence

curveto-argument:
    coordinate-pair coordinate-pair coordinate-pair

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

smooth-curveto-argument-sequence:
    smooth-curveto-argument
    | smooth-curveto-argument smooth-curveto-argument-sequence

smooth-curveto-argument:
    coordinate-pair 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
      quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument:
    coordinate-pair 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 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 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 coordinate

coordinate:

```

```
number-comma-wsp
nonnegative-number-comma-wsp:
  nonnegative-number wsp* comma? wsp*

number-comma-wsp:
  number wsp* comma? wsp*

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

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

flag-comma-wsp:
  flag wsp* comma? wsp*

flag:
  "0" | "1"

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)+
```

11 Basic Shapes

Contents

- [11.1 Introduction](#)
- [11.2 The <rect> element](#)
- [11.3 The <circle> element](#)
- [11.4 The <ellipse> element](#)
- [11.5 The <line> element](#)
- [11.6 The <polyline> element](#)
- [11.7 The <polygon> element](#)

11.1 Introduction

SVG contains the following set of basic shapes:

- `<rect/>` (a rectangle with optional rounding attributes `rx` and `ry` which represents the radii of an ellipse [axis-aligned with the rectangle] to use to round off the corners of the rectangle);
- `<circle/>`
- `<ellipse/>`
- `<polyline/>`
- `<polygon/>`
- `<line/>`

Mathematically, these shape elements are equivalent to the path objects that would construct the same shape. They may be stroked, filled and used as clip paths, and all the properties described above for paths apply equally to them.

11.2 The <rect> element

```

<!ELEMENT rect ( desc?, title?, ( animate | animateTransform | animateColor ) * ) >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  rx CDATA #IMPLIED
  ry CDATA #IMPLIED >

```

Attribute definitions:

`x = "x-coordinate"`

The *x-coordinate* of one corner of the rectangular. The default *x-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#).

`y = "y-coordinate"`

The *x-coordinate* of one corner of the rectangular. The default *y-coordinate* is zero. See [Coordinate Systems, Transformations and Units](#).

`width = "width"`

The width of the rectangle. See [Coordinate Systems, Transformations and Units](#).

`height = "height"`

The width of the rectangle. See [Coordinate Systems, Transformations and Units](#).

`rx = "length"`

For rounded rectangles, the x-axis radius of the ellipse used to round off the corners of the rectangle.

`ry = "length"`

For rounded rectangles, the y-axis radius of the ellipse used to round off the corners of the rectangle.

If a value is not provided for rx but a value is provided for ry, then rx is set to the same value as ry. If a value is not provided for ry but a value is provided for rx, then ry is set to the same value as rx. If values are not provided for rx or ry, no rounding occurs.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

11.3 The <circle> element

```
<!ELEMENT circle (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
  r CDATA #REQUIRED >
```

Attribute definitions:

`cx` = "x-coordinate"

The *x-coordinate* of the center of the circle.

`cy` = "y-coordinate"

The *y-coordinate* of the center of the circle.

`r` = "length"

The radius of the circle.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

11.4 The <ellipse> element

```

<!ELEMENT ellipse (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  cx CDATA "0"
  cy CDATA "0"
  rx CDATA #REQUIRED
  ry CDATA #REQUIRED >

```

Attribute definitions:

`cx` = "x-coordinate"

The *x-coordinate* of the center of the ellipse.

`cy` = "y-coordinate"

The *y-coordinate* of the center of the ellipse.

`rx` = "length"

The x-axis radius of the ellipse.

`ry` = "length"

The y-axis radius of the ellipse.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

11.5 The <line> element

```

<!ELEMENT line (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  x1 CDATA "0"
  y1 CDATA "0"
  x2 CDATA "0"
  y2 CDATA "0" >

```

Attribute definitions:

x1 = "x-coordinate"

The *x-coordinate* of the start of the line.

y = "y-coordinate"

The *x-coordinate* of the start of the line.

x2 = "x-coordinate"

The *x-coordinate* of the end of the line.

y2 = "x-coordinate"

The *y-coordinate* of the end of the line.

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

11.6 The <polyline> element

```
<!ELEMENT polyline (desc?,title?,(animate|animateTransform|animateColor))* >
<!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 CDATA #IMPLIED
  points CDATA #REQUIRED >
```

Attribute definitions:

points = "x-y-coordinate-pairs"

The points that make up the polyline. Space- or comma-separated.(??? Need to provide a BNF)

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

11.7 The <polygon> element

```

<!ELEMENT polygon (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  points CDATA #REQUIRED >

```

Attribute definitions:

points = "x-y-coordinate-pairs"

The points that make up the polygon. Space- or comma-separated.(??? Need to provide a BNF)

Attributes defined elsewhere:

[id](#), [xml:lang](#), [xml:space](#), [class](#), [style](#), [transform](#), [%graphicsElementEvents;](#), [system-required](#).

For example, the following will draw a blue circle with a red outline:

```

<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <desc>This is a blue circle with a red outline
  </desc>
  <g>
    <circle style="fill: blue; stroke: red"
      cx="200" cy="200" r="100"/>
  </g>
</svg>

```

[Download this example](#)

This ellipse uses default values for the center.

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <desc>This is an ellipse, axis aligned and centered on the origin
  </desc>
  <g>
    <ellipse rx="85" ry="45"/>
  </g>
</svg>

```

[Download this example](#)

Here is a polyline; for comparison, the equivalent path element is also given.

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">

```

```
<desc>A sample polyline, and equivalent path
</desc>
<polyline points="20,20 50,100 200,80 70,300"/>
<path d="M20,20 L50,100 L200,80 L70,300"/>
</svg>
```

[Download this example](#)

A polygon is exactly the same as a polyline, except that the figure is automatically closed.

The points attribute for polylines and polygons (i.e., the list of vertices) 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 points value.

12 Text

Contents

- [12.1 Introduction](#)
- [12.2 The <text> element](#)
- [12.3 White space handling](#)
- [12.4 Text selection](#)
- [12.5 Text and font properties](#)
 - [12.5.1 Introduction](#)
 - [12.5.2 CSS font properties used by SVG](#)
 - [12.5.3 CSS text properties used by SVG](#)
- [12.6 Ligatures and alternate glyphs](#)
- [12.7 Text on a path](#)
 - [12.7.1 Vector graphics along a path](#)

12.1 Introduction

SVG allows text to be inserted into the drawing. All of the same styling attributes available for paths and vector graphic shapes are also available for text. (See [Filling, Stroking and Paint Servers](#).)

12.2 The <text> element

The <text> element adds text to a drawing.

In the example below, the string "Hello, out there" is drawn in blue:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <text x=".5in" y="2in"
    style="fill:blue">Hello, out there</text>
</svg>
```

[Download this example](#)

<text> elements which are not positioned along a path (see [Text On A Path](#) below) can supply optional attributes *x=* and *y=* which represents the location in user coordinates (or Transformed CSS units or a percentage of the current viewport) for the initial *current text position*. Default values for *x=* and *y=* are zero in the user coordinate system. For left-aligned character strings, the placement point of the first glyph will be positioned exactly at the *current text position* and the glyph will be rendered based on the current set of text and font properties. After the glyph is rendered, the *current text position* is advanced based on the metrics of the glyph that were used along with the current set of text and font properties. For Roman text, the *current text position* is advanced along the x-axis by the width of that glyph. The cycle repeats until all characters in the <text> element have been rendered.

Within a <text> element, text and font properties and the *current text position* can be modified by including a <tspan> element. The <tspan> element can contain attributes **style** (which allows new visual rendering attributes to be specified) and the following attributes, which perform adjustments on the *current text position*:

- *x* and *y* - If provided, these attributes indicate new (absolute) *current text position(s)* within the user coordinate system for subsequent glyphs. If a single value is provided, then a new current text position is established before drawing the next subsequent glyph. If a comma- or space-separated list of values is provided, then the first value represents a new current text position for the next subsequent glyph, the second value represents a new current text position before rendering the next glyph, and so on until either there are no more glyphs or no more values. If there are more glyphs to be rendered than values provided, then each extra glyph is rendered at the current text position that resulted after rendering and advancing the previous glyph.
- *dx* and *dy* - If provided, these attributes indicate new (relative) *current text position(s)*. The meaning of the values depends on the value of attribute **dCoordUnits**. Like **dx=** and **dy=**, a list of space-separated or comma-separated values can be provided.
- **dCoordUnits=** - Can take on values *userSpace* (the default), *em* or *fontSpace*. **dCoordUnits=** indicates the coordinate system of attributes **dx=** and **dy=**. If **dCoordUnits="userSpace"**, then *dx* and *dy* are values in the user coordinate system. If **dCoordUnits="em"**, then *dx* and *dy* are values relative to the current 'font-size', where 1em equals the same distance as the current value of property 'font-size' would span along the Y coordinate axis of the current user coordinate system. If **dCoordUnits="fontSpace"**, then *dx* and *dy* are values in the coordinate system of the font used to render the given glyph. (This choice is very font specific and should only be used when it is certain that a particular font is available and will be used to render the given glyph and when you have detailed knowledge about the font's coordinate system.)

The *x* and *dx* values are cumulative; thus, if both are provided, the new *current text position* will have an X-coordinate of *x+dx* (i.e., after *dx* is converted into the user coordinate system). Similarly, if both *y* and *dy* are provided, the new *current text position* will have a Y-coordinate of *y+dy* (i.e., after *dy* is converted into the user coordinate system).

(Internationalization issues need to be addressed.)

A <tspan> element can also be used to specify that the character data from a different element should be used as character data for the given <tspan> element. In the example below, the first <text> element (with **id="TextToUse"**) will not draw because it is part of a <defs> element. The second <text> element draws the string "ABC". The third text element draws the string "XYZ" because it includes a <tspan> element which is a reference to element "TextToUse", and that element's character data is "XYZ":

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <defs>
    <text id="TextToUse">XYZ</text>
  </defs>
  <text>ABC</text>
  <text>
    <tspan xlink:href="#TextToUse"/>
  </text>
</svg>

```

[Download this example](#)

If a `<tspan>` element has both an *href* attribute and its own character data, the character data from the *href* attribute draws before its own character data.

12.3 White space handling

SVG supports the standard XML attribute **xml:space** for determining how it should handle 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 convert all carriage returns, linefeeds and tab characters into space characters. Then, it will strip off all leading and trailing space characters. Finally, it will consolidate all contiguous space characters into a single space character. (This algorithm is very close to standard practice in HTML4 web browsers.)
- **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").

12.4 Text selection

SVG user agents running on systems with have clipboards for copy/paste operations and which are equipped with input devices that allow for text selection should support the selection of text from an SVG document and the ability to copy selected text strings to the system clipboard.

Within an SVG user agent which supports text selection and pointer devices such as a mouse, the following behavior should exist. When the pointing device is clicked over an SVG character and then dragged, then whenever the mouse goes over another character defined within the same `<text>` elements, all characters whose position in the document is between the initial character and the current character should be highlighted, no matter where they might be located on the canvas.

When feasible, generators of SVG should attempt to order their text strings to facilitate properly ordered text selection within SVG viewing applications such as Web browsers.

12.5 Text and font properties

12.5.1 Introduction

SVG uses CSS properties to describe many of its font and text properties.

12.5.2 CSS font properties used by SVG

SVG uses the following font specification properties from CSS2. The detailed description of these properties can be found in the "Cascading Style Sheets (CSS) level 2" specification [[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

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.

'font-style'

Value: normal | italic | oblique | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

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.

'font-variant'

Value: normal | small-caps | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

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.

'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

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.

'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

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.

'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

This property refers to the size of the font from baseline to baseline, when set solid (in CSS terms, this is when the 'font-size' and 'line-height' properties have the same value). Refer to the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)] for more information.

'font-size-adjust'

Value: <number> | none | inherit
Initial: none
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

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.

'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

Shorthand property for setting 'font-style', 'font-variant', 'font-weight', 'font-size', 'line-height' and 'font-family'. The 'line-height' property setting will be ignored by the SVG user agent. [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.

Note that SVG allows unit-less values in CSS properties (see [Units](#)). This causes a potential ambiguity with property settings such as 'font:100 serif' in deciding whether the number represents a 'font-weight' or a 'font-size'. To resolve this ambiguity, font-size specifications in user units within a 'font' shorthand property are required to have the suffix "uu" (e.g., 'font:100uu serif').

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

12.5.3 CSS text properties used by SVG

'text-align'

Value: left | right | center | justify | <string> | inherit

Initial: depends on user agent and writing direction

Applies to: block-level elements

Inherited: yes

Percentages: N/A

Media: visual

This property describes how the characters within a [<text>](#) element should be aligned relative to the initial position for the <text> element. Values have the following meanings:

left, right, and center

Left-align, right-align and center the text, respectively, relative to the initial position for the <text> element. For left, position the first glyph at the initial position. For right and center, before painting the text, determine what the final offset would be if the text were painted with 'text-align:left' and compute the necessary adjustment of the initial position to achieve right- or center- alignment.

justify and <string>

These options from CSS2 [[CSS2](#)] are not supported by SVG. Specifying them will result in a value of left being used.

'vertical-align'

Value: baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <<length> | inherit

Initial: baseline

Applies to: inline-level and 'table-cell' elements

Inherited: no
Percentages: refer to the 'line-height' of the element itself
Media: visual

This property affects the vertical positioning of glyphs within a [<text>](#) elements. Using the CSS2 model for fonts as described in the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)], each font has various characteristics that are either implicit in the font itself or provided explicitly via a CSS2 font descriptor. These characteristics include the following vertical position points: a **baseline**, an **x-height**, a **top**, a **bottom** and, therefore, a **middle**. The 'vertical-align' property indicates how individual glyphs should be aligned vertically with respect to the vertical position points in the relevant font.

baseline

Align the baseline of the glyph with the baseline of the font.

middle

Align the vertical midpoint of the glyph with the baseline of the parent box plus half the x-height of the font.

sub

Lower the baseline of the glyph to the proper position for a subscript for the font.

super

Raise the baseline of the glyph to the proper position for a superscript for the font.

text-top

Align the top of the glyph with the top of the font.

text-bottom

Align the bottom of the glyph with the bottom of the font.

<percentage>

Raise (positive value) or lower (negative value) the glyph by this distance (a percentage of the 'font-size' value). The value '0%' means the same as 'baseline'.

<length>

Raise (positive value) or lower (negative value) the glyph by this distance. The value '0cm' means the same as 'baseline'.

top

Same as text-top.

bottom

Same as text-bottom.

'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

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.

'letter-spacing'

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

This property specifies spacing behavior between text characters. Refer to the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)] for more information.

'word-spacing'

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

This property specifies spacing behavior between words. Refer to the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)] for more information.

'text-advance'

Value: horizontal | vertical | vertical-ideographic | inherit
Initial: horizontal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual

This property specifies whether text strings should be drawn as horizontal or vertical text. (More detailed descriptions will come later.)

'direction'

Value: ltr | rtl | inherit
Initial: ltr
Applies to: all elements, but see prose
Inherited: yes
Percentages: N/A
Media: visual

This property specifies the base writing direction of text. Refer to the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)] for more information.

'unicode-bidi'

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

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

12.6 Ligatures and 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 a different glyph should be used to render some text than the glyph which normally corresponds to the given character data. Also, the W3C Character Model (??? add link) requires early normalization of character data to facilitate meaningful and unambiguous exchange of character data and correct comparisons between character strings. The W3C Character Model will bring about many common situations where the normalized character data will be different than the glyphs which the user want to use to render that character data.

To allow for control over the glyphs used to render particular character data, the '**altglyph**' property is available.

'altglyph'

Value: unicode(<value>) |
 glyphname(<string>) |
 glyphid(<value>) |
 ROS(<value>) cid(<value>) |
 inherit

Initial: none

Applies to: <text> elements

Inherited: yes

Percentages: N/A

Media: visual

unicode(<value>))

where <value> indicates a string of Unicode characters that should replace the text within the <text> element

glyphname(<string>))

where <string> provides a string of which is the name of the glyph that should be used to replace the text within the <text> element

glyphid(<value>))

where <value> a string of which is numeric ID/index of the glyph that should be used to replace the text within the <text> or <t> element

ROS and cid

are required for Web fonts in OpenType/CFF format and operate similar to *glyphid*

12.7 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. The basic algorithm is such that each glyph is positioned at a specific point on the path, with the glyph getting rotated such that the baseline of the glyph is either parallel or tangent to the tangent of the curve at the given point. Here are some examples of text on a path:

(??? include some drawings here)

The following is a simple example which illustrates many of the basic concepts:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<svg width="4in" height="3in">
  <desc>Simple text on a path
  </desc>
  <defs>
    <path id="MyPath"
          d="M 100 100 C 125 125 175 125 200 100" />
  </defs>
  <text>
    <textPath xlink:href="#MyPath">Text on path</textPath>
  </text>
</svg>
```

[Download this example](#)

The **startOffset** attribute defines the distance from the start of the path where the first character should be positioned.

Various CSS properties control aspects of text drawn along a path. The standard set of text and font properties from CSS2 apply, including '**text-align**' and '**vertical-align**'. Additional, the following additional properties control how the text is formatted and rendered:

- **orient-to-path**: Possible values are *true* and *false*, with a default of true. If true, the glyph/symbol is rotated to have its coordinate system line up with the tangent of the curve.
- **textPath-transform**: which defines a transformation that should be applied (conceptually) after the given glyph/symbol is properly sized, placed and oriented by all of the other CSS text properties. The options for this property are the same as the **transform** attribute (??? add link).

Text on a path opens the possibility of significant implementation differences due to different methods of calculating distance along a path. In order to ensure that distance calculations are sufficiently precise, the following two attributes are available on <path> elements. (??? Obviously, this section needs to be moved to the <path> element section.)

- **flatness**= A distance measurement in either local coordinates or in CSS units which serves as a hint for the allowable error tolerance allowable in approximating a curve with a series line segments. Commercial implementations should honor this attribute.
- **nominalLength**= The distance measurement (A) for the given <path> element computed at authoring time. The SVG user agent should compute its own distance measurement (B). The SVG user agent should then scale all distance-along-a-curve computations by A divided by B.

(??? Insert drawings here)

12.7.1 Vector graphics along a path

SVG's text on a path features set has been generalized to allow for arbitrary SVG along a path, by adding the **use** element as a valid child of **text**.

13 Fonts

Contents

- [13.1 Introduction](#)
- [13.2 SVG fonts](#)
 - [13.2.1 Overview of SVG fonts](#)
 - [13.2.2 The element](#)
 - [13.2.3 The <glyph> element](#)
 - [13.2.4 The <missingGlyph> element](#)
 - [13.2.5 The <kern> element](#)

13.1 Introduction

Reliable delivery of fonts is considered a critical requirement for SVG. Designers should be able to create SVG graphics with whatever fonts they care to use and then the same fonts should 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, subsetting web fonts for all text elements included in a given SVG document. Typically, the web fonts will be saved in a nearby location to the SVG document itself.

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. See [SVG Fonts](#) for more information.

13.2 SVG fonts

13.2.1 Overview of SVG fonts

An SVG font is a font defined using SVG's [](#) element.

The purpose of SVG fonts is to allow for delivery of glyph outlines in display-only environments. SVG fonts that accompany web pages should be supported only in browsing and viewing situations. Graphics editing applications or file translation tools should not attempt convert SVG fonts into system fonts. The intent is that SVG files should be interchangeable between two content creators, but not the SVG fonts that 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 [fullFontName](#) 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 may 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](#)].

An SVG font can be either embedded within the SVG document 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-19990812.dtd'>
  <defs>
    <font id="MyFont" fullFontName="Super Sans" >
      <glyph unicode="33"><path d="..."></glyph>
      <glyph unicode="34"><path d="..."></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-19990812.dtd'>
  <defs>
    <font id="MyFont" fullFontName="Super Sans" >
      <glyph unicode="33"><path d="..."></glyph>
      <glyph unicode="34"><path d="..."></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-19990812.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](#)

13.2.2 The element

The element defines an SVG font.

```

<!ELEMENT font (desc?,title?,missingGlyph,(glyph|kern)*) >
<!ATTLIST font
  id ID #IMPLIED
  fontStyle CDATA #IMPLIED
  fontVariant CDATA #IMPLIED
  fontWeight CDATA #IMPLIED
  fontStretch CDATA #IMPLIED
  unicodeRange CDATA #IMPLIED
  unitsPerEm CDATA #REQUIRED
  panose1 CDATA #IMPLIED
  slope CDATA #IMPLIED
  capHeight CDATA #REQUIRED
  xHeight CDATA #REQUIRED
  accentHeight CDATA #IMPLIED
  ascent CDATA #REQUIRED
  descent CDATA #REQUIRED
  horizOriginX CDATA #IMPLIED
  horizOriginY CDATA #IMPLIED
  horizAdvX CDATA #IMPLIED
  vertOriginX CDATA #IMPLIED
  vertOriginY CDATA #IMPLIED
  vertAdvY CDATA #IMPLIED
  bbox CDATA #REQUIRED
  baseline CDATA #REQUIRED
  centerline CDATA #REQUIRED
  mathline CDATA #REQUIRED
  topline CDATA #REQUIRED
  fullFontName CDATA #IMPLIED
  underlinePosition CDATA #IMPLIED
  underlineThickness CDATA #IMPLIED
  strikethroughPosition CDATA #IMPLIED
  strikethroughThickness CDATA #IMPLIED
  overlinePosition CDATA #IMPLIED
  overlineThickness CDATA #IMPLIED >

```

Attribute definitions:

fontStyle = "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.

fontVariant = "[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.

fontWeight = "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.

fontStretch = "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.

unicodeRange = "<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.

unitsPerEm = "<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](#)].

panose1 = "[<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.

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.

capHeight = "<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](#)].

xHeight = "<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](#)].

accentHeight = "<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.

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](#)].

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](#)].

horizOriginX = "<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.

horizOriginY = "<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.

horizAdvX = "<number>"

The default horizontal advance after rendering a glyph in horizontal orientation. The default value is the width of the font's [bbox](#).

vertOriginX = "<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 the center position within the font's [bbox](#).

vertOriginY = "<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.

vertAdvY = "<number>"

The default vertical advance after rendering a glyph in vertical orientation. The default value is the height of the font's [bbox](#).

bbbox = "<number>, <number>, <number>, <number>"

The maximum bounding box of the font within the font coordinate system. The value is a comma-separated list of exactly four numbers specifying, in order, the minimum x, minimum y, maximum width and maximum height. (Note that these four parameters differ from the 'bbox' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)], which calls for lower left x, lower left y, upper right x and upper right y, instead.) For more information, see the description of the 'bbox' descriptor in the "Cascading Style Sheets (CSS) level 2" specification [[CSS2](#)].

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](#)].

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](#)].

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](#)].

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](#)].

fullFontName = "<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](#)].

underlinePosition = "<number>"

The ideal position of an underline within the font coordinate system.

underlineThickness = "<number>"

The ideal thickness of an underline, expressed as a length within the font coordinate system.

strikethroughPosition = "<number>"

The ideal position of a strike-through within the font coordinate system.

strikethroughThickness = "<number>"

The ideal thickness of a strike-through, expressed as a length within the font coordinate system.

overlinePosition = "<number>"

The ideal position of an overline within the font coordinate system.

overlineThickness = "<number>"

The ideal thickness of an overline, expressed as a length within the font coordinate system.

Attributes defined elsewhere:

[id](#).

13.2.3 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 [](#) element.

The contents of a <glyph> can be any SVG graphics elements. However, in some implementations, faster font rendering (and possibly improved quality) may occur when glyph definitions consist of a single [<path>](#) element.

```

<!ELEMENT glyph ( (defs?, desc?, title?,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|g|switch)* ) )>
<!ATTLIST glyph
  unicode CDATA #REQUIRED
  glyphName CDATA #IMPLIED
  vertTextOrient CDATA #IMPLIED
  arabic CDATA #IMPLIED
  han CDATA #IMPLIED
  horizAdvX CDATA #IMPLIED
  vertAdvY CDATA #IMPLIED
  bbox CDATA #IMPLIED >

```

Attribute definitions:

unicode = "<number> [, <number>]* "

One or more decimal numbers indicating the sequence of Unicode characters which corresponds to this glyph. If a single decimal number is provided, then this glyph corresponds to the given Unicode character number. If a list of numbers 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="102 102 108", which corresponds to the letters "f", "f", and "l", then the given glyph will be used to render the sequence of characters "ffl". When determining the glyph to draw a given character sequence, longer Unicode number list matches take precedence over shorter list matches.

glyphName = "<name> [, <name>]* "

One or more strings (without white space or commas) which represent names for the glyph. The glyph names can be used in situations where Unicode character numbers do not provide sufficient information to access the correct glyph.

vertTextOrient = "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 should determine the orientation of this glyph.

arabic = "initial | medial | terminal | standard"

For Arabic glyphs, indicates which of the four possible forms this glyph represents.

han = "ja | zht | zhs | kor"

For glyphs in the Han range, indicates which of the four possible forms this glyph represents.

horizAdvX = "<number>"

The horizontal advance after rendering a glyph in horizontal orientation. The default value is the value of the font's [horizAdvX](#) attribute.

vertAdvY = "<number>"

The vertical advance after rendering a glyph in vertical orientation. The default value is the value of the font's [vertAdvY](#) attribute.

bbox = "<number>, <number>, <number>, <number>"

The bounding box of the glyph within the font coordinate system. The value is a comma-separated list of exactly four numbers specifying, in order, the minimum x, minimum y, maximum width and maximum height. The default value is the value of the font's [bbox](#) attribute.

13.2.4 The <missingGlyph> element

The <missingGlyph> 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 <missingGlyph> element have the same meaning as the corresponding attributes on the [<glyph>](#) element.

```
<!ELEMENT missingGlyph (defs?,desc?,title?,
                        (path|text|rect|circle|ellipse|line|polyline|polygon|
                         use|image|g|switch)* )>
<!ATTLIST missingGlyph
  horizAdvX CDATA #IMPLIED
  vertAdvY CDATA #IMPLIED
  bbox CDATA #IMPLIED >
```

Attributes defined elsewhere:

[horizAdvX](#), [vertAdvY](#), [bbox](#).

13.2.5 The <kern> element

The <kern> element defines kerning pairs and adjustment values in the horizontal advance value when drawing pairs of glyphs together horizontally. The spacing between characters is reduced by the kerning adjustment.

```
<!ELEMENT kern EMPTY >
<!ATTLIST kern
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >
```

Attribute definitions:

u1 = "[<number> | <urange>] [, [<number> | <urange>]]* "

One or more decimal numbers and/or unicode ranges identifying the characters which can be the first character in this kerning adjustment definition.

g1 = "<name> [, <name>]* "

One or more glyph names identifying the characters which can be the first character in this kerning adjustment definition.

u2 = "[<number> | <urange>] [, [<number> | <urange>]]* "

One or more decimal numbers and/or unicode ranges identifying the characters which can be the second character in this kerning adjustment definition.

g2 = "<name> [, <name>]* "

One or more glyph names identifying the characters which can be the second character in this kerning adjustment definition.

k = "<number>"

The amount to decrease the spacing between the two glyphs in the kerning pair. The value is in the font coordinate system. [bbox](#) attribute.

At least one each of u_1 or g_1 and at least one of u_2 or g_2 must be provided.

14 Filter Effects

Contents

- [14.1 Introduction](#)
- [14.2 Background](#)
- [14.3 Basic Model](#)
- [14.4 Defining and Invoking a Filter Effect](#)
- [14.5 Filter Effects Region](#)
- [14.6 Common Attributes](#)
- [14.7 Accessing the background image](#)
- [14.8 Filter Processing Nodes](#)

14.1 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.

14.2 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.

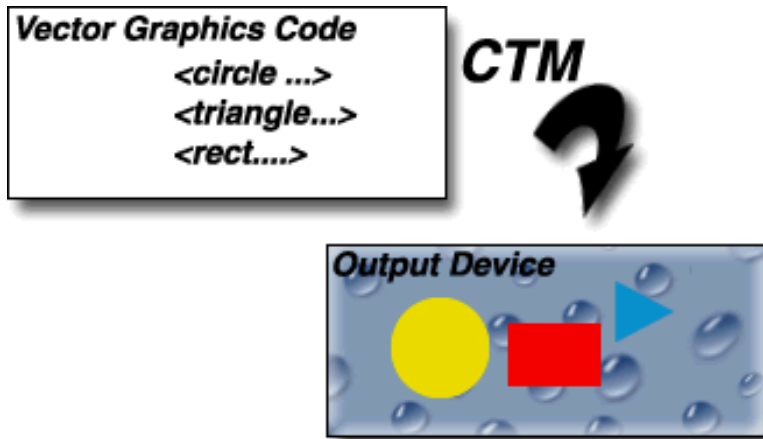
14.3 Basic Model

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:

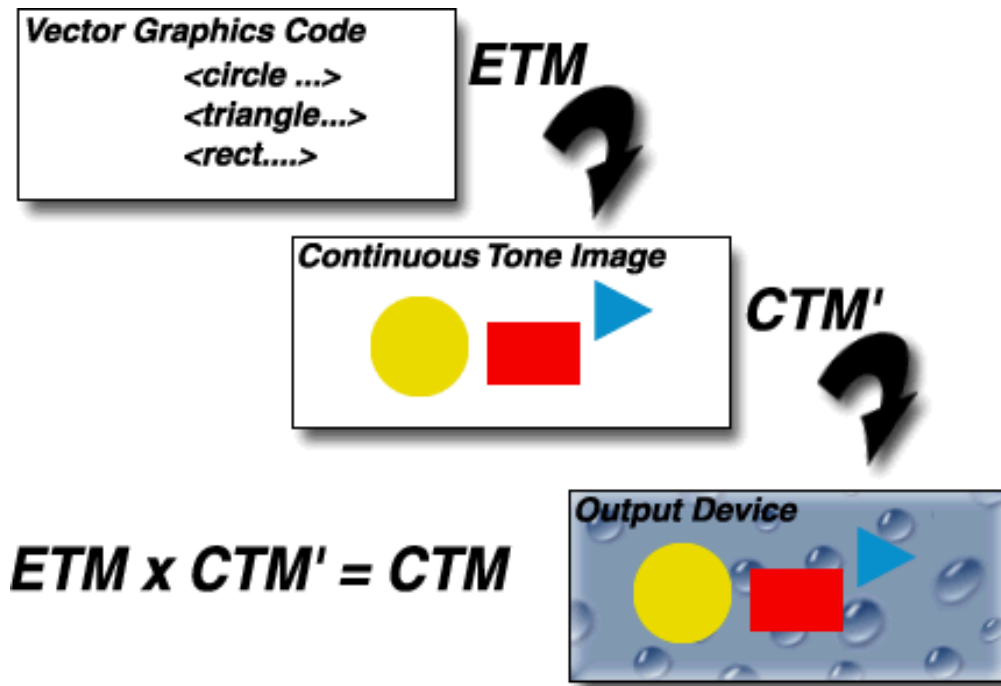
Traditional 2D graphics pipeline



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:

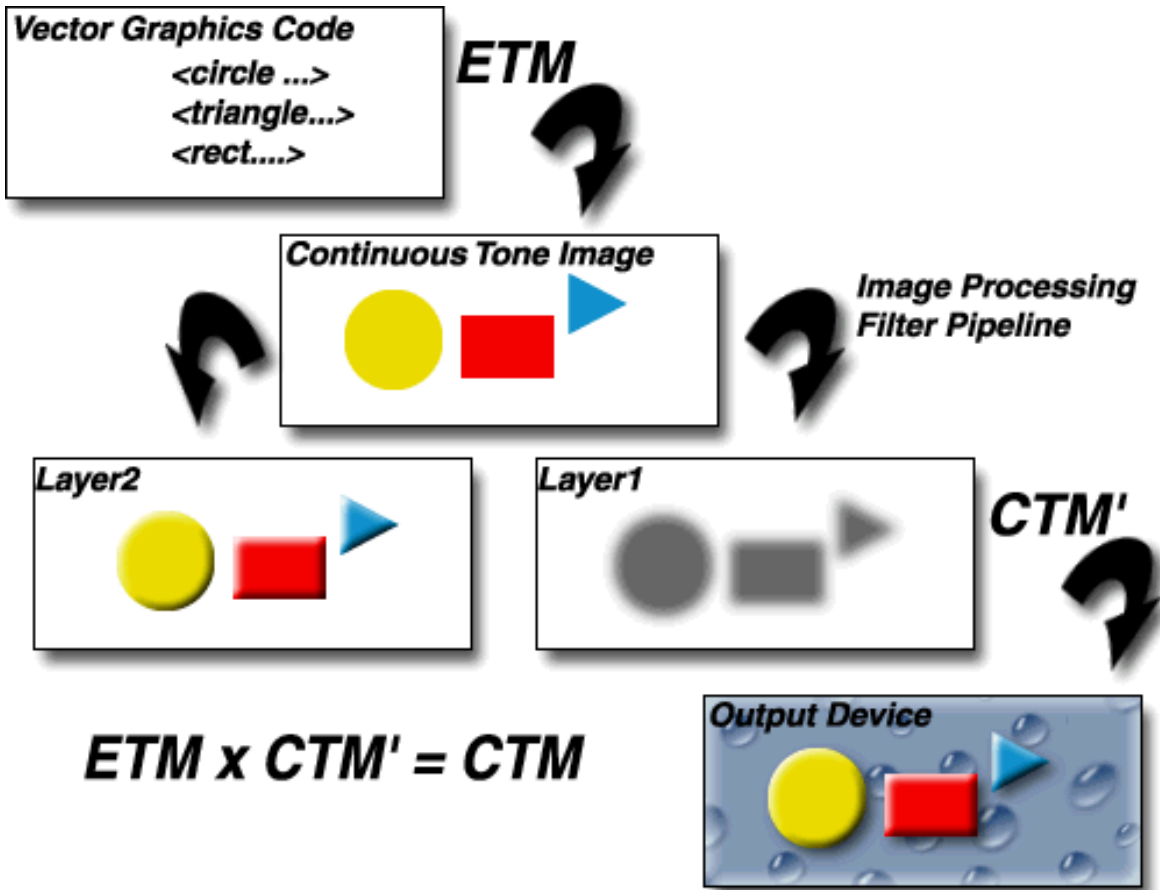
Rendering via Continuous Tone Intermediate Image



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.

14.4 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 August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.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 should be 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 should be 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. Here is an example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
"http://www.w3.org/Graphics/SVG/SVG-19990812.dtd">
<filter id="Shadow">
  <feGaussianBlur      in="SourceAlpha"
                      radius="3"
                      nodeId="blurredAlpha" />
  <feOffset            in="blurredAlpha"
                      dx="2" dy="1"
                      nodeId="offsetBlurredAlpha" />
  <feDiffuseLighting  in="blurredAlpha"
                      diffuseConstant=".5"
                      nodeId="bumpMapDiffuse" >
    <feDistantLight/>
  </feDiffuseLighting>
  <feComposite        in="bumpMapDiffuse" in2="SourcePaint"
                      operator="arithmetic" k1="1"
                      nodeId="litPaint" />
  <feSpecularLighting in="blurredAlpha"
                      specularConstant=".5"
                      specularExponent="10"
                      lightColor="feDistantLight"
                      nodeId="bumpMapSpecular" >
    <feDistantLight/>
  </feSpecularLighting>
  <feComposite        in="litPaint" in2="bumpMapSpecular"
                      operator="arithmetic" k2="1" k3="1"
                      nodeId="litPaint" />
  <feComposite        in="litPaint"
                      in2="SourceAlpha"
                      mode="AinB"
                      nodeId="litPaint" />
  <feMerge>
    <feMergeNode in="litPaint" />
    <feMergeNode in="offsetBlurredAlpha" />
  </feMerge>
</filter>

<text style="font-size:36; fill:red; filter:url(#Shadow)"
      x="10" y="250">Shadowed Text</text>
</svg>
```

[Download this example](#)

For most processing nodes, the **in** (and sometimes **in2**) attribute identifies the graphics which serve as input and the **nodeId** 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 should be used as input to the processing node.
- A reference to an earlier **nodeId**. In the example above, the `<feOffset>` processing node refers to the most recent processing node whose `nodeId` 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 **nodeId** 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 **nodeId** and **in** did not have to be provided.

Filters *do not* use XML IDs for **nodeIds**; instead, **nodeIds** can be any arbitrary attribute string value. **nodeIds** only are meaningful within a given `<filter>` element and thus have only local scope. If a `nodeId` appears multiple times within a given `<filter>` element, then a reference to that `nodeId` will use the closest preceding processing node with the given `nodeId`. Forward references to `nodeIds` are invalid.

14.5 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** | **objectBoundingBox** }. Defines the coordinate system for attributes **x**, **y**, **width**, **height**. If **filterUnits**="userSpace" (the default), **x**, **y**, **width**, **height** represent values in the current user coordinate system in place at the time when the <filter> element is defined. If **filterUnits**="objectBoundingBox", then **x**, **y**, **width**, **height** 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]` and 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, should be the default. For printing, an appropriate common printer resolution, such as 400dpi, should be 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="110%" height="110%"`.

14.6 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):

	<p><i>nodeId</i> 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.</p> <p><i>in</i> If supplied, then indicates that this processing node should use either the output of previous node as input or use some standard keyword to specify alternate input. (For the first processing node, the default <i>in</i> is SourceGraphic.) Available keywords representing pseudo image inputs include:</p> <ul style="list-style-type: none">● 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 may need to rasterize the graphics elements
--	---

Common Attributes

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).

14.7 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 documents 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>)]
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 should manage 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 should be established and that (in effect) all children of the current [container element](#) should 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 should proceed without interruption (i.e., no error message) and a fully transparent image should 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.

14.8 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.

NodeType	feBlend												
Processing Node-Specific Attributes	<i>mode</i> , One of the image blending modes (see table below). Default is: normal <i>in2</i> , The second image ("B" in the formulas) for the compositing operation.												
Description	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.												
Implementation Notes	<p>The compositing formula, expressed using premultiplied colors:</p> $qr = 1 - (1-qa)*(1-qb)$ $cr = (1-qa)*cb + (1-qb)*ca + qa*qb*(Blend(ca/qa,cb/qb))$ <p>where:</p> <ul style="list-style-type: none"> 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 <p>The following table provides the list of available image blending modes:</p> <table border="1"> <thead> <tr> <th>Image Blending Mode</th> <th>Blend() function</th> </tr> </thead> <tbody> <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> </tbody> </table>	Image Blending Mode	Blend() function	normal	Ca	multiply	(ca/qa)*(cb/qb)	screen	1-(1-(ca/qa))*(1-(cb/qb))	darken	(to be provided later)	lighten	(to be provided later)
Image Blending Mode	Blend() function												
normal	Ca												
multiply	(ca/qa)*(cb/qb)												
screen	1-(1-(ca/qa))*(1-(cb/qb))												
darken	(to be provided later)												
lighten	(to be provided later)												

NodeType	
-----------------	--

feColorMatrix

Processing Node-Specific Attributes

type, string (one of: matrix, saturate, hue-rotate, luminance-to-alpha)

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:

```
type="matrix"  
values="1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0"
```

- For saturate: one real number value (0 to 1) or one percentage value (e.g., 50%)
- For hue-rotate: one real number value (degrees)
- Not applicable for luminance-to-alpha

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 |
```

hue-rotate 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:

```
| a11 a12 a13 |      | [+0.213 +0.715 +0.072]  
| a21 a22 a13 | =    | [+0.213 +0.715 +0.072] +  
| a11 a12 a13 |      | [+0.213 +0.715 +0.072]
```

```
cos(hue-rotate value) * [+0.787 -0.715 -0.072]  
                        [-0.212 +0.285 -0.072] +  
                        [-0.213 -0.715 +0.928]
```

```
sin(hue-rotate value) * [-0.213 -0.715+0.928]  
                        [+0.143 +0.140-0.283]  
                        [-0.787 +0.715+0.072]
```

	<p>Thus, the upper left term of the hue matrix turns out to be:</p> $.213 + \cos(\text{hue-rotate value}) \cdot .787 - \sin(\text{hue-rotate value}) \cdot .213$ <p>luminance-to-alpha</p> $\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.299 & 0.587 & 0.114 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$
Implementation issues	<p>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.</p>

NodeType	feComponentTransfer
Processing Node-Specific Attributes	None.
Processing Node-Specific Sub-Elements	<p>Each <feComponentTransfer> element needs to have at most one each of the following sub-elements, each of which is an empty element:</p> <ul style="list-style-type: none"> <feFuncR>, transfer function for red component <feFuncG>, transfer function for green component <feFuncB>, transfer function for blue component <feFuncA>, transfer function for alpha component <p>Each of these sub-elements (i.e., <feFuncR>, <feFuncG>, <feFuncB>, <feFuncA>) can have the following attributes:</p> <p>Common parameters to all transfer modes:</p> <ul style="list-style-type: none"> <i>type</i>, string (one of: identity, table, linear, gamma) <p>Parameters specific to particular transfer modes:</p> <p>For <i>table</i>:</p> <ul style="list-style-type: none"> <i>tableValues</i>, list of real number values v_0, v_1, \dots, v_n. <p>For <i>linear</i>:</p> <ul style="list-style-type: none"> <i>slope</i>, real number value giving slope of linear equation. <i>intercept</i>, real number value giving Y-intercept of linear equation. <p>For <i>gamma</i> (see description below for descriptions):</p> <ul style="list-style-type: none"> <i>amplitude</i>, real number value. <i>exponent</i>, real number value. <i>offset</i>, real number value.

Description	<p>This filter performs component-wise remapping of data as follows:</p> <pre>R' = feFuncR(R) G' = feFuncG(G) B' = feFuncB(B) A' = feFuncA(A)</pre> <p>for every pixel. The RGBA and R'G'B'A' values are automatically <i>non-premultiplied</i> temporarily for this operation.</p> <p>When <i>type="table"</i>, the transfer function consists of a linearly interpolated lookup table.</p> $k/N \leq C < (k+1)/N \Rightarrow C' = vk + (C - k/N)*N * (vk+1 - vk)$ <p>When <i>type="linear"</i>, the transfer function consists of a linear function describes by the following equation:</p> $C' = slope*C + offset$ <p>When <i>type="gamma"</i>, the transfer function consists of the following equation:</p> $C' = amplitude*pow(C, exponent) + offset$
Comments	<p>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 transfer, sine-wave, etc.</p>
Implementation issues	<p>Similar to the feColorMatrix filter, the undoing and redoing of the premultiplication can be avoided if feFuncA is the identity transform and A = 1.</p>

NodeType	feComposite
Processing Node-Specific Attributes	<p><i>operator</i>, one of (<i>over</i>, <i>in</i>, <i>out</i>, <i>atop</i>, <i>xor</i>, <i>arithmetic</i>). Default is: <i>over</i>.</p> <p><i>arithmetic-constants</i>, k1,k2,k3,k4</p> <p><i>in2</i>, The second image ("B" in the formulas) for the compositing operation.</p>
Description	<p>This filter performs the combination of the two input images pixel-wise in image space.</p> <p><i>over</i>, <i>in</i>, <i>atop</i>, <i>out</i>, <i>xor</i> use the Porter-Duff compositing operations.</p> <p>For these operations, the extent of the resulting image can be affected.</p> <p>In other words, even if two images do not overlap in image space, the extent for <i>over</i> will essentially include the union of the extents of the two input images.</p> <p><i>arithmetic</i> evaluates $k1*i1*i2 + k2*i1 + k3*i2 + k4$, using componentwise arithmetic with the result clamped between [0..1].</p>
Comments	<p><i>arithmetic</i> are useful for combining the output from the feDiffuseLighting and feSpecularLighting filters with texture data. <i>arithmetic</i> is also useful for implementing <i>dissolve</i>.</p>

NodeType	feDiffuseLighting
Processing Node-Specific Attributes	<p><i>resultScale</i> (Multiplicative scale for the result. This allows the result of the feDiffuseLighting node to represent values greater than 1)</p> <p><i>surfaceScale</i> height of surface when $A_{in} = 1$.</p> <p><i>diffuseConstant</i> k_d in Phong lighting model. Range 0.0 to 1.0</p> <p><i>lightColor</i> RGB</p>

Processing Node-Specific Sub-Elements	<p>One of</p> <pre> <feDistantLight azimuth= elevation= > <fePointLight x= y= z= > <feSpotLight x= y= z= pointsAtX= pointsAtY= pointsAtZ= specularExponent=> </pre>
Description	<p>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 calculation follows the standard diffuse component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map. Color or texture is meant to be applied via a multiply (mul) composite operation.</p> <pre> Dr = (kd * N.L * Lr) / resultScale Dg = (kd * N.L * Lg) / resultScale Db = (kd * N.L * Lb) / resultScale Da = 1.0 / resultScale </pre> <p>where</p> <p>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</p> <p>N is a function of x and y and depends on the surface gradient as follows:</p> <p>The surface described by the input alpha image $A_{in}(x,y)$ is:</p> $Z(x,y) = \text{surfaceScale} * A_{in}(x,y)$ <p>Surface normal is calculated using the Sobel gradient 3x3 filter:</p> <pre> 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 </pre> $N = (N_x, N_y, N_z) / \text{Norm}(N_x, N_y, N_z)$ <p>L, the unit vector from the image sample to the light is calculated as follows:</p> <p>For Infinite light sources it is constant:</p> <pre> Lx = cos(azimuth)*cos(elevation) Ly = -sin(azimuth)*cos(elevation) Lz = sin(elevation) </pre> <p>For Point and spot lights it is a function of position:</p> <pre> Lx = Lightx - x Ly = Lighty - y Lz = Lightz - Z(x,y) </pre> $L = (L_x, L_y, L_z) / \text{Norm}(L_x, L_y, L_z)$ <p>where Lightx, Lighty, and Lightz are the input light position.</p> <p>Lr,Lg,Lb, the light color vector is a function of position in the spot light case only:</p> <pre> Lr = Lightr*pow((-L.S), specularExponent) Lg = Lightg*pow((-L.S), specularExponent) Lb = Lightb*pow((-L.S), specularExponent) </pre>

	<p>where S is the unit vector pointing from the light to the point (pointsAtX, pointsAtY, pointsAtZ) in the x-y plane:</p> $S_x = \text{pointsAtX} - \text{Lightx}$ $S_y = \text{pointsAtY} - \text{Lighty}$ $S_z = \text{pointsAtZ} - \text{Lightz}$ $S = (S_x, S_y, S_z) / \text{Norm}(S_x, S_y, S_z)$ <p>If $L \cdot S$ is positive no light is present. ($L_r = L_g = L_b = 0$)</p>
Comments	This filter produces a light map, which can be combined with a texture image using the multiply term of the <i>arithmetic</i> <Composite> 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	<i>scale</i> <i>xChannelSelector</i> one of R,G,B or A. <i>yChannelSelector</i> one of R,G,B or A <i>in2</i> , The second image ("B" in the formulas) for the compositing operation.
Description	<p>Uses Input2 to spatially displace Input1, (similar to the Photoshop displacement filter). This is the transformation to be performed:</p> $P'(x,y) \leftarrow P(x + \text{scale} * ((XC(x,y) - .5), y + \text{scale} * (YC(x,y) - .5))$ <p>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 <i>xChannelSelector</i> and <i>yChannelSelector</i>. 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 <i>xChannelSelector</i> to "R" and <i>yChannelSelector</i> to "G".</p>
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 <i>scale</i> which represents the maximum displacement in either x or y.

NodeType	feFlood
Processing Node-Specific Attributes	<i>style</i> , 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 <i>color</i>

NodeType	feGaussianBlur
Processing Node-Specific Attributes	<i>stdDeviation</i> .

Description	<p>Perform gaussian blur on the input image.</p> <p>The Gaussian blur kernel is an approximation of the normalized convolution: $H(x) = \exp(-x^2 / (2s^2)) / \sqrt{2 * \pi * s^2}$ where 's' is the standard deviation specified by stdDeviation.</p> <p>This can be implemented as a separable convolution.</p> <p>For larger values of 's' ($s \geq 2.0$), 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%.</p> <pre>let d = floor(s * 3*sqrt(2*pi)/4 + 0.5)</pre> <p>... if d is odd, use three box-blurs of size 'd', centered on the output pixel.</p> <p>... 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.</p>
Implementation Issues	<p>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.</p>

NodeType	feImage
Processing Node-Specific Attributes	<p><i>href</i>, reference to external image data. <i>transform</i>, supplemental transformation specification</p>
Description	<p>Refers to an external image which is loaded or rendered into an RGBA raster. If <i>imaging-matrix</i> is not specified, the image takes on its natural width and height and is positioned at 0,0 in image space.</p> <p>The <i>imageref</i> 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 <i>SourceGraphic</i> except from an external source.</p>

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 <i>over</i> operator with <i>Input1</i> on the bottom and the last specified input, <i>InputN</i> , 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 may 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	<i>operator</i> , one of <i>erode</i> or <i>dilate</i> . <i>radius</i> , 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, <i>SourceAlpha</i> . 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	<i>dx,dy</i>
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	<i>surfaceScale</i> height of surface when $A_{in} = 1$. <i>specularConstant</i> k_s in Phong lighting model. Range 0.0 to 1.0 <i>specularExponent</i> exponent for specular term, larger is more "shiny". Range 1.0 to 128.0. <i>lightColor</i> RGB
Processing Node-Specific Sub-Elements	One of <pre> <feDistantLight azimuth= elevation= > <fePointLight x= y= z= > <feSpotLight x= y= z= pointsAtX= pointsAtY= pointsAtZ= specularExponent=> </pre>
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 calculation follows the standard specular component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map. The result of the lighting calculation is added. 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. $S_r = k_s * \text{pow}(N.H, \text{specularExponent}) * L_r$ $S_g = k_s * \text{pow}(N.H, \text{specularExponent}) * L_g$ $S_b = k_s * \text{pow}(N.H, \text{specularExponent}) * L_b$ $S_a = \max(S_r, S_g, S_b)$ where k_s = specular lighting constant N = surface normal unit vector, a function of x and y H = "halfway" unit vector between eye unit vector and light unit vector L_r, L_g, L_b = RGB components of light See <i>feDiffuseLighting</i> for definition of N and (L_r, L_g, L_b) .

	<p>The definition of H reflects our assumption of the constant eye vector $E = (0,0,1)$:</p> $H = (L + E) / \text{Norm}(L+E)$ <p>where L is the light unit vector.</p> <p>Unlike the <code>feDiffuseLighting</code>, the <code>feSpecularLighting</code> filter produces a non-opaque image. This is due to the fact that specular result (S_r, S_g, S_b, S_a) is meant to be added to the textured image. The alpha channel of the result is the max of the color components, so that where the specular light is zero, no additional coverage is added to the image and a fully white highlight will add opacity.</p>
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 <i>add</i> term of the <i>arithmetic</i> 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 <code>feDiffuseLighting</code> and <code>feSpecularLighting</code> 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	<i>baseFrequency</i> <i>numOctaves</i> <i>type</i> , one of <i>fractalNoise</i> or <i>turbulence</i> .
Description	<p>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 of the Perlin turbulence-function, see "Texturing and Modeling", Ebert et al, AP Professional, 1994.</p> <p>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.</p>
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 Interactivity

Contents

- [15.1 Introduction](#)
- [15.2 Zoom and pan control: the allowZoomAndPan attribute on the <svg> element](#)
- [15.3 Cursors](#)
 - [15.3.1 Introduction to cursors](#)
 - [15.3.2 The 'cursor' property](#)
 - [15.3.3 The <cursor> element](#)

15.1 Introduction

(Introduction to the various flavors of interactivity in SVG, zoom and pan of drawings, event handlers such as onclick in JavaScript, event-driven animations, hyperlinks and cursors.)

15.2 Zoom and pan control: the allowZoomAndPan attribute on the <svg> element

The outermost <svg> element in an SVG document can have the optional attribute allowZoomAndPan, which takes the possible values of *true* and *false*, with the default being *true*. If true, the user agent should allow the user to zoom in, zoom out and pan around the given document. If false, the user agent should not allow the user to zoom and pan on the given document. If a allowZoomAndPan attribute is assigned to an inner <svg> element, the allowZoomAndPan setting on the inner <svg> element will be ignored.

15.3 Cursors

15.3.1 Introduction to cursors

Some interactive display environments provide the ability to modify the appearance of the pointer, which is also known as the *cursor*. Two types of cursors are available:

- Standard built-in cursors

- Custom cursors

The `'cursor'` property is used to specify which cursor to use. Custom cursors are defined with a `<cursor>` element.

15.3.2 The 'cursor' property

'cursor'

<i>Value:</i>	[[<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
<i>Initial:</i>	auto
<i>Applies to:</i>	all elements
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	visual, interactive

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 may be selected. Often rendered as an I-bar.

wait

Indicates that the program is busy and the user should wait. 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 should 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.

15.3.3 The <cursor> element

```
<!ELEMENT cursor (desc?,title?) >
<!ATTLIST cursor
  id ID #IMPLIED
  system-required CDATA #IMPLIED
  x CDATA "0"
  y CDATA "0"
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >
```

Attribute definitions:

x = "x-coordinate"

The *x-coordinate* of the position in the cursor's coordinate system which represents the precise position that is being pointed to.

y = "y-coordinate"

The *y-coordinate* of the position in the cursor's coordinate system which represents the precise position that is being pointed to.

href = "uri-reference"

A [URI reference](#) to the file or element which provides the image of the cursor.

Attributes defined elsewhere:

[id](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#).

SVG user agents are required to support PNG format images as targets of the xlink:href property.

16 Linking

Contents

- [16.1 Links out of SVG documents: the <a> element](#)
- [16.2 Linking into SVG documents: URI fragments and SVG views](#)
 - [16.2.1 Introduction: URI fragments and SVG views](#)
 - [16.2.2 SVG fragment identifiers](#)
 - [16.2.3 Predefined views: the <view> element](#)

16.1 Links out of SVG documents: 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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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-19990812.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.

```

<!ELEMENT a (defs?,desc?,title?,
             (path|text|rect|circle|ellipse|line|polyline|polygon|
              use|image|svg|g|switch|a)*)>
<!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|parsed|replace) #FIXED 'replace'
  xlink:actuate (user|auto) #FIXED 'user'
  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].

`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].

`xlink:role = '<string>'`

A generic string used to describe the function of the link's content. Refer to the "XML Linking Language (XLink)" [XLink].

`xlink:title = '<string>'`

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

`xlink:show = 'replace'`

Indicates that upon activation of the link the referenced document should replace the entire contents of the current document. Refer to the "XML Linking Language (XLink)" [XLink].

`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].

`xlink:href = "<URI-reference>"`

The location of the referenced object, expressed as a <URI-reference>. Refer to the "XML

Linking Language (XLink)" [\[XLink\]](#).

16.2 Linking into SVG documents: URI fragments and SVG views

16.2.1 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 documents 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.

16.2.2 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(...)`, `allowZoomAndPan(...)` 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 |
                   allowZoomAndPanSpec |
                   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 ::= 'allowZoomAndPanSpec(' 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 allowZoomAndPanSpec corresponds to to the parameter values for the [allowZoomAndPan](#) attribute on the `<svg>` element. For example, allowZoomAndPan(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 should be used to separate numeric values within an SVG view specification (e.g., #svgView(viewBox(0,0,200,200))) and semicolons should be 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., `` 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.

16.2.3 Predefined views: the `<view>` element

The `<view>` element is defined as follows:

```
<!ELEMENT view (desc?,title?)
<!ATTLIST view
  id ID #IMPLIED
  viewBox CDATA #IMPLIED
  preserveAspectRatio CDATA 'xMidYMid meet'
  allowZoomAndPan (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) should be highlighted.

Attributes defined elsewhere:

[class](#), [viewBox](#), [preserveAspectRatio](#), [allowZoomAndPan](#).

17 Scripting

Contents

- [17.1 Specifying the scripting language](#)
 - [17.1.1 Specifying the default scripting language](#)
 - [17.1.2 Local declaration of a scripting language](#)
- [17.2 The <script> element](#)

17.1 Specifying the scripting language

17.1.1 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](#)].

In the absence of a `contentScriptType` attribute, the default can be set by a "Content-Script-Type" HTTP header:

`Content-Script-Type: <contentType>`

User agents should determine the default scripting language for an SVG document 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 should set the default scripting language to "text/ecmascript".

17.1.2 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.

17.2 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 SVG 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 August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >
```

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](#).

Attributes defined elsewhere:

[xml:link](#), [show](#), [actuate](#), [href](#).

17.3 Event Handling

Any `<g>`, `<image>`, `<path>`, `<text>` or vector graphic shape (such as a `<rect>`) can be assigned any of the following standard HTML event handlers:

Mouse Events

- **onmousedown**
- **onmouseup**
- **onclick**
- **ondblclick**
- **onmouseover**
- **onmousemove**
- **onmouseout**

Keyboard Events

- **onkeydown**
- **onkeypress**
- **onkeyup**

State Change Events

- **onload**
- **onerror** (corresponds to DOM level 2 error event)
- **onabort** (corresponds to DOM level 2 abort 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)
- **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.)

Additionally, SVG's scripting engine needs to have the *altKey*, *ctrlKey* and *shiftKey* properties available.

18 Animation

Contents

- [18.1 Introduction](#)
- [18.2 Animation elements](#)
 - [18.2.1 Introduction](#)
 - [18.2.2 The <animate> element](#)
 - [18.2.3 The <animateMotion> element](#)
 - [18.2.4 The <animateTransform> element](#)
 - [18.2.5 The <animateColor> element](#)
 - [18.2.6 The <animateFlipbook> element](#)
- [18.3 Animation example using the SVG DOM](#)

18.1 Introduction

The Web is a dynamic medium and that SVG needs to support the ability to change vector graphics over time. SVG documents can be animated in the following ways:

- Using SVG's [Animation Elements](#). An SVG document 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 [Document Object Model \(DOM\) Level 1 Specification](#) and [Document Object Model \(DOM\) Level 2 Specification](#). 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 documents can be components within [SMIL \(Synchronized Multimedia Integration Language\)](#) documents.
- 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 such as XHTML to achieve animation effects.

18.2 Animation elements

18.2.1 Introduction

The animation elements in SVG were developed in collaboration with the W3C Synchronized Multimedia (SYMM) Working Group, developers of the Synchronized Multimedia Integration Language (SMIL) 1.0 Specification [[SMIL1](#)], with an attempt at compatibility with SYMM work-in-progress efforts towards formulating SMIL 2.0 and general animation facilities for XML grammars such as XHTML [[XHTML10](#)].

18.2.2 The <animate> element

The <animate> element is used to animate a single XML attribute or property over time. For example, to make a rectangle repeatedly fade away over 5 seconds, you can specify (??? better examples will come later):

```
<rect id="MyRect" ...>
<animate href="#foo"
        attributeType="css" attribute="opacity"
        from="1" to="0" dur="5s" repeatCount="indefinite" />
```

```
<!ELEMENT animate (desc?,title?) >
<!ATTLIST animate
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  attribute CDATA #REQUIRED
  attributeType (xml|css) 'xml'
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  additive (true | false) "false"
  accumulate (true | false) "false"
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"
  vtimes CDATA #IMPLIED
  interpSpline CDATA #IMPLIED >
```

Attribute definitions:

xlink:href = "*uri-reference*"

A [URI reference](#) to the element which is the target of this animation and which therefore will have one of its attributes modified over time. The target element must be part of the current SVG document.

attribute = "<attributeName>"

The name of the attribute or property which is to be animated.

attributeType = "xml | css"

Indicate the type of the attribute or property. A value of attributeType="xml" indicates the attribute represents the name of an XML attribute on the element referenced by xlink:href. A value of attributeType="css" indicates the attribute represents the name of a CSS property on the element referenced by xlink:href.

begin = "value"

Specifies the time for the explicit begin of the animation relative to the completion of document loading. The attribute can contain the following two types of values:

<delay-value>

A <delay-value> is a clock-value measuring presentation time. Presentation time advances at the speed of the presentation. It behaves like the timecode shown on a counter of a tape-deck. It can be stopped, decreased or increased either by user actions, or by the player itself.

<event-value>

The element begins when a certain event occurs. Its value is an element-event. (??? Need definition of "element-event".

The element generating the event must be "in scope". (??? Need definition of "in scope".)

none (the default)

The animation should not start automatically; instead, the animation should only start in response to an explicit event.

For more information, see the SMIL 1.0 specification [[SMIL1](#)].

end = "value"

Specifies the time for the explicit end of the animation relative to the completion of document loading. The attribute can contain the same types of attribute values as the [begin](#) attribute.

dur = "<clock-value>"

Specifies the explicit duration of the animation.

repeatCount = "<number> | indefinite"

Specifies either the <number> of times to loop through the animation or indefinite, which indicates that the animation should be repeated indefinitely. Legal <number> values are integer or fractional iterations, greater than 0. A repeat count of less than 1 will cut short the specified simple duration. At most one of repeatCount or repeatDur should be specified (if both are specified, the repeat duration is defined as the minimum of the specified repeatDur, and the simple duration multiplied by the repeatCount (count). The default value is one.

repeatDur = "<clock-value>"

This causes the element to play repeatedly (loop) for the specified duration. Each repeat iteration

lasts for the simple duration. This attribute has no affect if the duration is 0. A repeat duration less than the simple duration specified in dur will cut short the duration. A repeat duration equal to the simple duration is a no-op. Legal values are clock values greater than 0. In addition, "indefinite" may be specified to indicate that the element should repeat indefinitely (subject to the time container semantics). At most one of repeatCount or repeatDur should be specified (if both are specified, the repeat duration is defined as the minimum of the specified repeatDur, and the simple duration multiplied by the repeatCount (count).

`beginEvent = "<clock-value>"`

Defines the timebase to be the referenced event. Events can be timing events (e.g. `element.onBegin`) as well as any other event supported by the DOM (e.g. `element.onclick`, `document.onscroll`). This supports interactive timing within a document, and reduces the need to add script to "wire" elements together.

The current element will begin when the referenced event is raised (plus any offset value). If the referenced event is never raised, the current element may never be active/displayed. If a negative begin (delay) value is used with this attribute, the element will become active when the event is raised, but will define the local timeline sync according to the offset.

Legal values are:

event-base-value :

`("id(" id-ref ")")? (event-ref) (signed-clock-val)?`

Any defined event name, referenced by the associated element ID. E.g.

`beginEvent="id(button1)(onclick)(3s)"`. Note that if the id-ref is omitted, the current element is assumed. An alternative syntax would follow ECMAScript conventions:

`[id-ref "."]? event-ref ["+"|"- " signed-clock-val]?` E.g.

`beginEvent="button1.onclick+3s"`

"none"

The string "none".

`endEvent = "<clock-value>"`

Defines the end of the active duration to be relative to the referenced event. Events can be timing events (e.g. `element.onBegin`) as well as any other event supported by the DOM (e.g. `element.onClick`, `document.onscroll`). See also the notes on `beginEvent`. The active duration of the current element will end when the referenced event is raised (plus any offset). If the referenced event is never raised, the current element may remain active. If a negative delay value is used with this attribute, it will be ignored. This attribute may be combined with a determinate specification for the element end. If a finite active duration is defined for the element (with the `end` or `dur` attributes and/or `repeatCount/repeatDur`), the element will end at the earlier of the specified duration or the `endEvent` time. This allows authors to specify a maximum duration in addition to and interactive end. If the named event is "none", this element will simply wait to be turned off (e.g. by script). This is not required, and is more easily accomplished by specifying no end or duration for an element; it will default to an indefinite duration, and can be ended via the DOM interfaces. Legal values are:

event-base-value :

`("id(" id-ref ")")? (event-ref) (signed-clock-val)?`

Any defined event name, referenced by the associated element ID. E.g.

`beginEvent="id(button1)(onclick)(3s)"`. Note that if the id-ref is omitted, the current element is assumed. An alternative syntax would follow ECMAScript conventions:

`[id-ref "."]? event-ref ["+"|"- " signed-clock-val]?` E.g. `endEvent="button1.onclick+3s"`

"none"

The string "none".

additive = "true | false"

Controls whether or not the animation is additive. Possible values are "true" and "false". Default is "false", unless another attribute is used which specifies otherwise. This attribute is ignored if the target attribute does not support addition.

accumulate = "true | false"

Controls whether or not the animation is cumulative. Possible values are "true" and "false". Default is "false". This attribute is ignored if the target attribute does not support addition.

fill = "remove | freeze"

Controls whether the final result of the animation should remain past the end of the duration of the animation. remove indicates that the element should revert to its original values at the end of the animation. freeze indicates that the element should retain the final attribute/property value that is in place at the end of the animation. Default is "remove".

values = "<list>"

A semicolon-separated list of one or more values, compatible with the target attribute.

from = "<value>"

Specifies the starting value of the animation. If from is specified, its value must match the to or by type. If the from value is not specified, The value must be compatible with the target attribute.

to = "<value>"

Specifies the ending value of the animation. The argument value must match the attribute type. Specifies a non-additive animation with 2 elements in the values array, the value of the from attribute and the value of the to attribute. The from attribute defaults to the base value of the attribute for the animation when to is used. by = "<value>"

Specifies a relative offset value for the animation. The argument value must match the attribute type. Specifies an additive animation with two elements in the values array, from and by. The from attribute defaults to zero when by is used.

calcMode = "discrete | linear | spline"

Specifies the interpolation mode for the animation. This can take any of the following values:

"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. Treated as "discrete" if the attribute does not support linear interpolation. This is the default calcMode.

"spline"

As for linear, interpolating from one value in the values list to the next by the amount of time elapsed defined by a cubic Bezier spline defined by the interpSpline attribute.

vtimes = "<list>"

A semicolon-separated list of explicit time values defining when each of the values in the values attribute should become the current value. vtimes are specified as decimal fractions between 0 and 1, representing the relative offset into the duration of the <animate> element. If the vtimes attribute is not present, the values in the values attribute are assumed to be equally spaced through the animation duration, with the first value being applied at the very beginning of the animation, and the last value being applied at the very end of the animation. There should be

exactly as many values in the vtimes list as in the values list. If more vtimes are provided than values, then the extra vtimes are ignored. If more values are provided than vtimes, then the extra values are ignored.

interpSpline = "<list>"

A semicolon-separated list of 4-tuples of numbers that specify a non-linear interpolation between each corresponding pair of values. Each 4-tuple x1 y1 x2 y2 defines the control points for a cubic bezier segment within the unit square (0,0) to (1,1). The interpolations are done as follows: Suppose you are at time t, where $t[i] < t < t[i+1]$, where $t[i]$ is the timevalue corresponding to values[i]. Then, calculate a weighting factor $w = y(t)$, where $y(t)$ is the cubic bezier formula for y as a function of t, where the cubic bezier is defined by anchor points (0,0) and (1,1) and control points (x1,y1) and (x2,y2). Finally, the value to use at time t is calculated by: $value = (1-w) * values[i] + w * values[i+1]$.

Attributes defined elsewhere:

[id](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#).

18.2.3 The <animateMotion> element

The <animateMotion> element cause a [<symbol>](#) or [<svg>](#) element. to move along a motion path. (??? Examples coming later.)

```
<!ELEMENT animateMotion (desc?,title?) >
<!ATTLIST animateMotion
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  motionPath CDATA #IMPLIED
  rotate CDATA #IMPLIED
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"
  vtimes CDATA #IMPLIED
  interpSpline CDATA #IMPLIED >
```

Attribute definitions:

xlink:href = "uri-reference"

A [URI reference](#) to the element which is the target of this animation and which therefore will have its position modified over time. The target element must be part of the current SVG document. The target element must be a [<symbol>](#) or [<svg>](#) element. The [<symbol>](#) will be scaled to fit into a viewport whose width and height are one unit in user space. To scale the symbol to the appropriate size, use the [<animateTransform>](#) element and have it reference this [<animateMotion>](#) element.

motionPath = "<path-data>"

The motion path, expressed in the same format and interpreted the same way as the [d=](#) attribute on the [<path>](#) element. The target element moves along the given path.

rotate = "<angle> | auto"

auto is default, which says to rotate object to track the tangent of the motion path. An actual angle value can also be given, which is relative to X-axis of current coordinate system.

values = "<list>"

A semicolon-separated list of one or more values which represents the relative percentage of distance along the motion path. (Note: the default yields constant velocity from the start of the path to the end of the path.) For example: values="0% 30% 90%" or values="0 .3 .9", which says (assuming the default value for vtimes) that you start at the beginning of the path, end up at 30% along the path at time .5, and end up at .9 along the path at time 1. from="" to="" by="" are also available. Values less than zero or greater than one are adjusted to stay between zero and one.

from = "<value>"

Specifies the starting value of the animation as a relative percentage distance along the motion path.

to = "<value>"

Specifies the ending value of the animation as a relative percentage distance along the motion path.

by = "<value>"

Specifies a relative offset value for the animation as a relative percentage distance along the motion path.

Attributes defined elsewhere:

[id](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#), [begin](#), [end](#), [dur](#), [repeatCount](#), [repeatDur](#), [beginEvent](#), [endEvent](#), [fill](#), [calcMode](#), [vtimes](#), [interpSpline](#).

18.2.4 The [<animateTransform>](#) element

The [<animateTransform>](#) element adds a supplemental transformation onto a target element so that it can be translated, scaled, rotated or skewed. (??? Examples coming later.)

```

<!ELEMENT animateTransform (desc?,title?) >
<!ATTLIST animateTransform
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  additive (true | false) "false"
  accumulate (true | false) "false"
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"
  vtimes CDATA #IMPLIED
  interpSpline CDATA #IMPLIED >

```

Attribute definitions:

`xlink:href` = "uri-reference"

A [URI reference](#) to the element which is the target of this animation and which therefore will have its current transformation matrix modified over time. The target element must be part of the current SVG document.

`values` = "<list>"

A semicolon-separated list of one or more values which represents one of the following simple forms of the [transform](#) attribute:

- `translate(x,y)`
- `scale(x[,y])`
- `rotate(angle)`
- `skewX(angle)`
- `skewY(angle)`

All of the values must use the same form; thus, you can specify `values="rotate(30);rotate(90);rotate(180)"` but you cannot specify `values="rotate(30);scale(2);translate(3,5)"`. To mix different types of transformations, use multiple `<animateTransform>` operating on the same target element with `additive="true"`.

`from` = "<value>"

Specifies the starting value of the animation. The value must use the same form as `to` and/or `by`.

to = "<value>"

Specifies the ending value of the animation. The value must use the same form as from and/or by.

by = "<value>"

Specifies a relative offset value for the animation. The value must use the same form as from and/or to.

Attributes defined elsewhere:

[id](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#), [begin](#), [end](#), [dur](#), [repeatCount](#), [repeatDur](#), [beginEvent](#), [endEvent](#), [fill](#), [calcMode](#), [vtimes](#), [interpSpline](#).

18.2.5 The <animateColor> element

The <animateColor> element specifies a color transformation over time. (??? Examples coming later.)

```
<!ELEMENT animateColor (desc?,title?) >
<!ATTLIST animateColor
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  attribute CDATA #REQUIRED
  attributeType (xml|css) 'xml'
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  additive (true | false) "false"
  accumulate (true | false) "false"
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"
  vtimes CDATA #IMPLIED
  interpSpline CDATA #IMPLIED >
```

Attribute definitions:

xlink:href = "uri-reference"

A [URI reference](#) to the element which is the target of this animation and which therefore will

have its color modified over time. The target element must be part of the current SVG document.

values = "<list>"

A semicolon-separated list of color values using any color formulation that is valid for SVG's various color properties. Out of range and negative values are permitted, such as, "rgb(-50, 0, 0)".

from = "<value>"

Specifies the starting color value of the animation. Allowable forms are the same as for the values attribute.

to = "<value>"

Specifies the ending color value of the animation. Allowable forms are the same as for the values attribute.

by = "<value>"

Specifies a relative offset color value for the animation. Allowable forms are the same as for the values attribute.

Attributes defined elsewhere:

[id](#), [system-required](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#), [begin](#), [end](#), [dur](#), [repeatCount](#), [repeatDur](#), [beginEvent](#), [endEvent](#), [fill](#), [calcMode](#), [vtimes](#), [interpSpline](#).

18.2.6 The <animateFlipbook> element

The <animateFlipbook> element produces a flipbook-like animation effect. The list of child <animateFlipbookValue> elements provides indicates the series of elements to make visible one after another as the animation executes. (??? Examples coming later.)

```
<!ELEMENT animateFlipbook (desc?,title?,(animateFlipbookValue)*) >
<!ATTLIST animateFlipbook
  id ID #IMPLIED
  system-required CDATA #IMPLIED
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  fill (remove | freeze) "remove"
  vtimes CDATA #IMPLIED >
```

Attributes defined elsewhere:

[id](#), [system-required](#), [begin](#), [end](#), [dur](#), [repeatCount](#), [repeatDur](#), [beginEvent](#), [endEvent](#), [fill](#), [calcMode](#), [vtimes](#).

```

<!ELEMENT animateFlipbookValue EMPTY >
<!ATTLIST animateFlipbookValue
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

```

Attribute definitions:

`xlink:href` = "uri-reference"

A [URI reference](#) to the element which should be made visible at the appropriate time in this animation. The target elements must be part of the current SVG document.

Attributes defined elsewhere:

[id](#), [xmlns:xlink](#), [xlink:type](#), [xlink:role](#), [xlink:title](#), [xlink:show](#), [xlink:actuate](#).

18.3 Animation example using the SVG DOM

The following example shows a simple animation:

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG August 1999//EN"
  "http://www.w3.org/Graphics/SVG/SVG-19990812.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 document should take up a rectangle of size 4inches by 3inches. The `viewBox` attribute indicates that the initial coordinate system should have (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.

19 Metadata

Contents

- [19.1 Introduction](#)
- [19.2 The SVG Metadata Schema](#)
- [19.3 An Example](#)

19.1 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](#)

Metadata within an SVG document should be expressed in the appropriate RDF namespaces and should be 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:

- Content creators should refer to [W3C Metadata Recommendations and activities](#) when deciding which metadata schema to use in their documents.
- Content creators should 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.

19.2 The SVG Metadata Schema

(This schema has not yet been defined. Here are some candidate attributes for the schema: MeetsAccessibilityGuidelines, UsesDynamicElements, ListOfExtensionsUsed, ListOfICCProfilesUsed, ListOfFontsUsed, ListOfImagesUsed, ListOfForeignObjectsUsed, ListOfExternalReferences.)

19.3 An Example

Here is an example of how metadata can be included in an SVG document. The example uses the Dublin Core version 1.0 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-19990812.dtd'>
  <desc>Floor layout for MyCompany office space</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/1.0/"
      xmlns:svgmetadata = "http://www.w3.org/..." >
      <rdf:Description about=""
        dc:title="MyCompany Floor Layout"
        dc:description="Floor layout for MyCompany office space"
        dc:publisher="MyCompany Incorporated"
        dc:date="1999-03-03"
        dc:format="image/svg"
        dc:language="en" >
        <dc:creator>
          <rdf:Bag>
            <rdf:li>Billy Potts</rdf:li>
            <rdf:li>Mary Graham</rdf:li>
          </rdf:Bag>
        </dc:creator>
        <svgmetadata:General MeetsAccessibilityGuidelines="true"/>
      </rdf:Description>
    </rdf:RDF>
  </metadata>
</svg>
```

[Download this example](#)

20 Backwards Compatibility

A user agent (UA) might not have the ability to process and view SVG documents. The following list outlines two of the backwards compatibility scenarios associated with SVG documents:

- For XML grammars with the ability to embed SVG documents, 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 (??? add links)) 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 an SVG document, 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 document is included via a URL reference. With some parent XML grammars it will also be possible to include an SVG document inline within the same file as its parent grammar.)

```
<?xml version="1.0" standalone="yes"?>
<smil>
  <body>
    <!-- The SMIL <switch> element will process the
         first child element which tests true and skip
         past all others. -->
    <switch>
      <!-- The system-required attribute tests to see if
           the user agent supports SVG. If true, then
           render the file drawing.svg. -->
      <ref system-required="http://www.w3.org/Graphics/SVG/SVG-19990812.dtd"
          type="image/svg" src="drawing.svg" />
      <!-- Else, render the alternate image. -->
      
    </switch>
  </body>
</smil>
```

[Download this example](#)

- For HTML 4.0, SVG drawings should be embedded using the `<object>` element. The alternate representation should be included as the content of the `<object>` element. In this case, the SVG document 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. -->
      
    </object>
  </body>
```

</html>

21 Extensibility

Contents

- [21.1 Foreign Namespaces and Private Data](#)
- [21.2 Embedding Foreign Object Types](#)
- [21.3 The system-required attribute](#)

21.1 Foreign Namespaces and Private Data

SVG allows inclusion of elements from foreign namespaces anywhere with the SVG document tree. 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 document 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 document 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-19990812.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"/>
    </myapp:piechart>
  </defs>
</svg>
```

```

    <!-- 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](#)

21.2 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 within the SVG document. 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 an SVG document. Another example is inserting a 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 contains an XHTML document which in turn contains yet another SVG document).

Additionally, there is a capability for alternative representations so that something meaningful will appear in SVG viewing environments which do not have the ability to process a given `<foreignObject>`. To accomplish this, SVG has a `<switch>` element and **system-required** attribute similar to the corresponding facilities within the [SMIL 1.0 Recommendation](#). The rules for `<switch>` are that the first child element whose **system-required** evaluates to "true" will be processed and all others ignored.

Here is an example:

```

<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in"
  xmlns = 'http://www.w3.org/Graphics/SVG/SVG-19990812.dtd'>
  <desc>This example uses the switch element to provide a
  fallback graphical representation of an equation, if
  MathML is not supported.
  </desc>
  <!-- The <switch> element will process the first child element
        whose testing attributes evaluate to true.-->
  <switch>

    <!-- Process the MathML if the system-required attribute
          evaluates to true (i.e., the user agent supports MathML
          embedded within SVG). -->
    <foreignObject
      system-required="http://www.w3.org/TR/REC-MathML-19980407"
      width="100" height="50">
      <!-- MathML 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 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 should 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 layout 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 feature represents an objective, not a requirement.

(The exact mechanism for providing these capabilities hasn't been decided yet. Many details need to be worked out.)

21.3 The system-required attribute

Definition of system-required:

system-required = *feature*

Determines whether the named *feature* is supported by the user agent. If the given feature is supported, then the current element and its children are processed; otherwise, the current element and its children are ignored.

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 19990812).
  The specification for SVG that corresponds to this DTD is available at:

      http://www.w3.org/1999/08/12/WD-SVG-19990812/
-->

<!--===== Generic Attributes =====>

<!ENTITY % graphicsElementEvents
"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
"onunload CDATA #IMPLIED
 onzoom CDATA #IMPLIED ">

<!ENTITY % structured_text
"content CDATA #FIXED 'structured text' ">

<!--===== Document Structure and Grouping =====>

<!ELEMENT svg (defs?,desc?,title?,
             (path|text|rect|circle|ellipse|line|polyline|polygon|
             use|image|svg|g|switch|a)* )>

<!ATTLIST svg
  xmlns CDATA #FIXED 'http://www.w3.org/Graphics/SVG/SVG-19990812.dtd'
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  %documentEvents;
  system-required 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'
allowZoomAndPan (true | false) "true"
contentScriptType CDATA #IMPLIED >

<!ELEMENT g ( (defs?,desc?,title?,
              (path|text|rect|circle|ellipse|line|polyline|polygon|
               use|image|svg|g|switch|a|
               animate|animateTransform|animateColor))* )>

<!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 CDATA #IMPLIED>

<!ELEMENT defs ( (script|style|symbol|marker|clipPath|mask|
                 linearGradient|radialGradient|pattern|filter|cursor|font|
                 animate|animateMotion|animateTransform|animateColor|animateFlipbook|
                 path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|switch)* >

<!ATTLIST defs
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED>

<!--===== Shapes =====>
<!ELEMENT path (desc?,title?,(animate|animateTransform|animateColor))* >
<!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 CDATA #IMPLIED
  d CDATA #REQUIRED
  flatness CDATA #IMPLIED
  nominalLength CDATA #IMPLIED >

<!ELEMENT rect (desc?,title?,(animate|animateTransform|animateColor))* >
<!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 CDATA #IMPLIED
    x CDATA #IMPLIED
    y CDATA #IMPLIED
    width CDATA #REQUIRED
    height CDATA #REQUIRED
    rx CDATA #IMPLIED
    ry CDATA #IMPLIED >

<!ELEMENT circle (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
    cx CDATA "0"
    cy CDATA "0"
    r CDATA #REQUIRED >

<!ELEMENT ellipse (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
    cx CDATA "0"
    cy CDATA "0"
    rx CDATA #REQUIRED
    ry CDATA #REQUIRED >

<!ELEMENT line (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
    x1 CDATA "0"
    y1 CDATA "0"
    x2 CDATA "0"
    y2 CDATA "0" >

<!ELEMENT polyline (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
points CDATA #REQUIRED >

<!ELEMENT polygon (desc?,title?,(animate|animateTransform|animateColor)*) >
<!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 CDATA #IMPLIED
  points CDATA #REQUIRED >

<!--===== Text =====>

<!ELEMENT text (#PCDATA|tspan|textPath|animate|animateTransform|animateColor)* >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED >

<!ELEMENT tspan (#PCDATA|animate|animateColor)* >
<!ATTLIST tspan
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  %graphicsElementEvents;
  system-required CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  dx CDATA #IMPLIED
  dy CDATA #IMPLIED
  dCoordUnits CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

<!ELEMENT textPath (#PCDATA|tspan)* >
<!ATTLIST textPath
  startOffset CDATA "0"
  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|parsed|replace) #FIXED 'parsed'
xlink:actuate (user|auto) #FIXED 'auto'
xlink:href CDATA #REQUIRED >

<!--===== SVG Fonts =====>
<!ELEMENT font (desc?,title?,missingGlyph,(glyph|kern)*) >
<!ATTLIST font
  id ID #IMPLIED
  fontStyle CDATA #IMPLIED
  fontVariant CDATA #IMPLIED
  fontWeight CDATA #IMPLIED
  fontStretch CDATA #IMPLIED
  unicodeRange CDATA #IMPLIED
  unitsPerEm CDATA #REQUIRED
  panose1 CDATA #IMPLIED
  slope CDATA #IMPLIED
  capHeight CDATA #REQUIRED
  xHeight CDATA #REQUIRED
  accentHeight CDATA #IMPLIED
  ascent CDATA #REQUIRED
  descent CDATA #REQUIRED
  horizOriginX CDATA #IMPLIED
  horizOriginY CDATA #IMPLIED
  horizAdvX CDATA #IMPLIED
  vertOriginX CDATA #IMPLIED
  vertOriginY CDATA #IMPLIED
  vertAdvY CDATA #IMPLIED
  bbox CDATA #REQUIRED
  baseline CDATA #REQUIRED
  centerline CDATA #REQUIRED
  mathline CDATA #REQUIRED
  topline CDATA #REQUIRED
  fullFontName CDATA #IMPLIED
  underlinePosition CDATA #IMPLIED
  underlineThickness CDATA #IMPLIED
  strikethroughPosition CDATA #IMPLIED
  strikethroughThickness CDATA #IMPLIED
  overlinePosition CDATA #IMPLIED
  overlineThickness CDATA #IMPLIED >

<!ELEMENT glyph (defs?,desc?,title?,
  (path|text|rect|circle|ellipse|line|polyline|polygon|
  use|image|g|switch)*)>
<!ATTLIST glyph
  unicode CDATA #REQUIRED
  glyphName CDATA #IMPLIED
  vertTextOrient CDATA #IMPLIED
  arabic CDATA #IMPLIED
  han CDATA #IMPLIED
  horizAdvX CDATA #IMPLIED
  vertAdvY CDATA #IMPLIED
  bbox CDATA #IMPLIED >

<!ELEMENT missingGlyph (defs?,desc?,title?,
  (path|text|rect|circle|ellipse|line|polyline|polygon|
  use|image|g|switch)*)>
<!ATTLIST missingGlyph
  horizAdvX CDATA #IMPLIED

```

```

vertAdvY CDATA #IMPLIED
bbox CDATA #IMPLIED >

<!ELEMENT kern EMPTY >
<!ATTLIST kern
  u1 CDATA #IMPLIED
  g1 CDATA #IMPLIED
  u2 CDATA #IMPLIED
  g2 CDATA #IMPLIED
  k CDATA #REQUIRED >

<!--===== Graphics Referencing Elements =====>

<!ELEMENT use (desc?,title?,(animate|animateTransform|animateColor)* ) >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

<!ELEMENT image (desc?,title?,(animate|animateTransform)* ) >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #REQUIRED
  height CDATA #REQUIRED
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

<!--===== Symbols and Markers =====>

```

```

<!ELEMENT symbol (defs?,desc?,title?,
    (path|text|rect|circle|ellipse|line|polyline|polygon|
    use|image|svg|g|switch|a|
    animate|animateTransform|animateColor)*>
<!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' >

<!ELEMENT marker (defs?,desc?,title?,
    (path|text|rect|circle|ellipse|line|polyline|polygon|
    use|image|svg|g|switch|a|
    animate|animateTransform|animateColor)*>
<!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) "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
    %graphicsElementEvents;
    system-required 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 =====>

<!ELEMENT clipPath (desc?,title?,

```

```

                (path|text|rect|circle|ellipse|line|polyline|polygon|
                 use)* ) >
<!ATTLIST clipPath
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED >

<!ELEMENT mask (defs?,desc?,title?,
                (path|text|rect|circle|ellipse|line|polyline|polygon|
                 use|image|svg|g|switch|a|
                 animate)* )>

<!ATTLIST mask
  id ID #IMPLIED
  xml:lang NMTOKEN #IMPLIED
  xml:space (default|preserve) #IMPLIED
  class NMTOKENS #IMPLIED
  style CDATA #IMPLIED
  maskUnits (userSpace | objectBoundingBox) "userSpace"
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  width CDATA #IMPLIED
  height CDATA #IMPLIED >

<!--===== Built-in Types of Paint =====>

<!ELEMENT linearGradient (stop|animate|animateTransform)* >
<!ATTLIST linearGradient
  id ID #IMPLIED
  gradientUnits (userSpace | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  x1 CDATA #IMPLIED
  y1 CDATA #IMPLIED
  x2 CDATA #IMPLIED
  y2 CDATA #IMPLIED
  spreadMethod (pad | reflect | repeat) "pad">

<!ELEMENT radialGradient (stop|animate|animateTransform)* >
<!ATTLIST radialGradient
  id ID #IMPLIED
  gradientUnits (userSpace | objectBoundingBox) 'userSpace'
  gradientTransform CDATA #IMPLIED
  cx CDATA #IMPLIED
  cy CDATA #IMPLIED
  r CDATA #IMPLIED
  fx CDATA #IMPLIED
  fy CDATA #IMPLIED>

<!ELEMENT stop (animate|animateColor)* >
<!ATTLIST stop
  id ID #IMPLIED
  style CDATA #IMPLIED
  offset CDATA #REQUIRED >

<!ELEMENT pattern (defs?,desc?,title?,
                  (path|text|rect|circle|ellipse|line|polyline|polygon|
                   use|image|svg|g|switch|a|
                   animate|animateTransform)* )>
<!ATTLIST pattern

```

```

id ID #IMPLIED
xml:lang NMTOKEN #IMPLIED
xml:space (default|preserve) #IMPLIED
class NMTOKENS #IMPLIED
style CDATA #IMPLIED
patternUnits (userSpace | 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' >

<!--===== Linking =====>

<!ELEMENT a (defs?,desc?,title?,
             (path|text|rect|circle|ellipse|line|polyline|polygon|
              use|image|svg|g|switch|a)*)>

<!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|parsed|replace) #FIXED 'replace'
  xlink:actuate (user|auto) #FIXED 'user'
  xlink:href CDATA #REQUIRED >

<!--===== Animation =====>

<!ELEMENT animate (desc?,title?) >
<!ATTLIST animate
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  attribute CDATA #REQUIRED
  attributeType (xml|css) 'xml'
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  additive (true | false) "false"
  accumulate (true | false) "false"
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED

```

```

by CDATA #IMPLIED
calcMode (discrete | linear | spline) "discrete"
vtimes CDATA #IMPLIED
interpSpline CDATA #IMPLIED >

<!ELEMENT animateMotion (desc?,title?) >
<!ATTLIST animateMotion
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  motionPath CDATA #IMPLIED
  rotate CDATA #IMPLIED
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"
  vtimes CDATA #IMPLIED
  interpSpline CDATA #IMPLIED >

<!ELEMENT animateTransform (desc?,title?) >
<!ATTLIST animateTransform
  id ID #IMPLIED
  system-required CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #IMPLIED
  begin CDATA #IMPLIED
  end CDATA #IMPLIED
  dur CDATA #IMPLIED
  repeatCount CDATA "1"
  repeatDur CDATA #IMPLIED
  beginEvent CDATA #IMPLIED
  endEvent CDATA #IMPLIED
  additive (true | false) "false"
  accumulate (true | false) "false"
  fill (remove | freeze) "remove"
  values CDATA #IMPLIED
  from CDATA #IMPLIED
  to CDATA #IMPLIED
  by CDATA #IMPLIED
  calcMode (discrete | linear | spline) "discrete"

```

```

    vtimes CDATA #IMPLIED
    interpSpline CDATA #IMPLIED >

<!ELEMENT animateColor (desc?,title?) >
<!ATTLIST animateColor
    id ID #IMPLIED
    system-required CDATA #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|parsed|replace) #FIXED 'parsed'
    xlink:actuate (user|auto) #FIXED 'auto'
    xlink:href CDATA #IMPLIED
    attribute CDATA #REQUIRED
    attributeType (xml|css) 'xml'
    begin CDATA #IMPLIED
    end CDATA #IMPLIED
    dur CDATA #IMPLIED
    repeatCount CDATA "1"
    repeatDur CDATA #IMPLIED
    beginEvent CDATA #IMPLIED
    endEvent CDATA #IMPLIED
    additive (true | false) "false"
    accumulate (true | false) "false"
    fill (remove | freeze) "remove"
    values CDATA #IMPLIED
    from CDATA #IMPLIED
    to CDATA #IMPLIED
    by CDATA #IMPLIED
    calcMode (discrete | linear | spline) "discrete"
    vtimes CDATA #IMPLIED
    interpSpline CDATA #IMPLIED >

<!ELEMENT animateFlipbook (desc?,title?,(animateFlipbookValue)*) >
<!ATTLIST animateFlipbook
    id ID #IMPLIED
    system-required CDATA #IMPLIED
    begin CDATA #IMPLIED
    end CDATA #IMPLIED
    dur CDATA #IMPLIED
    repeatCount CDATA "1"
    repeatDur CDATA #IMPLIED
    beginEvent CDATA #IMPLIED
    endEvent CDATA #IMPLIED
    fill (remove | freeze) "remove"
    vtimes CDATA #IMPLIED >

<!ELEMENT animateFlipbookValue EMPTY >
<!ATTLIST animateFlipbookValue
    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|parsed|replace) #FIXED 'parsed'
    xlink:actuate (user|auto) #FIXED 'auto'
    xlink:href CDATA #REQUIRED >

```

```

<!--===== Defining Scripts and Declaring Styles =====>

<!ELEMENT script (#PCDATA)* >
<!ATTLIST script
  language CDATA #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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

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

<!--===== Custom cursors =====>

<!ELEMENT cursor (desc?,title?) >
<!ATTLIST cursor
  id ID #IMPLIED
  system-required CDATA #IMPLIED
  x CDATA "0"
  y CDATA "0"
  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|parsed|replace) #FIXED 'parsed'
  xlink:actuate (user|auto) #FIXED 'auto'
  xlink:href CDATA #REQUIRED >

<!--===== Extensibility =====>

<!ELEMENT switch (defs?,desc?,title?,
  (path|text|rect|circle|ellipse|line|polyline|polygon|
  use|image|svg|g|switch|a|foreignObject)*)>
<!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 CDATA #IMPLIED>

<!ELEMENT foreignObject (#PCDATA)* >
<!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 CDATA #IMPLIED
  x CDATA #IMPLIED

```

```

y CDATA #IMPLIED
width CDATA #REQUIRED
height CDATA #REQUIRED
%structured_text; >

<!--===== Filter Effects =====>

<!ENTITY % filter_node_attributes
"in CDATA #IMPLIED
nodeId 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">

<!ELEMENT filter (feBlend|feFlood|
feColorMatrix|feComponentTransfer|
feComposite|feDiffuseLighting|feDisplacementMap|
feGaussianBlur|feImage|feMerge|
feMorphology|feOffset|feSpecularLighting|
feTile|feTurbulence)* >
<!ATTLIST filter
id ID #IMPLIED
xml:lang NMTOKEN #IMPLIED
xml:space (default|preserve) #IMPLIED
filterUnits (userSpace | objectBoundingBox) "userSpace"
x CDATA #IMPLIED
y CDATA #IMPLIED
width CDATA #IMPLIED
height CDATA #IMPLIED
filterRes CDATA #IMPLIED>

<!ELEMENT feBlend EMPTY >
<!ATTLIST feBlend
%filter_node_attributes;
mode (normal | multiple | screen | darken | lighten) "normal"
in2 CDATA #REQUIRED>

<!ELEMENT feFlood (animateColor)* >
<!ATTLIST feFlood
%filter_node_attributes;
style CDATA #IMPLIED>

<!ELEMENT feColorMatrix (animate)* >
<!ATTLIST feColorMatrix
%filter_node_attributes;
type CDATA #REQUIRED
values CDATA #IMPLIED>

<!ELEMENT feComponentTransfer (feFuncR?,feFuncG?,feFuncB?,feFuncA?) >
<!ATTLIST feComponentTransfer
%filter_node_attributes;>

<!ELEMENT feFuncR (animate)* >
<!ATTLIST feFuncR
%component_transfer_function_attributes;>

<!ELEMENT feFuncG (animate)* >
<!ATTLIST feFuncG
%component_transfer_function_attributes;>

```

```

<!ELEMENT feFuncB (animate)* >
<!ATTLIST feFuncB
  %component_transfer_function_attributes;>

<!ELEMENT feFuncA (animate)* >
<!ATTLIST feFuncA
  %component_transfer_function_attributes;>

<!ELEMENT feComposite EMPTY >
<!ATTLIST feComposite
  %filter_node_attributes;
  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|animateColor))* >
<!ATTLIST feDiffuseLighting
  %filter_node_attributes;
  resultScale CDATA #IMPLIED
  surfaceScale CDATA #IMPLIED
  diffuseConstant CDATA #IMPLIED
  lightColor CDATA #IMPLIED>

<!ELEMENT feDistantLight (animate)* >
<!ATTLIST feDistantLight
  azimuth CDATA #IMPLIED
  elevation CDATA #IMPLIED>

<!ELEMENT fePointLight (animate)* >
<!ATTLIST fePointLight
  x CDATA #IMPLIED
  y CDATA #IMPLIED
  z CDATA #IMPLIED>

<!ELEMENT feSpotLight (animate)* >
<!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)* >
<!ATTLIST feDisplacementMap
  %filter_node_attributes;
  scale CDATA #IMPLIED
  xChannelSelector (R | G | B | A) "A"
  yChannelSelector (R | G | B | A) "A"
  in2 CDATA #REQUIRED>

<!ELEMENT feGaussianBlur (animate)* >
<!ATTLIST feGaussianBlur
  %filter_node_attributes;
  stdDeviation CDATA #IMPLIED>

<!ELEMENT feImage (animateTransform)* >
<!ATTLIST feImage
  nodeId CDATA #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|parsed|replace) #FIXED 'parsed'

```

```
xlink:actuate (user|auto) #FIXED 'auto'  
xlink:href CDATA #REQUIRED  
transform CDATA #IMPLIED>  
  
<!ELEMENT feMerge (feMergeNode)* >  
<!ATTLIST feMerge  
  %filter_node_attributes;>  
  
<!ELEMENT feMergeNode EMPTY >  
<!ATTLIST feMergeNode  
  in CDATA #IMPLIED>  
  
<!ELEMENT feMorphology (animate)* >  
<!ATTLIST feMorphology  
  %filter_node_attributes;  
  operator (erode | dilate) "erode"  
  radius CDATA #IMPLIED>  
  
<!ELEMENT feOffset (animate)* >  
<!ATTLIST feOffset  
  %filter_node_attributes;  
  dx CDATA #IMPLIED  
  dy CDATA #IMPLIED>  
  
<!ELEMENT feSpecularLighting ((feDistantLight|fePointLight|feSpotLight),(animate|animateColor))* >  
<!ATTLIST feSpecularLighting  
  %filter_node_attributes;  
  surfaceScale CDATA #IMPLIED  
  specularConstant CDATA #IMPLIED  
  specularExponent CDATA #IMPLIED  
  lightColor CDATA #IMPLIED>  
  
<!ELEMENT feTile EMPTY >  
<!ATTLIST feTile  
  %filter_node_attributes;>  
  
<!ELEMENT feTurbulence (animate)* >  
<!ATTLIST feTurbulence  
  %filter_node_attributes;  
  baseFrequency CDATA #IMPLIED  
  numOctaves CDATA #IMPLIED  
  type (fractalNoise | turbulence) "turbulence">
```

Appendix B: SVG's Document Object Model (DOM)

Contents

- [B.1 SVG DOM Overview](#)
- [B.2 Naming Conventions](#)
- [B.3 Objects related to SVG documents](#)

This appendix is normative.

B.1 SVG DOM Overview

The SVG DOM has the following general characteristics:

- It supports all appropriate and relevant facilities defined by the two documents [Document Object Model \(DOM\) Level 1 Specification](#) and [Document Object Model \(DOM\) Level 2 Specification](#)
- Wherever possible, the SVG DOM maintains consistency with the DOM for HTML 4.0, which was defined initially in [Document Object Model \(HTML\) Level 1](#) of the [DOM Level 1 Specification](#) and which has been enhanced in various ways in the [DOM Level 2 Specification](#)
- In those cases where the above two approaches do not provide sufficient capabilities, SVG-specific DOM facilities are provided.

In particular, the following should be noted:

- The SVG DOM supports the CSS DOM facilities described in the [DOM Level 2 Specification](#) document
- The SVG DOM supports the event handling facilities described in the [DOM Level 2 Specification](#) document

B.2 Naming Conventions

The SVG DOM follows similar naming conventions to the [HTML DOM Level 1](#).

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.

The return value of an attribute that has a data type that is a value list is always capitalized, independent of the case of the value in the source document. For example, if the value of the align attribute on a P element is "left" then it is returned as "Left". For attributes with the CDATA data type, the case of the return value is that given in the source document.

B.3 Objects related to SVG documents

Interface *SVGDocument*

An *SVGDocument* is the root of the SVG hierarchy and holds the entire content. Beside providing access to the hierarchy, it also provides some convenience methods for accessing certain sets of information from the document.

IDL Definition

```
interface SVGDocument : Document {

    // Same meanings as in HTML DOM Level 1
    attribute DOMString          title;
    readonly attribute DOMString  referrer;
    readonly attribute DOMString  domain;
    readonly attribute DOMString  URL;
    void                          open();
    void                          close();
    void                          write(in DOMString text);
    void                          writeln(in DOMString text);
    Element                       getElementById(in DOMString elementId);

    // Methods to eliminate flicker in animations.
    unsigned long suspend_redraw(in unsigned long max_wait_milliseconds);
    void          unsuspend_redraw(in unsigned long suspend_handle_id)
                raises(DOMException);
    void          unsuspend_redraw_all();

    // The following utility methods for matrix arithmetic will be
    // available from the DOM. (Details not yet available.)
    // matrix += Vector2D          - translation
    // matrix -= Vector2D          - translation
    // matrix *= number            - scaling
    // matrix /= number            - scaling
    // matrix *= Vector2D          - non-orthogonal scaling
    // matrix /= Vector2D          - non-orthogonal scaling
    // matrix *= Matrix2D          - matrix concatenation
    // matrix == number            - test scaling matrix identity
    // matrix == Vector2D          - test for simple scaling
    // matrix.FSimple()            - transformation is an offset/scale
    // matrix.Rotate(angle)        - rotation
    // matrix.Rotate(Vector2D)      - rotation defined by vector (atan(y/x))
    // matrix.FlipX()              - flip +x <-> -x
    // matrix.FlipY()              - flip +y <-> -y
    // matrix.SkewX()              - skew in direction of X axis
    // matrix.SkewY()              - skew in direction of Y axis
    // matrix.Inverse()            - invert the matrix
};
```

Attributes

title

The title of a document as specified by the title sub-element of the svg element that encompasses the document (i.e., <svg><defs><title>Here

```
is the title</title></defs><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

open

Note. This method and the ones following allow a user to add to or replace the structure model of a document using strings of unparsed SVG. Open a document stream for writing. If a document exists in the target, this method clears it.

This method has no parameters.

This method returns nothing.

This method raises no exceptions.

close

Closes a document stream opened by `open()` and forces rendering.

This method has no parameters.

This method returns nothing.

This method raises no exceptions.

write

Write a string of text to a document stream opened by `open()`. The text is parsed into the document's structure model.

Parameters

`text` The string to be parsed into some structure in the document structure model.

This method returns nothing.

This method raises no exceptions.

writeln

Write a string of text followed by a newline character to a document stream opened by `open()`. The text is parsed into the document's structure model.

Parameters

`text` The string to be parsed into some structure in the document structure model.

This method returns nothing.

This method raises no exceptions.

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.

`suspend_redraw`

Takes a time-out value which indicates that redraw should not occur until: (a) the corresponding `unsuspend_redraw(suspend_handle_id)` call has been made, (b) an `unsuspend_redraw_all()` call has been made, or (c) its timer has timed out. In environments that do not support interactivity (e.g., print media), then redraw should not be suspended. `suspend_handle_id = suspend_redraw(max_wait_milliseconds)` and `unsuspend_redraw(suspend_handle_id)` should 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 = suspend_redraw(max_wait_milliseconds)` and follow the changes with a method call similar to `unsuspend_redraw(suspend_handle_id)`. Note that multiple `suspend_redraw` calls can be used at once and that each such method call is treated independently of the other `suspend_redraw` 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 `suspend_redraw()` call. This value should be passed as the parameter to the corresponding `unsuspend_redraw()` method call.

This method raises no exceptions.

`unsuspend_redraw`

Cancels a specified `suspend_redraw()` by providing a unique `suspend_handle_id`.

Parameters

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

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.

`unsuspend_redraw_all`

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

Parameters

None.

Return Value

None.

This method raises no exceptions.

Interface *SVGElement*

All SVG element interfaces derive from this class.

IDL Definition

```
interface SVGElement : Element {
    readonly attribute SVGDocument ownerSVGDocument;
    CSSStyleDeclaration style;
};
```

Interface *SVGPathElement*

Corresponds to the `<path>` element.

IDL Definition

```
interface SVGPathElement : SVGElement {
    // 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 functions 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);
SVGPathSeg    insertSVGPathSegBefore(in SVGPathSeg newSVGPathSeg,
                                      in unsigned long index)
                                      raises(DOMException);
SVGPathSeg    replaceSVGPathSeg(in SVGPathSeg newSVGPathSeg,
                                 in unsigned long index)
                                 raises(DOMException);
SVGPathSeg    removeSVGPathSeg(in unsigned long index)
                                 raises(DOMException);
SVGPathSeg    appendSVGPathSeg(in SVGPathSeg newSVGPathSeg)
                                 raises(DOMException);

// This alternate set of functions 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
// may not be 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    insertNormalizedSVGPathSegBefore(in SVGPathSeg newSVGPathSeg,
                                              in unsigned long index)
                                              raises(DOMException);
SVGPathSeg    replaceNormalizedSVGPathSeg(in SVGPathSeg newSVGPathSeg,
                                          in unsigned long index)
                                          raises(DOMException);
SVGPathSeg    removeNormalizedSVGPathSeg(in unsigned long index)
                                          raises(DOMException);
SVGPathSeg    appendNormalizedSVGPathSeg(in SVGPathSeg newSVGPathSeg)
                                          raises(DOMException);
};

```

Attributes

Details will be provided later.

Methods

Details will be provided later.

Interface *SVGPathSeg*

Corresponds to a single segment within a path data specification.

IDL Definition

```

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_SWEEP_ABS            = 10; // A
const unsigned short kSVG_PATHSEG_ARC_SWEEP_REL           = 11; // a
const unsigned short kSVG_PATHSEG_ARC_VECTOR_ABS          = 12; // B
const unsigned short kSVG_PATHSEG_ARC_VECTOR_REL          = 13; // b
const unsigned short kSVG_PATHSEG_LINETO_HORIZONTAL_ABS    = 14; // H
const unsigned short kSVG_PATHSEG_LINETO_HORIZONTAL_REL    = 15; // h
const unsigned short kSVG_PATHSEG_LINETO_VERTICAL_ABS      = 16; // V
const unsigned short kSVG_PATHSEG_LINETO_VERTICAL_REL      = 17; // v
const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_SMOOTH_ABS = 18; // S
const unsigned short kSVG_PATHSEG_CURVETO_CUBIC_SMOOTH_REL = 19; // s
const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_SMOOTH_ABS = 20; // T
const unsigned short kSVG_PATHSEG_CURVETO_QUADRATIC_SMOOTH_REL = 21; // 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.
// (We don't want to define 20 different subclasses to PathSeg -
// our document is long enough already.)
attribute double      x; // end point for pathseg (or center for A/a/B/b)
attribute double      y; // end point for pathseg (or center for A/a/B/b)
attribute double      x0,y0; // for control points (start point for B/b)
attribute double      x1,y1; // for control points (end point for B/b)
attribute double      r1,r2; // radii for A/a/B/b
attribute double      a1,a2,a3; // angles for A/a

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

```

Appendix C: Implementation Notes

Contents

- [C.1 Introduction](#)
- [C.2 Forward and undefined references](#)
- [C.3 Referenced objects are "pinned" to their own coordinate systems](#)
- [C.4 <path> element implementation notes](#)

This appendix is normative.

C.1 Introduction

The following are implementation notes that correspond to features which span multiple sections of the SVG specifications.

C.2 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 should 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](#)).

Invalid references should be treated as if no value were provided for the referencing attribute or

property. For example, if there is no element with ID "BogusReference" in the current document, then **fill="url(#BogusReference)"** would represent an invalid reference. In cases like this, the element should be processed as if no 'fill' property were provided. Where a list of property values are possible and one of the property values is an undefined reference, then process the list as if the reference were removed from the list.

C.3 Referenced objects are "pinned" to their own coordinate systems

When a graphical object is referenced by another graphical object (such as when the referenced graphical object is used as a clipping path), 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.4 <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 should 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.
 - Wherever feasible, all SVG user agents should report all errors to the user. This rule will greatly discourage generation of invalid SVG path data.
- Markers, directionality and zero-length path segments:
 - If markers are specified, then a marker should be 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 should be 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 should use the start point's directionality. If the end point has directionality but the start point doesn't, then the start point should use 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 should be calculated by reflecting the previous path segments second control point relative to the current point. The exact math that should be used 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:

$$\begin{aligned}(\text{newx1}, \text{newy1}) &= (\text{curx} - (\text{oldx2} - \text{curx}), \text{cury} - (\text{oldy2} - \text{cury})) \\ &= (2*\text{curx} - \text{oldx2}, 2*\text{cury} - \text{oldy2})\end{aligned}$$

- A non-positive radius value should be treated as an error.
- Unrecognized contents within a path data stream (i.e., contents that are not part of the path data grammar) should be treated as an error.

Appendix D: Conformance Criteria

Contents

- [D.1 Introduction](#)
- [D.2 Conforming SVG Documents](#)
- [D.3 Conforming SVG Stand-Alone Files](#)
- [D.4 Conforming SVG Included Documents](#)
- [D.5 Conforming SVG Generators](#)
- [D.6 Conforming SVG Interpreters](#)
- [D.7 Conforming SVG Viewers](#)

This appendix is informative, not normative.

D.1 Introduction

Different sets of SVG conformance criteria exist for:

- [Conforming SVG Documents](#)
- [Conforming SVG Stand-Alone Files](#)
- [Conforming SVG Included Documents](#)
- [Conforming SVG Generators](#)
- [Conforming SVG Interpreters](#)
- [Conforming SVG Viewers](#)

D.2 Conforming SVG Documents

An SVG document is a *Conforming SVG Document* if it adheres to the specification described in this document ([Scalable Vector Graphics \(SVG\) Specification](#)) and also:

- is a [well-formed XML document](#)
- if all non-SVG namespace elements and attributes are removed from the given document, 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 should 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 a [Conforming SVG Document](#)
- its outermost XML element is an `svg` element

D.4 Conforming SVG Included Documents

SVG documents can be included within parent XML documents using the XML namespace facilities described in [Namespaces in XML](#).

An SVG document that is included within a parent XML document is a *Conforming Included SVG Document* if the SVG document, 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-19990812.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, 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-19990812.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 Documents](#), [Conforming SVG Stand-Alone Files](#) or [Conforming SVG Included Documents](#)
- does not create non-conforming SVG documents of any of the above types
- conforms to the [SVG accessibility guidelines](#)

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 interchangeability 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 documents. A *Conforming SVG Interpreter* must be able to:

- Successfully parse and process any [Conforming SVG Document](#), (It is not required, however, that every possible SVG feature be supported beyond parsing. Thus, for example, a Conforming SVG Interpreter might bypass the processing of selected SVG elements.)
- If the program allows scripts to run against the SVG document's [Document Object Model](#), then a Conforming SVG Interpreter must support the entire DOM model for SVG defined in this specification
- The XML parser must be able to handle arbitrarily long data streams.

D.7 Conforming SVG Viewers

An SVG viewer is a program which can parse and process an SVG document 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*.

Specific criteria for a *Conforming SVG Viewer*:

- In the typical case where the SVG Viewer is also an SVG Interpreter, then the program must also be a [Conforming SVG Interpreter](#),
- All SVG features described in this specification (including all graphic elements, attributes and properties) must be supported and rendered.
- If display devices are supported, facilities must exist for zooming and panning of standalone SVG documents or SVG documents embedded within parent XML documents.
- If printing devices are supported, SVG documents 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).

- The viewer should receive enough information from the parent environment to determine the device resolution. (In situations where this information is impossible to determine, the parent environment should pass a reasonable value for device resolution which tends to approximate most common target devices.)
- In web browser environments, the viewer must have the ability to search and select text strings within SVG documents.
- If display devices are supported, the viewer must have the ability to select and copy text from an SVG document to the system clipboard
- The viewer must have complete support for an ECMAScript binding of the [SVG Document Object Model](#).
- The viewer must support JPEG and PNG image formats.
- The viewer must support alpha channel blending of the SVG document image onto the target canvas.
- The viewer must support the following W3C Recommendations with regard to SVG documents:
 - complete support for the XML 1.0 specification ([Extensible Markup Language \(XML\) 1.0](#))
 - complete support for inclusion of non-SVG namespaces within an SVG document [Namespaces in XML](#) (Note that data from non-SVG namespaces can be 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](#)

Although anti-aliasing support isn't a strict requirement for a Conforming SVG Viewer, it is highly recommended. Lack of anti-aliasing support will generally result in poor results on display devices.

A higher class concept is that of a *Conforming High-Quality SVG Viewer* which 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.)
- 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
- 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

Appendix E: Accessibility Support

Contents

- [E.1 Accessibility and SVG](#)
- [E.2 SVG Accessibility Guidelines](#)

This appendix is informative, not normative.

E.1 Accessibility and SVG

Drawings done in SVG will be much more accessible than drawings done as image formats 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](#) and/or a [short description](#) in the form of a title. 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.
- The description of [Conforming SVG Generators](#) defines a set of accessibility requirements for SVG authoring applications which will promote the accessibility in SVG documents generated from authoring applications.
- Because of SVG's support of Cascading Style Sheets Level 2 (??? need to add link), there will be the ability to set up personal style sheets to adjust the color contrast of graphic elements.
- Because SVG documents are scalable, people with partial visual impairment will be able to zoom in on graphics for easier viewing.
- (See [Tooltips](#).)

(Additional information on accessibility is forthcoming.)

E.2 SVG Accessibility Guidelines

The definition of a [Conforming SVG Generator](#) requires that it adhere to W3C accessibility authoring guidelines [[ACCESS](#)].

(We are considering an authoring guideline where certain classes of authoring applications would

automatically translate object names such as layer names into the content of <title> or <tooltip> elements. Furthermore, SVG viewers would be required to display the given text string when the mouse was positioned over the SVG viewport. The result is that visually impaired people could "hear" the shape of the drawing by moving the pointing device around and listening to the vocalization of the tooltips.)

Appendix F: Internationalization Support

Contents

- [F.1 Internationalization and SVG](#)
- [F.2 SVG Internationalization Guidelines](#)

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 [Ligatures and alternate glyphs](#). This facility provides:

- access to glyphs which are not defined in Unicode
- the ability to conform to the guidelines for normalizing character data for the purposes of enhanced interoperability (see [[CHARMOD](#)]), while still having precise control over the which glyphs are drawn.

SVG supports:

- Horizontal, left-to-right text found in Roman scripts (see the ['text-advance'](#) property)
- Vertical and vertical-ideographic text (see the ['text-advance'](#) property)
- Arabic bi-directional text (see the ['direction'](#) and ['unicode-bidi'](#) properties)

[SVG fonts](#) support alternate glyphs for [Arabic](#) and [Han](#) text.

F.2 SVG Internationalization Guidelines

(Only sketched out so far.)

Will indicate that SVG generators should follow W3C guidelines for normalizing character data [[CHARMOD](#)] and should use the facilities for [Ligatures and alternate glyphs](#) to override the standard glyphs used to represent normalized character data with specified glyphs.

It also might be valuable to provide a guideline for how to author a single SVG document that supports substitution of character data from different languages.

Appendix G: Sample SVG Files

This appendix is informative, not normative.

Not yet written.

Appendix H: 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 tessellate 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	83%
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 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:

- 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 I. References

Contents

- [I.1 Normative references](#)
- [I.2 Informative references](#)

I.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/ikedai/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>.

[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/>.

[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.

[ICC32]

"ICC Profile Format Specification, version 3.2", 1995.
Available at <ftp://sgigate.sgi.com/pub/icc/ICC32.pdf>.

"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.

[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>.

[SRGB]

"Proposal for a Standard Color Space for the Internet - sRGB", M. Anderson, R. Motta, S. Chandrasekar, M. Stokes.
Available at <http://www.w3.org/Graphics/Color/sRGB.html>.

[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.)

[XHTML10]

"XHTML(tm) 1.0: The Extensible HyperText Markup Language",

Available at <http://www.w3.org/TR/xhtml1/>.

[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/>.

I.2 Informative references

[ACCESS]

"Authoring Tool Accessibility Guidelines 1.0 (working draft)", J. Treviranus, J. Richards, I. Jacobs, C. McCathieNeville, editors, 13 July 1999.

Available at <http://www.w3.org/TR/1999/WAI-AUTOOLS-19990713/>

[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>

[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/1999/WD-DOM-Level-2-19990719>

[SMIL1]

"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", P. Hoschka, editor,
15 June 1998.
Available at <http://www.w3.org/TR/REC-smil/>

[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>

[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>

[WAI-PAGEAUTH]

"WAI Accesibility Guidelines: Page Authoring" for designing accessible documents are
available at:
<http://www.w3.org/TR/WD-WAI-PAGEAUTH>.

Appendix J: Change History

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 \(Filling and Stroking\)](#)
 - Removed 'fill-params' and 'stroke-params'. Instead, use private data via foreign namespaces.
- Changes to [Built-in Types of Paint](#)
 - 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 should help developers create complete implementations and should help industry and the media to judge the quality 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 [Terminology](#).
- 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 represent 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 expressed 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 [Built-in Types of Paint](#).
 - 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 should 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 [Built-in Types of Paint](#)
 - 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 should 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 discontinuous 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>.
- 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 it 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 should 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 progress 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